DISCIPLINARUM SCIENTIA *Disciplinarum Scientia.* **Série: Naturais e Tecnológicas**
Santa Maria, v. 27, n. 3, p. 51-74, 2026
Recebido em: 11.03.2026. Aprovado em: 12.06.2026.
ISSN 2176-462X | DOI: doi.org/10.37779/nt.v27i1.5745

# MÓDULO LLM LOCAL EM SISTEMA WEB PARA GERAÇÃO DE RELATÓRIOS TEXTUAIS VIA PROMPT

## *LOCAL LLM MODULE IN A WEB SYSTEM FOR GENERATING TEXTUAL REPORTS VIA PROMPT*

**Luiz Batista Cardoso[1], Pedro Guilherme Gabriel Maurer[2], Ana Paula Canal[3], Hérysson Rodrigues Figueiredo[4], Robertson Ebling dos Santos[5] e Alexandre de Oliveira Zamberlan[6]**

**RESUMO**

Este trabalho aborda a necessidade de aprimoramento dos sistemas de informação do Laboratório de Práticas da Computação da Universidade Franciscana (UFN), com ênfase na melhoria da interação humano-computador. Alguns sistemas existentes carecem de funcionalidades avançadas para a geração automatizada de relatórios, exigindo a realização de consultas manuais complexas em SQL. Diante desse cenário, foi desenvolvido um módulo de interação textual que integra técnicas de Processamento de Linguagem Natural (PLN) e modelos de linguagem de grande porte (LLMs), capazes de converter perguntas em linguagem natural em consultas SQL, facilitando a obtenção de relatórios e a análise de dados. Para a implementação da solução foram utilizadas as bibliotecas LangChain, Ollama e xmlrpc, além do modelo Llama 3.1 8B. Adicionalmente, empregou-se o *framework* LangChain para a implementação da técnica de Recuperação Aumentada por Geração (RAG). Os resultados obtidos demonstraram desempenho consistente na geração de consultas, especialmente para perguntas curtas e bem estruturadas, quando comparadas às consultas SQL elaboradas manualmente.

**Palavras-chave:** Integração de sistemas; Computação em Nuvem; RPC; Inteligência Artificial; Python.

***ABSTRACT***

*This study addresses the need to enhance the information systems of the Laboratório de Práticas da Computação at Universidade Franciscana (UFN), with a particular emphasis on human-computer interaction. The existing systems lacked advanced capabilities for automated report generation, relying instead on complex manual SQL queries. To address this limitation, a textual interaction module was developed, integrating Natural Language Processing (NLP) techniques and Large Language Models (LLMs) to translate natural language questions into SQL queries, thereby streamlining the report generation process. The implementation employed the Llama 3.1 8B model and leveraged the LangChain framework, together with the Ollama and xmlrpc libraries, to support a Retrieval-Augmented Generation (RAG) architecture. The proposed solution produced consistent and accurate results for short, well-structured queries when compared to traditional manual SQL queries.*

***Keywords:*** *Systems Integration; Cloud Computing; RPC; Artificial Intelligence; Python.*

1 Egresso do Curso de Ciência da Computação UFN. Email: luiz.bcardoso@ufn.edu.br. ORCID: https://orcid.org/0009-0003-7344-4008

2 Egresso do Curso de Ciência da Computação UFN. Email: p.maurer@ufn.edu.br

3 Professora dos cursos de Computação UFN. Email: apc@ufn.edu.br. ORCID: https://orcid.org/0000-0001-6360-1688

4 Professor dos cursos de Computação UFN. Email: herysson.figueiredo@ufn.edu.br. ORCID: https://orcid.org/0009-0009-2615-9785

5 Egresso do Curso de Sistemas de Informação UFN. Email: robertson@ersistemas.info. ORCID: https://orcid.org/0000-0003-0683-6063

6 Professor dos cursos de Computação UFN. Email: alexz@ufn.edu.br. ORCID: https://orcid.org/0000-0002-9815-2031

## 1 INTRODUÇÃO

Os sistemas de informação projetados, desenvolvidos e disponibilizados no Laboratório de Práticas da Computação da Universidade Franciscana (UFN) são aplicados em diversas áreas. No entanto, tais sistemas, até o presente trabalho, apresentam carência de funcionalidades avançadas de interação humano-computador, especialmente no que diz respeito a interfaces de *prompts* para consulta de dados em linguagem natural. Assim, projetar e integrar um módulo que permita a geração de relatórios detalhados com base nas informações armazenadas nos bancos de dados desses sistemas torna-se uma necessidade premente e relevante.

Até o segundo semestre de 2024, sistemas Web como o SIGEP-COMIC (MÜLLER; ZAMBERLAN, 2020) não ofereciam módulos integrados para a geração automatizada de relatórios. Quando os coordenadores desses sistemas precisam de informações específicas, enfrentam um processo quase manual, recorrendo a filtros predefinidos e realizando consultas limitadas. Para melhorar essa situação, o módulo permite que os usuários possam formular suas solicitações de maneira descritiva e em português, facilitando a obtenção de relatórios personalizados e mais detalhados com maior eficiência e menor esforço manual. Para converter consultas em linguagem natural para SQL, é possível utilizar ferramentas de Processamento de Linguagem Natural (PLN), como Meta Llama. Essas ferramentas empregam Inteligência Artificial Generativa para realizar a conversão de texto para SQL. No contexto deste trabalho, um *Large Language Model* (LLM) desempenha papel crucial, facilitando a interpretação da linguagem natural e a geração de consultas SQL, entre outras funções.

O objetivo geral desta pesquisa foi desenvolver e integrar um módulo de interação textual em linguagem natural, via *prompt*, para a geração de relatórios em um sistema Web construído em Python-Django. Como objetivos específicos, assumiram-se: pesquisar, compilar e escrever sobre PLN e LLM; pesquisar bibliotecas de PLN; pesquisar e testar bibliotecas Python para LLM; pesquisar, compilar e escrever sobre trabalhos relacionados; projetar e implementar um módulo de geração de relatórios via *prompt*; definir e aplicar um ambiente de avaliação e teste.

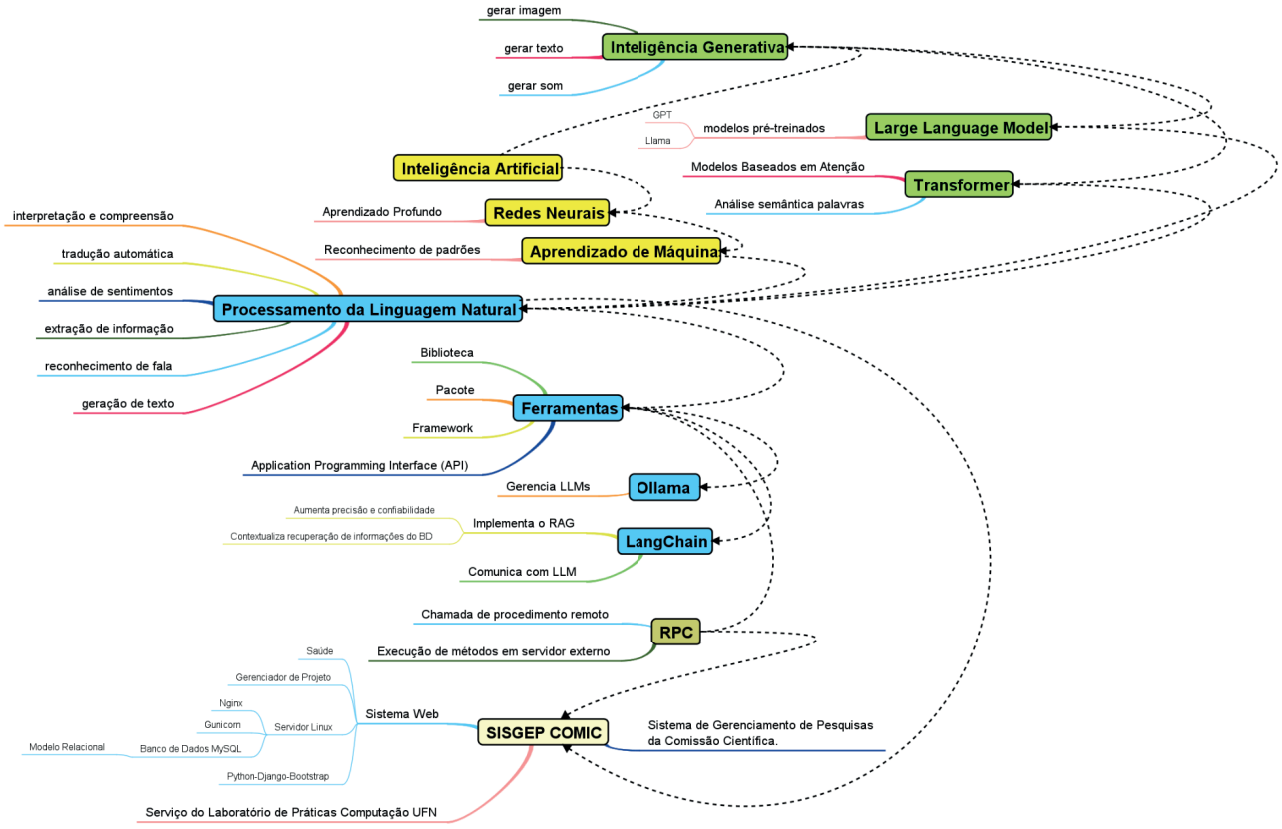
Para auxiliar na compreensão da pesquisa, o texto foi dividido em Referencial Teórico (com conceitos, técnicas, ferramentas e trabalhos relacionados à esta pesquisa); Materiais e Métodos (com informações metodológicas sobre a construção do módulo); Resultados e Discussões (com apontamentos sobre os resultados alcançados e limitações encontradas); Conclusões e Referências Bibliográficas.

## 2 REFERENCIAL TEÓRICO

Nesta seção, abordam-se os tópicos centrais relacionados à proposta, com ênfase nas estratégias utilizadas na construção do projeto. Nas seções, são exploradas áreas fundamentais, como a Inteligência Artificial Generativa, sua implementação e integração em sistemas Web.

A Figura 1 ilustra a relação entre conceitos e recursos (ferramentas) necessários para que o SISGEP-COMIC (MÜLLER; ZAMBERLAN, 2020) possa gerar consultas em linguagem natural escrita. O foco da pesquisa é o Processamento de Linguagem Natural, que depende de subáreas da Inteligência Artificial, como Redes Neurais, Aprendizado de Máquina e Inteligência Artificial Generativa. A IA Generativa, por sua vez, baseia-se em duas estruturas principais: os *Large Language Models*, que são modelos de texto pré-treinados, e os *Transformers*, que constituem a base arquitetural para a aplicação da IA Generativa no PLN. Essas estruturas estão presentes em ferramentas desenvolvidas como OpenAI GPT e Meta Llama.

**Figura 1** - Mapa mental dos conceitos da pesquisa.




Produzido pelos autores.

## 2.1 INTELIGÊNCIA ARTIFICIAL: PROCESSAMENTO DA LÍNGUA NATURAL E APRENDIZADO DE MÁQUINA

Segundo Peter Norvig e Stuart Russell (2016), a IA é uma área da Ciência da Computação que aborda a capacidade das máquinas de executar tarefas que tenha alguma habilidade ou característica do ser humano, criando um conjunto de vantagens como precisão, rapidez e capacidade de gerenciar grande volumes de memória. Nas diversas áreas da IA, existe uma que foca no reconhecimento e na interpretação da língua humana, comumente chamada de PLN (ORACLE, 2024), o qual possibilita

a comunicação entre o ser humano e a máquina, que faz interpretação do que foi falado ou escrito. Sendo utilizado em várias áreas da IA como o aprendizado de máquina e reconhecimento de padrões (MISHRA, 2020). O aprendizado de máquina (*Machine Learning* - ML) é outra subárea da IA, que foca na construção de sistemas que aprendem com base em uma fonte de dados já existente (ORACLE, 2024). Segundo Stryker e Holdsworth (2024), o estudo de como interpretar as diversas formas de como um humano pode escrever (tratar a língua escrita, por exemplo) foi fundamental para o avanço da geração de conteúdo e a popularização dos assistentes virtuais e *chatbots*. Dessa forma, a IA Generativa é uma subcategoria da Inteligência Artificial que combina aprendizado de máquina e um vasto conjunto de dados para produzir novos conteúdos, fundamentando-se no que já existe (MICROSOFT LEARN A, 2024). Essas aplicações utilizam o PLN como parâmetro, permitindo que, ao ser processada, retornem uma resposta baseada no formato ou na escrita. Essa resposta pode manifestar-se como um texto, uma imagem ou um som (MICROSOFT LEARN A, 2024), conforme já apresentado na Figura 1 do mapa mental.

## 2.2 *LARGE LANGUAGE MODELS*

Todas as aplicações que utilizam IA Generativa fundamentam-se em modelos de linguagem, que operam em conjunto com técnicas de aprendizado de máquina e PLN (MICROSOFT LEARN B, 2024). Esses modelos são treinados com enormes quantidades de dados, sendo denominados Modelos de Linguagem de Larga Escala (LLM) (CLOUDFLARE, 2024). Registra-se que há outras fases importantes (VASWANI, 2017), como: Pré-processamento: antes do treinamento, os dados passam por uma limpeza e transformação; Tokenização: o texto é dividido em *tokens*, que podem representar palavras ou partes de palavras, facilitando o processamento e a compreensão do modelo; Filtragem e curadoria: inúmeras vezes, os dados são filtrados para remover conteúdo indesejado ou inadequado, garantindo que o modelo aprenda com informações de qualidade, como se fosse um processo de melhoria ou sintonia (*tuning*).

Com o avanço da tecnologia (como processadores e memórias mais rápidos, sistemas em nuvem, *clusters* computacionais, etc), os *Transformers* originaram LLMs mais sofisticados, como o GPT (*Generative Pre-Trained Transformer*) (OPENAI A, 2024) e o Llama (*Large Language Model Meta AI*) (META, 2024). Esses modelos, desenvolvidos para compreender e gerar texto de maneira mais eficiente, popularizaram o uso da Inteligência Artificial em diversas aplicações, como assistentes virtuais, geração de conteúdo e tradução automática (D'SOUZA, 2023). Segundo a definição sobre os fundamentos da IA Generativa, no Microsoft Learn (MICROSOFT LEARN B, 2024), os modelos *Transformers*, ou Modelos Baseados em Atenção, são arquiteturas treinadas com grandes volumes de texto que permitem uma representação semântica das palavras (ou seja, seus significados e sentidos). Essa abordagem utiliza as relações entre essas palavras para determinar sequências probabilísticas

de texto que fazem sentido. A integração dos *Transformers* está presente na maioria das LLMs e serve como uma forma evolutiva para o entendimento e a geração de novos conteúdos em linguagem natural (VASWANI, 2017). Esses modelos podem ser facilmente implementados em projetos com o auxílio de ferramentas prontas, como as discutidas na próxima seção.

## 2.3 FERRAMENTAS PARA IA GENERATIVA

Para o desenvolvimento integrado com IA Generativa, a linguagem de programação orientada a objetos Python (2024) tem se destacado, fornecendo recursos mais variados, desde bibliotecas até *Application Programming Interface* (API). Dentro desse ecossistema, o principal *framework* para comunicação é o LangChain (CHASE, 2024), que possibilita o desenvolvimento de aplicações utilizando LLMs, oferecendo ferramentas e abstrações que aprimoram a personalização, precisão e relevância das informações geradas pelos modelos. Além disso, essa ferramenta permite que LLMs acessem diferentes conjuntos de dados sem a necessidade de retreinamento (AMAZON LANGCHAIN, 2024).

Por sua vez, o Ollama (2024) é uma ferramenta que simplifica o processo de implementação de instâncias de LLMs, executando localmente. Sua principal funcionalidade é baixar e configurar um servidor com o modelo Llama, na versão 3.1, mais específico que possui 8 bilhões de parâmetros (8B) da Meta (2024). O Ollama Python (2024), é uma biblioteca que permite a integração de modelos de linguagem instanciados no Ollama de maneira simples e eficiente, facilitando a interação com o modelo para requisição e resposta de suas funcionalidades. Com esses recursos, é possível criar um sistema básico de requisição de informações. No entanto, a geração de conteúdo pode ser impactada por grandes volumes de dados, exigindo uma técnica para garantir maior consistência, denominada RAG.

Na geração de conteúdo, quando muitas informações são transmitidas ou solicitadas, a natureza da tecnologia utilizada nas LLMs pode resultar em respostas imprevisíveis (AMAZON RAG, 2024). Isso gera problemas como informações falsas, desatualizadas ou fora de contexto, além de respostas baseadas em fontes inventadas e imprecisões gerais de conteúdos com terminologia ambígua ou semelhante. A Geração Aumentada de Recuperação (*Retrieval-Augmented Generation*) é uma técnica empregada para aumentar a precisão e a confiabilidade das respostas da IA Generativa, utilizando informações de fontes externas (MERRITT, 2024). Esta ferramenta opera por meio da interação direta do modelo de linguagem com um sistema de armazenamento de dados persistente, utilizando consultas para acessar e recuperar informações relevantes (MERRITT, 2024).

Modelos pré-treinados, desenvolvidos a partir de grandes volumes de dados por empresas, tornam o uso acessível ao usuário comum e abrem novas possibilidades para o desenvolvimento de aplicações de uso geral. A utilização de uma LLM local possibilita seu funcionamento *offline*, evitando a coleta de informações pelas empresas que a hospeda (OPENAI B, 2024). Um dos principais fatores que limitam a utilização abrangente das LLMs locais é a infraestrutura computacional.

Segundo o site *Llama AI Model* (META, 2024), os requisitos mínimos de hardware para a operação do modelo incluem um processador de 8 núcleos e pelo menos 16 GB de memória RAM.

## 2.4 REMOTE PROCEDURE CALL

No contexto da computação distribuída, uma Chamada de Procedimento Remoto (*Remote Procedure Call* - RPC) ocorre quando um programa de computador invoca uma sub-rotina ou procedimento para ser executado em um espaço de memória distinto, de forma sincronizada. Geralmente, isso envolve outro computador ou uma rede compartilhada de computadores (NELSON, 1981). Essa interação entre máquinas caracteriza uma arquitetura cliente-servidor, na qual o cliente é quem solicita a execução de um procedimento em outro sistema, atuando este como servidor. Utilizando a biblioteca xmlrpc, é possível implementar o RPC em Python de maneira simples. A teoria RPC foi abordada devido à inviabilidade de executar uma IA Generativa baseada em LLM em servidores Web tradicionais. Conforme discutido na seção sobre IA Generativa, um dos desafios de executar uma LLM local é a alta demanda computacional, que pode causar instabilidade e alta latência nas respostas do servidor (XU, 2005). Uma solução para reduzir essa carga é contratar um serviço especializado, com máquinas externas que atendem ou superam os requisitos necessários para uma implementação mais eficaz. Empresas oferecem esse tipo de serviço, conhecido como computação em nuvem.

Segundo Kolbe (JUNIOR, 2020), computação em nuvem refere-se ao fornecimento de serviços de computação pela Internet. É denominada "nuvem" por não exigir a presença de uma máquina física para o funcionamento do serviço, transmitindo a ideia de algo externo, facilmente acessível e abrangente. A computação em nuvem destaca-se pela fácil escalabilidade e pelo custo-benefício de sua utilização. O processamento das informações das LLMs pode ser realizado em um modelo de "pague conforme o uso" (*pay-as-you-go*), que oferece uma solução mais econômica e confiável do que uma máquina dedicada (McAULEY, 2024).

## 2.5 TRABALHOS CORRELATOS

Os trabalhos correlatos, nesta seção, têm como objetivo contextualizar as pesquisas e tecnologias relevantes para este estudo. O propósito é identificar as possibilidades e desafios que fundamentam o presente trabalho. O trabalho relacionado de Maurer e Zamberlan (2024) visa o desenvolvimento de uma interação humano-computador com linguagem natural sem depender de conexão com a Internet, utilizando LLMs locais para a geração de texto ou voz. Além disso, os autores detalham de forma aprofundada os fundamentos da IA Generativa e como ocorre sua integração em um ecossistema Python. No trabalho de Barros e Martins (2024), a principal relação com esta pesquisa reside na metodologia proposta para a avaliação das respostas geradas por LLMs na produção de código nas linguagens Python e Lua. A partir de testes, são apresentadas as métricas de avaliação, como precisão,

legibilidade e desempenho. O trabalho de Müller e Zamberlan (2020) tem como principal relevância o sistema desenvolvido, que propõe um serviço capaz de gerenciar o processo de submissão, avaliação e acompanhamento de projetos junto à Comissão Científica (COMIC).

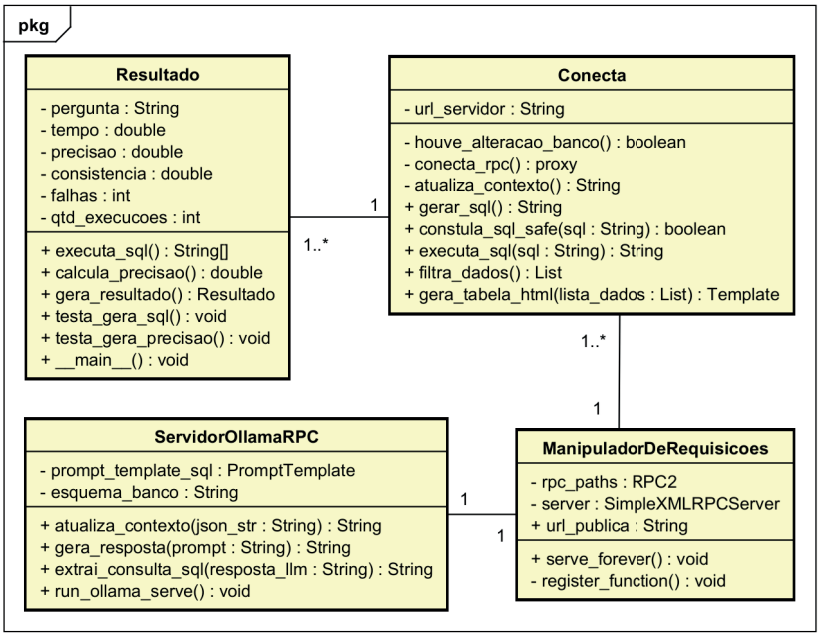
Os estudos como os de Müller e Zamberlan (2020) e de Barros e Martins (2024) apresentam sistemas relevantes que podem ser aplicados diretamente ao contexto desta pesquisa. Por outro lado, o trabalho de Maurer e Zamberlan (2024) fornece *insights* sobre como esse módulo pode ser implementado de maneira eficiente e correta.

## 3 MATERIAIS E MÉTODOS

Para a elaboração do texto, utilizou-se pesquisa de revisão bibliográfica. Já para o desenvolvimento deste projeto, optou-se por uma abordagem de metodologia ágil Scrum, com apoio do método Kanban. As ferramentas utilizadas foram: Linguagem Python; Bibliotecas Python LangChain, Ollama, xmlrpc; Modelo linguagem pré-treinado Llama 3.1 8B; *Framework* LangChain para implementação do RAG. Os encontros para a construção do trabalho foram semanais com o uso de repositório de versionamento de código. O orientador da pesquisa atuou como *Scrum Master*, o primeiro autor atuou como desenvolvedor (*Development Team*), e gestora e administradora do SISGEP-COMIC, como cliente e *Product Owner*. A modelagem do sistema concentra-se nos aspectos funcionais, sendo fundamentada pelo *Product Backlog* e os diagramas UML.

Para representar de forma clara a interação entre os dois servidores, foi elaborado um diagrama de classes, apresentado na Figura 2, com o objetivo de estruturar e explicar as etapas envolvidas na geração de relatórios pelo módulo construído.

**Figura 2** - Diagrama de classes do SISGEP-COMIC e Servidor Ollama.

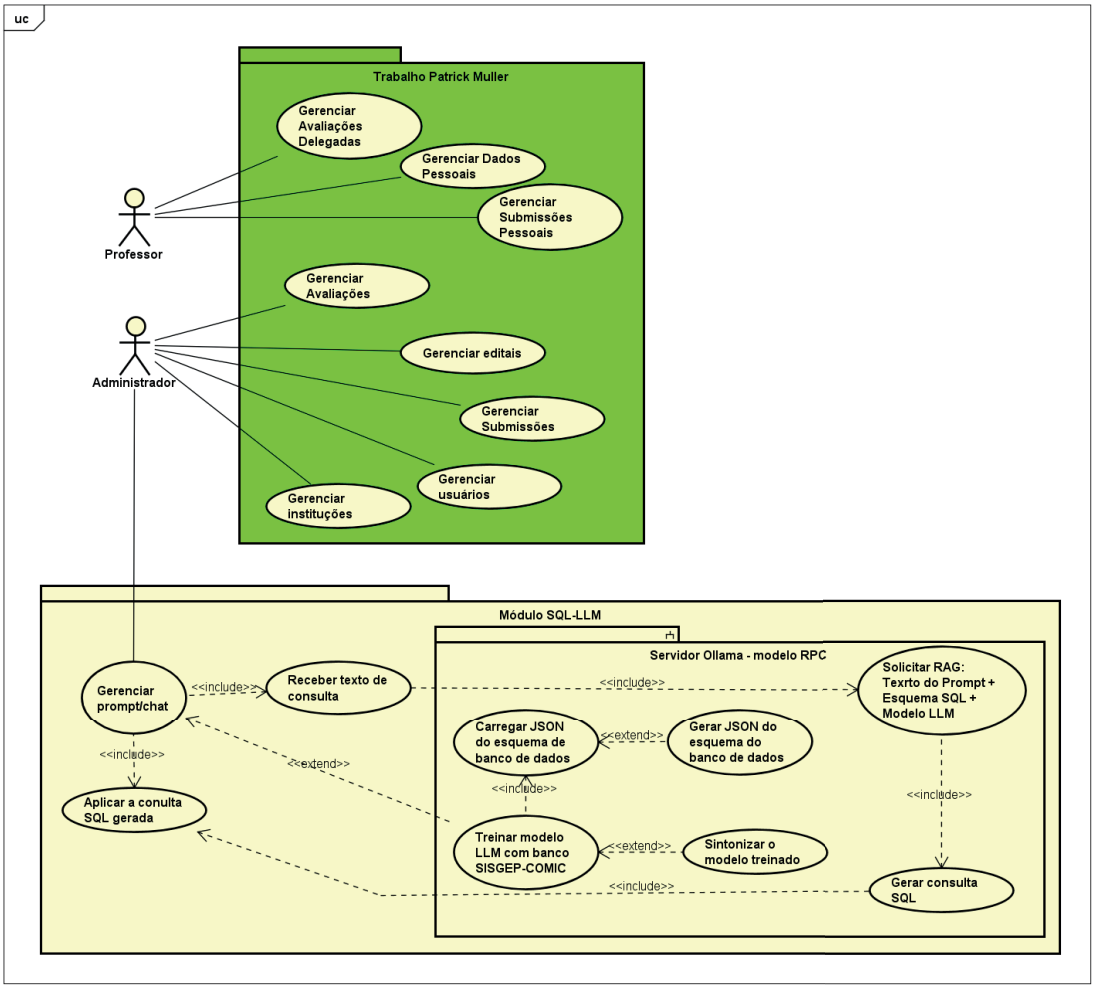



Produzido pelos autores.

No diagrama, a classe *ServidorOllamaRPC* centraliza a lógica de manipulação do LLM, sendo responsável por gerar respostas e atualizar o contexto do RAG. Além disso, a classe possuí um método para realizar a triagem da resposta gerada para determinar se existe uma instrução SQL usando expressões regulares. Para expor esses métodos via RPC, é criada a classe *ManipuladorDeRequisicoes*, que permite o registro de funções e a conexão a partir em uma URL pública. A classe *Conecta* é responsável por estabelecer a conexão com o servidor Ollama, com os métodos de geração de consultas e atualização de contexto executados remotamente via RPC. Os demais métodos são executados localmente, incluindo a verificação de segurança da consulta antes da execução no banco de dados. Para a exibição das informações, dois métodos adicionais são utilizados para filtrar e estruturar os dados em uma tabela HTML. A classe *Resultado* é abordada em mais detalhes no estudo de caso, no qual é descrito sua finalidade e principais atributos.

O diagrama de caso de uso, adaptado de Müller e Zamberlan (2020), representado na Figura 3, descreve a interação entre o sistema e seus usuários. O principal ator neste contexto é o administrador, que possui acesso às funcionalidades do módulo responsável pela geração de relatórios.

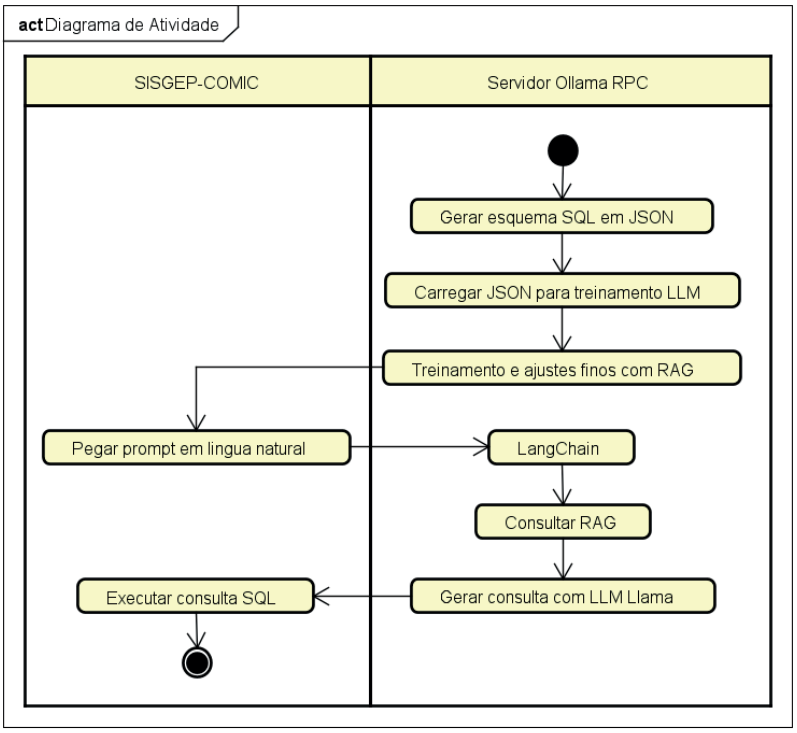
**Figura 3** - Diagrama adaptado de caso de uso SISGEP-COMIC.




Produzido pelos autores com base em Müller e Zamberlan (2020).

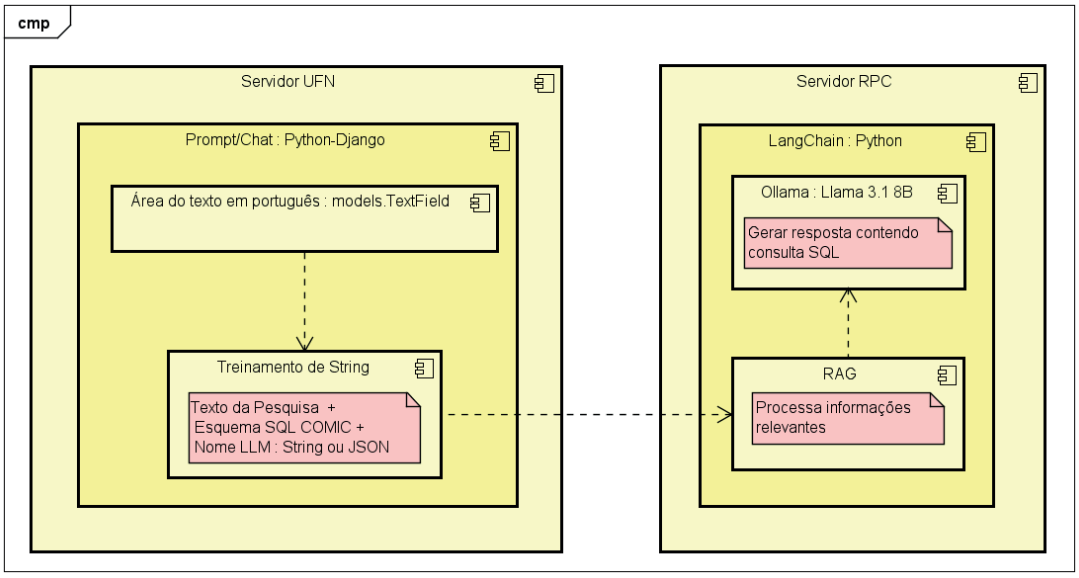
O diagrama de atividade apresentado na Figura 4, ilustra o fluxo de interação entre os dois servidores, detalhando o processo de geração da consulta SQL por meio do *prompt* usando PLN. Já o diagrama de componentes, apresentado na Figura 5, representa esta estrutura, destacando a interação entre os dois servidores, onde um utiliza o *framework* Python-Django do SISGEP-COMIC, e o outro representa o servidor Ollama com a LLM comunicando via RPC.

**Figura 4** - Diagrama de atividade do SISGEP-COMIC e Ollama.




Produzido pelos autores.

**Figura 5** - Diagrama de componentes com os dois servidores.




Produzido pelos autores.

## 3.1 ESTUDO DE CASO: PROCESSOS DE INTEGRAÇÃO E DE AVALIAÇÃO

Para processo de integração, a geração de conteúdo usando IA Generativa é obtida por meio da ferramenta Ollama para instanciar (criar um objeto a partir de classe) LLMs já treinadas, no qual já possuem as tecnologias de PLN e ML necessárias para interpretar requisições via *prompt*. O processamento das requisições do cliente é realizado via métodos, executados no servidor via RPC, que é gerenciada pelo *framework* LangChain. A utilização de contextos na base de conhecimentos do módulo RAG é aplicada para gerar consultas de forma mais confiável, tendo como principal base um arquivo JSON contendo o esquema do banco de dados do SISGEP-COMIC. Por fim, a resposta é enviada ao servidor Web de forma textual, contendo a consulta a ser executada. Após a inserção do *prompt*, ele é enviado por meio de uma chamada RPC para um servidor externo. Esse servidor utilizará a LLM para gerar uma consulta SQL com base na estrutura treinada pelo RAG. A consulta gerada é então retornada ao sistema cliente, onde é aplicada ao sistema gerenciador do banco de dados por meio do *framework* Python-Django. Quando a consulta for executada, o sistema retornará as informações solicitadas e as exibirá de forma estruturada ao usuário.

Para implementação do servidor responsável, por instanciar o Ollama e expor métodos, foi realizada por meio de duas abordagens. A primeira utilizou uma máquina virtual em nuvem, via plataforma Google Colab, com acesso a uma GPU Nvidia T4. Já a segunda, foi executada de maneira local, em uma máquina com sistema operacional Linux, utilizando uma GPU dedicada da Nvidia. Ambas as abordagens utilizam o mesmo código, diferenciando-se apenas na forma de conexão. No Google Colab, a conexão é viabilizada por um túnel via serviço ngrok, permitindo acesso externo ao servidor Ollama. Segundo NGROK (2026), ngrok é uma ferramenta que expõe um servidor local para a Internet de forma rápida e segura, mostrando o andamento de um site para um cliente sem precisar colocar o serviço do site em um servidor real.

Para a avaliação do módulo, uma série de testes foi elaborada para garantir que os dados gerados sejam factuais como forma de resultados. As perguntas de avaliação foram executadas de duas maneiras distintas. A primeira segue o processo tradicional, no qual a consulta é criada manualmente para o SGBD, de forma semelhante ao que já é feito no SISGEP-COMIC. A segunda maneira utiliza o módulo gerador de relatórios, empregando as perguntas como *prompt.*

As métricas de avaliação são: i) tempo de execução; ii) precisão dos dados obtidos; iii) consistência das respostas; iv) taxa de falhas. No primeiro item, o tempo é medido em segundos. Para isso, utiliza-se a biblioteca *time* da linguagem Python, que inicia um cronômetro quando o *prompt* é enviado ao sistema e o interrompe quando a resposta é completamente gerada e exibida ao usuário. No segundo item, a precisão é avaliada com base na correspondência entre os dados retornados pela consulta SQL gerada e os dados de referência presentes no banco. Por exemplo, se todas as informações coincidirem com as de referência, a precisão será de 100%. Essa porcentagem é obtida

pela Equação 1, onde *G* representa o conjunto de dados retornados pela consulta gerada pelo módulo, e *R* é o conjunto de dados corretos da consulta de referência.

Para comparar os dois conjuntos, uma função no servidor recebe as informações de forma estruturada, realizando a comparação e determinando a diferença.

$$P = G/R \quad (1)$$

No terceiro item, é representado o percentual de consistência da precisão obtida a partir de várias execuções. Para calcular esse percentual, utiliza-se a Equação 2, na qual é feita a média das precisões *P* obtidas sobre o número total de *n* consultas realizadas.

$$C = \frac{1}{n}\sum_{i=1}^{n} P_i \quad (2)$$

No quarto e último item, é definido o percentual de falhas para as consultas executadas que não foram bem-sucedidas. A Equação 3, descreve como essa porcentagem será obtida, onde *nFalhas* representa o número de consultas que falharam, e *nTotal* é o número total de consultas executadas.

$$F = nFalhas/nTotal \quad (3)$$

Para medir a taxa de falhas, é necessário definir o que caracteriza uma consulta com falha. Segundo Jalote (1994), uma falha é um comportamento inesperado no software, ocorrido no meio físico, e pode ser determinada pela sua natureza, duração, extensão e valor. Uma consulta com falha pode ocorrer por diversos motivos, como erros de sintaxe ou lógica, ou ainda problemas na execução ou comunicação com o SGBD.

## 4 RESULTADOS E DISCUSSÕES

Nesta seção, são discutidos os resultados obtidos com o desenvolvimento do módulo proposto, bem como seu desempenho em diferentes situações e com base no modelo proposto no estudo de caso.

## 4.1 IMPLEMENTAÇÃO DO SERVIDOR OLLAMA

**Figura 6** - Código dos métodos locais para serem registrados.

```
17 # (RPC) Define a função que atualiza o contexto usando JSON
18 def atualizar_contexto(json_str):
19     global esquema_banco
20     try:
21         print('------ ATUALIZANDO CONTEXTO ------')
22         if isinstance(json_str, dict):
23             esquema_banco = json_str
24         else:
25             esquema_banco = json.loads(json_str)
26         return 'Contexto atualizado com sucesso!'
27     except Exception as e:
28         print(f"Erro ao atualizar o contexto: {str(e)}")
29         return f"Erro ao atualizar o contexto: {str(e)}"
30     finally:
31         print('CONTEXTO ATUAL > ',esquema_banco)
32         print('---- FIM ATUALIZAÇÃO CONTEXTO ----')
33
34 # (RPC) Define a função que gera consultas SQL
35 def gerar_resposta(prompt_usuario: str) -> str:
36     print('------ NOVA REQUISIÇÃO DO USUÁRIO ------')
37     print('PERGUNTA > ', prompt_usuario)
38     try:
39         # Gerar a resposta
40         llm_response = sql_chain.invoke({
41             "input_text"  : prompt_usuario,
42             "schema_info" : json.dumps(esquema_banco)
43         })
44         print('RESPOSTA > ', llm_response)
45
46         # Extrair o SQL usando expressões regulares
47         sql_query = extrair_consulta_sql(llm_response)
48         print('QUERYSQL > ', sql_query)
49
50         return sql_query
51     except Exception as e:
52         error_msg = f"Erro ao gerar resposta: {str(e)}"
53         print(error_msg)
54         return error_msg
55     finally:
56         print('------- FIM REQUISIÇÃO DO USUÁRIO ------')
```

Produzido pelos autores.

Na Figura 6, são ilustrados os dois principais métodos de chamada remota ao servidor Ollama. O primeiro é responsável por gerar consultas SQL a partir de perguntas em linguagem natural, recebidas por parâmetro. O segundo método tem como finalidade atualizar o contexto do mecanismo do RAG, utilizando como parâmetro dados estruturados no formato JSON.

**Figura 7** - Código do servidor Ollama definindo o *PromptTemplate*.

```
# Define o template (personalidade) que será utilizada pelo modelo.
prompt_template_sql = PromptTemplate.from_template(
    """
    Dado o seguinte esquema de banco de dados SQL:
    {schema_info}

    Converta a seguinte consulta em linguagem natural em uma instrução
    SQL SELECT utilizando apenas o esquema como referência:
    {input_text}

    Regras:
    1. Se perguntando algo não relacionado a consultas,
       responda: **"Eu não sei."**
    2. Ignore tabelas ou colunas não reconhecidas,
       não invente novas.
    3. Responda apenas com consultas SQL entre três crases (```)
       sem explicações ou comentários.
    4. Use os nomes completos das tabelas conforme definidos pelo
       Django (por exemplo, `usuario` torna-se `usuario_usuario`).
    5. Nomes de tabelas e colunas podem conter acentos,
       remova os acentos ao comparar com o texto da consulta.
    6. Para consultas sobre pesquisadores, investigadores
       ou professores, use a tabela `usuario_usuario`.
    7. Considere o texto sem distinção entre maiúsculas
       e minúsculas (`professor` = `PROFESSOR`).
    8. Para perguntas sobre conteúdo ou área de pesquisa, consulte
       a tabela `submissao_submissao` utilizando os campos relevantes.
    """
)
```

Produzido pelos autores.

A geração das respostas pelo modelo de linguagem é feita por meio do *PromptTemplate* da biblioteca LangChain. A partir desta ferramenta, foi possível criar uma forma de molde dinâmico contendo duas variáveis essenciais, a pergunta do usuário e o esquema do banco de dados. Este *template*, ilustrado na Figura 7, é utilizado sempre que o modelo precisa gerar uma resposta, garantindo que o *prompt* seja processado de maneira estruturada e contextualizada.

Para que esses métodos possam ser acessados remotamente, foi implementada uma nova classe chamada *ManipuladorDeRequisicoes*. Representada na Figura 8, a variável *server* é um objeto dessa classe, que registra estes dois métodos mencionados anteriormente. Após o registro, o servidor Ollama entra em execução e a URL pública de conexão é fornecida ao usuário, possibilitando a integração com outros serviços, como o SISGEP-COMIC.

A implementação local do servidor Ollama mostrou-se significativamente mais complexa do que a alternativa de terceirização por serviços como o ChatGPT da OpenAI. Essa diferença vem de diversos fatores, como a maior complexidade técnica e os custos associados ao hardware. Além disso, outras limitações também impactaram negativamente os resultados obtidos em relação a serviço de terceiros, conforme será detalhado na subseção da tabela de resultados.

Por outro lado, este método também apresenta vantagens importantes, como o maior controle sobre a manipulação do modelo e a maior independência, uma vez que o sistema pode ser utilizado de forma totalmente *offline* e dentro da própria infraestrutura em que foi implementado.

**Figura 8** - Código da classe *ManipuladorDeRequisicoes* e inicialização.

```
def run_ollama_serve():
  subprocess.Popen(["ollama", "serve"])
thread = threading.Thread(target=run_ollama_serve)
thread.start()
time.sleep(5)
# Incializa o modelo usando recém criado llama3.1.
llm = OllamaLLM(
    model="llama3.1",
)
# Inicializa a chain para fazer chamadas para LLM usando template.
sql_chain = prompt_template_sql | llm
# Cria o servidor RPC para responder requisições.
class ManipuladorDeRequisicoes(SimpleXMLRPCRequestHandler):
    rpc_paths = ("/RPC2",)

server = SimpleXMLRPCServer(
    ("0.0.0.0", 1346),
    requestHandler=ManipuladorDeRequisicoes,
    allow_none=True
)
# Registra as funções para gerar SQL e atualizar contexto via RPC.
server.register_function(atualizar_contexto, "atualizar_contexto")
server.register_function(gerar_resposta, "gerar_resposta")
# Cria URL pública pelo ngrok para acessar fora do Google Colab.
try:
    url_publica = ngrok.connect(1346, bind_tls=True).public_url
    print("URL Pública: ", url_publica)
except Exception as e:
    print(f"Falha ao conectar com ngrok: {str(e)}")
    raise
# Instancia o servidor e executa sem interrupção
print("Servidor XML-RPC em execução...")
server.serve_forever()
```

Produzido pelos autores.

## 4.2 ADAPTAÇÕES NO SISGEP-COMIC

Após a implementação do servidor Ollama, capaz de responder com consultas SQL, a próxima etapa foi implementar a interação com o usuário em uma aplicação Web usando *framework* Django, neste caso, o SISGEP-COMIC.

A Figura 9 apresenta três métodos da classe *Conecta*, utilizados para a conexão com o servidor Ollama por meio de uma URL pública. A própria classe também é responsável por estabelecer essa conexão, invocando o método correspondente sempre que for necessário gerar uma instrução SQL ou atualizar o contexto.

**Figura 9** - Códigos utilizando RPC da classe *Conecta*.

```python
    @staticmethod
    def conecta_rpc():
        url_servidor = config('GERADORSQL_URL')
        try:
            # Verifica se houve alteração pelo migrations
            if Conecta.houve_alteracao_banco():
                Conecta.atualiza_contexto()

            proxy = xmlrpc.client.ServerProxy(url_servidor)
            return proxy
        except Exception as e:
            print('Erro ao conectar no servidor: ', e)
            return 'Não foi possível conectar no servidor. Tente novamente mais tarde.'
    @staticmethod
    def atualiza_contexto():
        try:
            proxy = Utils.gera_esquema_banco()
            with open("esquema_banco.json", "r") as file:
                #Lê o arquivo Json e envia para o servidor
                json_data = file.read()
                resposta = proxy.atualiza_contexto(json_data)
                return resposta
        except Exception as e:
            return f"Erro ao atualizar contexto. Exceção: {str(e)}"
    @staticmethod
    def gera_sql(pergunta):
        try:
            proxy = Conecta.conecta_rpc()
            resposta = proxy.gera_resposta(pergunta)
            return resposta
        except Exception as e:
            print('Erro ao gerar SQL: ', e)
            return 'Não foi possível conectar no servidor. Tente novamente mais tarde.'
```

Produzido pelos autores.

Para recuperar as informações, a Figura 10 apresenta o método responsável por realizar a conexão com o banco de dados, utilizando a ferramenta *cursor* para executar as instruções SQL. A consulta ocorre apenas se a verificação de segurança for bem-sucedida, ou seja, se o método de validação retornar verdadeiro. Ao final do processo, o objeto *cursor* e a conexão com o banco de dados são fechados.

**Figura 10** - Código do método para executar SQL da classe *Conecta*.

```python
    @staticmethod
    def executa_sql(sql):
        try:
            # Verifica se a consulta é segura
            if not Conecta.checa_consulta_segura(sql):
                return "Uma consulta potencialmente insegura foi detectada. Por favor, tente novamente."
            # Conecta no banco com cursor e obtém os resultados
            with connection.cursor() as cursor:
                cursor.execute(sql)
                resultados = cursor.fetchall()
                # Se não houver resultados, retorna a mensagem
                if not resultados:
                    resultados = "Nenhum resultado relevante foi encontrado."
                # Obtém os nomes dos campos
                nomes_campos = [i[0].upper() for i in cursor.description]
                cursor.close()
            # Se não houver "order by" na consulta, ordena a lista de listas
            if "order by" not in sql.upper():
                lista_dados = sorted(resultados, key=lambda x: x[0])
            else:
                lista_dados = resultados
            # Filtra os resultados
            lista_filtrada = Conecta.filtra_resultados(lista_dados, nomes_campos)
            # Gera a tabela HTML com os resultados filtrados
            return Conecta.gera_tabela_html(nomes_campos, lista_filtrada)
        except Exception as e:
            print('Erro ao executar a consulta: ', e)
            return f"Erro na execução da consulta. Tente gerar a consulta novamente."
```

Produzido pelos autores.

Na etapa de exibição dos resultados, o código da Figura 11 chama, no final, outros dois métodos que filtram os nomes dos campos e os dados de cada linha em listas. Em seguida, essas informações são estruturadas em uma tabela HTML usando a ferramenta *Template* do Django.

Na Figura 12, é apresentado o código do método responsável por gerar a tabela HTML junto de sua localização na camada *view* do relatório. Como o objeto resposta é de um tipo reconhecido pelo Django, a renderização em HTML dos dados ocorre automaticamente, porém apenas quando é utilizada a palavra-chave *safe*, junto da pergunta e da consulta executada.

Como resultado, a visão do usuário, representada na Figura 13, mostra o que é retornado para a seguinte pergunta: "*Informe o nome, o e-mail e o currículo Lattes de todos os usuários cadastrados que são professores*".

No servidor Ollama, a Figura 14 representa a visão do terminal durante a execução dessa pergunta. Nela é possível observar a requisição do usuário, que ao terminar de processar, retorna como resposta a consulta SQL.

**Figura 11** - Código do método para gerar uma tabela HTML.

conecta_llm.py

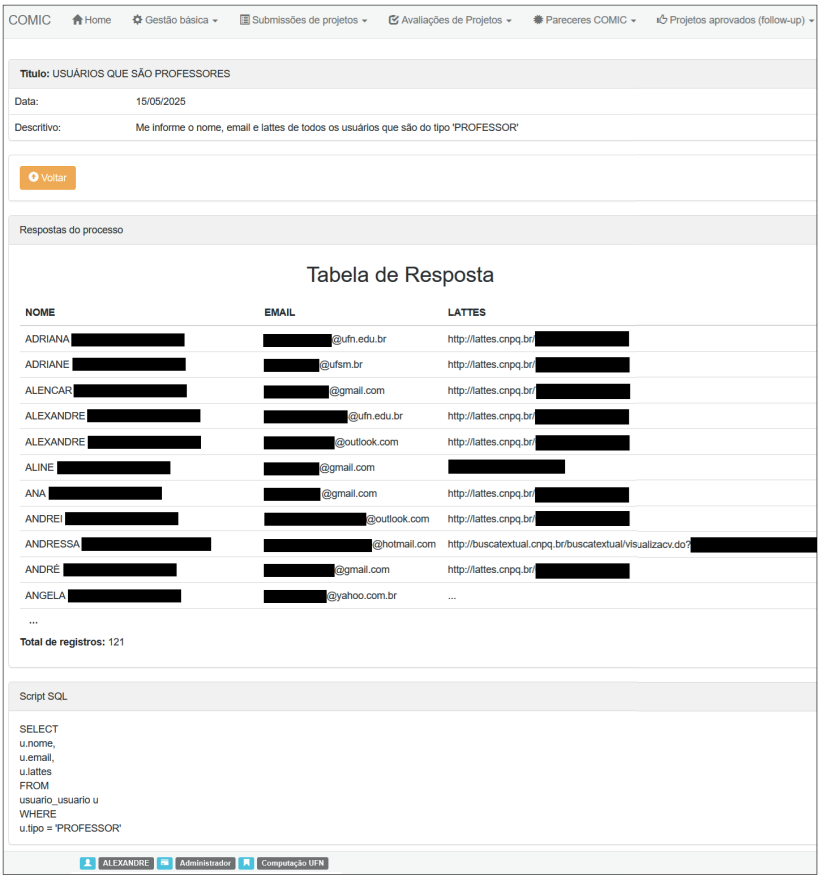
```
@staticmethod
def gera_tabela_html(nomes_campos, lista_dados):
    template_str = """
                    <h2 style="text-align:center;">Tabela de Resposta</h2>
                    <table class="table table-hover">
                        <thead>
                            <tr>
                                {% for campo in nomes_campos %}
                                    <th>{{ campo }}</th>
                                {% endfor %}
                            </tr>
                        </thead>
                        <tbody>
                            {% for item in lista_dados %}
                                <tr>
                                    {% for valor in item %}
                                        <td>{{ valor }}</td>
                                    {% endfor %}
                                <tr>
                            {% endfor %}
                        </tbody>
                    </table>
                    <p><b>Total de registros:</b> {{ lista_dados|length }}</p>
                    """
    template = Template(template_str)
    context = Context({
        'nomes_campos': nomes_campos,
        'lista_dados': lista_dados
    })
    return template.render(context)
```

relatorio_detail.html

```
<div class="panel panel-default">
    <div class="panel-heading">
        Respostas do processo
    </div>
    <div class="panel-body">
        {% if object.resposta %}
            {{ object.resposta|safe }}
        {% else %}
            <span class="label label-warning" >
                Resposta não disponível
            </span>
        {% endif %}
        <br>
```

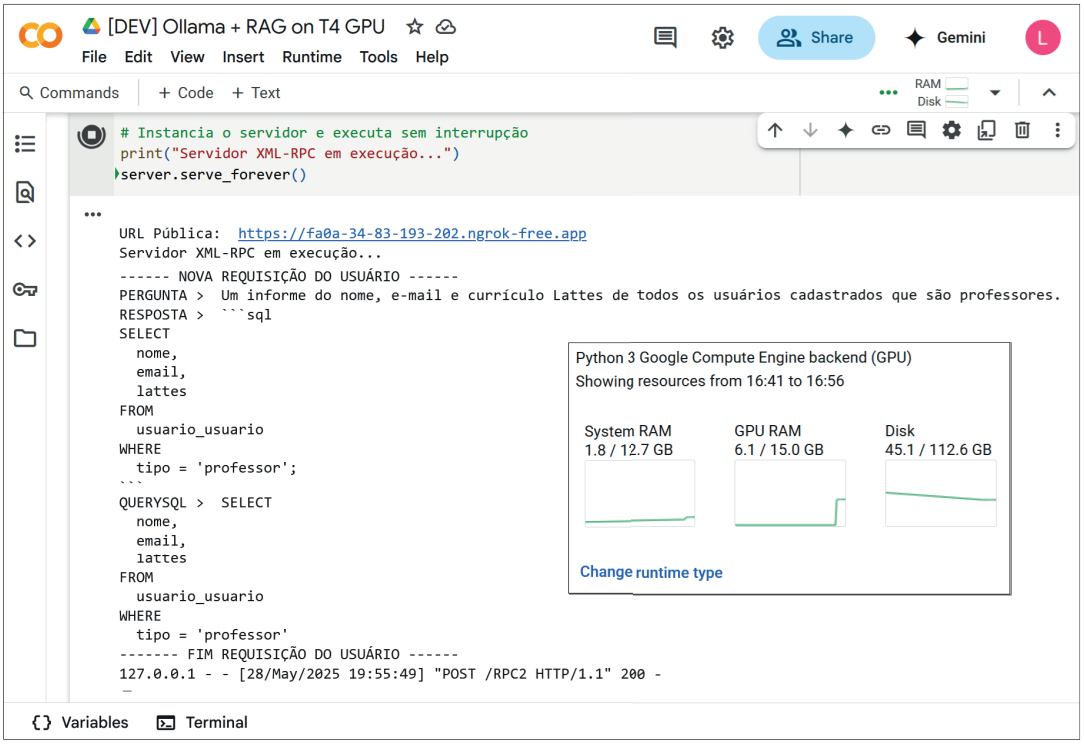
Produzido pelos autores.

**Figura 12** - Resultado da pergunta de exemplo.




Produzido pelos autores.

**Figura 13** - Tela do servidor Ollama executando no ambiente Google Colab.




Produzido pelos autores.

A partir dos métodos apresentados, foi possível utilizar componentes de ambos os sistemas para aplicar a sequência de testes proposta no estudo de caso, organizando-os em uma tabela de resultados.

## 4.3 ELABORAÇÃO DA TABELA DE RESULTADOS

As métricas de avaliação propostas no estudo de caso fornecem uma base quantitativa essencial para medir a usabilidade do produto desenvolvido. Permitindo por meio de sua análise determinar se o módulo, em seu estado atual, está apto para uso em produção. Para obter essas métricas, foi implementado um código isolado do *framework* Django, além de uma nova classe específica para agrupar os dados, seguindo as boas práticas da programação orientada a objetos. A implementação desse sistema faz uso da classe *Resultado*, representada no diagrama de classes da Figura 2.

Conforme detalhado no diagrama, as métricas foram geradas por meio de diversos métodos. O principal deles, representado na Figura 14, é responsável por gerar e retornar uma lista de objetos *Resultado*. Esse método mede o tempo para gerar o SQL de cada pergunta, além de chamar outro método para calcular a precisão, contabilizar as falhas por meio do tratamento de exceções e avaliar a consistência dos dados ao realizar múltiplas iterações da mesma pergunta. Como parâmetros, ele recebe uma lista de perguntas, uma lista de comandos SQL de referência e a quantidade de consultas realizada para cada pergunta.

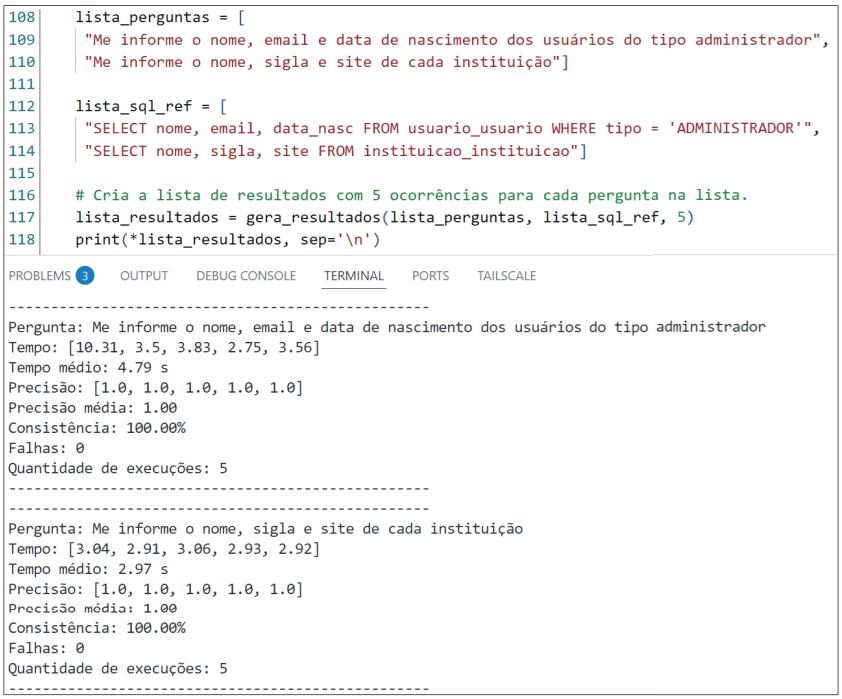
**Figura 14 -** Método utilizado para gerar resultado e suas métricas.

```
44 def gera_resultados(lista_perguntas, lista_sql_referencia, qtd_execucoes):
45     lista_resultados = []
46     if(len(lista_perguntas) != len(lista_sql_referencia)):
47         raise ValueError("A lista de perguntas e SQL de ref. não tem o mesmo tamanho!")
48     for pergunta, sql_referencia in zip(lista_perguntas, lista_sql_referencia):
49         lista_tempos = []
50         lista_precisao = []
51         consistencia = 0.0
52         qtd_falhas = 0
53         for i in range(qtd_execucoes):
54             try:
55                 #1. Gera a consulta SQL via RPC (Temporizado)
56                 tempo_inicial = time.time()
57                 sql_gerado = Conecta.gera_sql(pergunta)
58                 print(f"[{i+1}] SQL : {sql_gerado}")
59                 tempo_passado = time.time() - tempo_inicial
60                 lista_tempos.append(round(tempo_passado, 2))
61                 #2. Executa a consulta SQL gerada e de referência
62                 (linha_ref, campos_ref) = executa_sql(sql_referencia)
63                 (linha_ger, campos_res) = executa_sql(sql_gerado)
64                 #3. Filtra os resultados
65                 lista_referencia = Conecta.filtra_resultados(linha_ref, campos_ref)
66                 lista_gerada = Conecta.filtra_resultados(linha_ger, campos_res)
67                 #4. Calcula a precisão
68                 precisao = calcular_precisao(lista_referencia, lista_gerada)
69                 lista_precisao.append(precisao)
70             except Exception as e:
71                 # Se ocorrer um erro, contabiliza a falha e continua
72                 print(f"Erro na execução {i}: {e}")
73                 qtd_falhas += 1
74                 lista_precisao.append(0.0)
75         #5. Calcula a consistência
76         consistencia = sum(lista_precisao) / qtd_execucoes
77         #6. Cria o objeto Resultado e adiciona à lista
78         resultado = Resultado(pergunta, lista_tempos, lista_precisao,
79                               consistencia, qtd_falhas, qtd_execucoes)
80         lista_resultados.append(resultado)
81     return lista_resultados
```

Produzido pelos autores.

Para fins de teste e validação dos métodos apresentados, uma lista de duas perguntas simples foi elaborada. A Figura 15 apresenta essas perguntas e seu respectivo SQL de referência, as métricas do resultado dessas perguntas pode ser visualizada na saída do terminal.

**Figura 15** - Exemplo para duas perguntas e geração do resultado.




Produzido pelos autores.

Após confirmar que todos os métodos estavam funcionando corretamente, foi possível realizar o teste final com as dez perguntas propostas no estudo de caso. Sendo elas:

- Informe o nome, curso e área de todos os usuários;
- Informe o nome, e-mail e Lattes de todos os usuários que são do tipo professor;
- Informe o nome, e-mail e último acesso de todos os usuários do tipo administrador;
- Informe o nome, sigla e site de cada instituição;
- Informe o local de execução, área e curso de cada submissão cadastrada;
- Informe quantos usuários cadastrados nasceram antes do ano 2000.;
- Informe o nome, área de conhecimento e curso pós de todos os usuários que têm curso de pós graduação;
- Informe o nome do avaliador suplente e o seu parecer de suplente para todas as avaliações;
- Informe a descrição do edital, o nome, e-mail e Lattes do responsável e título de todas as submissões realizadas no edital de número 2021;
- Informe o nome do avaliador, a data de avaliação do responsável e o parecer do avaliador suplente para todas as avaliações que foram realizadas pelo avaliador responsável Cristina De Freitas Rodrigues;

Para calcular a precisão e consistência dessas perguntas, a Figura 16 representa as instruções SQL de referência de cada pergunta, ordenadas por nível de complexidade para simular um cenário de melhor caso e pior caso.

**Figura 16** - Lista de SQL de referência para testar cada pergunta.

```
lista_sql_ref_final = [
  SELECT nome, curso_graduacao_vinculado, area_conhecimento_cnpq
  FROM usuario_usuario,

  SELECT nome, email, lattes
  FROM usuario_usuario
  WHERE tipo = 'PROFESSOR',

  SELECT nome, email, last_login
  FROM usuario_usuario
  WHERE tipo = 'ADMINISTRADOR',

  SELECT nome, sigla, site
  FROM instituicao_instituicao,

  SELECT local_execucao, area, curso_graduacao_vinculado
  FROM submissao_submissao,

  SELECT COUNT(*) AS total_usuarios
  FROM usuario_usuario
  WHERE data_nasc < '2000-01-01',

  SELECT nome, area_conhecimento_cnpq, curso_pos_graduacao
  FROM usuario_usuario
  WHERE curso_pos_graduacao IS NOT NULL,

  SELECT u.nome AS avaliador_suplente, a.parecer_avaliador_suplente
  FROM avaliacao_avaliacao a
  LEFT JOIN usuario_usuario u ON a.avaliador_suplente = u.id,

  SELECT e.descricao, u.nome, u.email, u.lattes, s.titulo
  FROM submissao_submissao s
  JOIN edital_edital e ON s.edital = e.id
  JOIN usuario_usuario u ON s.responsavel = u.id
  WHERE e.numero = '2021',

  SELECT u.nome AS avaliador_responsavel, a.dt_avaliacao_responsavel,
  FROM avaliacao_avaliacao a                a.parecer_avaliador_suplente
  JOIN usuario_usuario u ON a.avaliador_responsavel = u.id
  WHERE u.nome = 'CRISTINA DE FREITAS RODRIGUES'
]
```

Produzido pelos autores.

Com a execução das dez perguntas, cada uma com dez iterações adicionais para medir a consistência, foi possível extrair as informações utilizando o tempo médio e a precisão média como generalização dos resultados obtidos. A Tabela 1 apresenta esses dados, sendo possível chegar a uma conclusão sobre a qualidade do sistema e se está pronto para ser utilizado na produção.

**Tabela 1** - Desempenho por Pergunta.

| Pergunta | Tempo | Precisão | Consistência | Falhas | Consultas |
|---|---|---|---|---|---|
| 01 | 3,20s | 100% | 100% | 0% | 10 |
| 02 | 3,34s | 100% | 100% | 0% | 10 |
| 03 | 3,34s | 100% | 100% | 0% | 10 |
| 04 | 2,86s | 100% | 100% | 0% | 10 |
| 05 | 4,09s | 0% | 0% | 100% | 10 |
| 06 | 2,86s | 100% | 100% | 0% | 10 |
| 07 | 5,58s | 100% | 100% | 0% | 10 |
| 08 | 3,58s | 0% | 0% | 100% | 10 |
| 09 | 3,26s | 0% | 0% | 100% | 10 |
| 10 | 3,79s | 0% | 0% | 100% | 10 |

Produzido pelos autores.

A partir da análise dos resultados obtidos e da realização de testes adicionais no sistema SISGEP-COMIC, concluiu-se que o sistema é capaz de gerar consultas SQL, embora dependa de diversos fatores que dificultam a generalização do módulo. Ao testar as mesmas perguntas, mas sem a ajuda de ambiguidades específicas do SISGEP-COMIC nas regras do *PromptTemplate* (linha 99 da Figura 7) , observou-se uma redução significativa na precisão e consistência das respostas. Concluindo a necessidade de atenção especial à estrutura do contexto do banco de dados utilizado (linha 25 da Figura 6), bem como um ajuste fino nas regras e diretrizes definidas no *PromptTemplate*.

A partir dos resultados apresentados, foi possível avaliar o desempenho do módulo como adequado. A resposta do sistema é precisa para perguntas mais simples e bem definidas, mas tende a perder essa precisão diante de ambiguidades, seja na formulação da pergunta ou na própria estrutura do banco de dados fornecida. Outro fator relevante é o desempenho do modelo adotado. Ao utilizar a LLM GPT-4º, disponibilizada pelo serviço da OpenAI, obteve-se um aumento significativo na precisão, além de otimizações e boas práticas específicas para cada cenário, resultando em tempos de resposta mais rápidos para cada pergunta testada.

## 5 CONCLUSÕES

Neste trabalho, foram explorados os fundamentos teóricos da Inteligência Artificial Generativa, com ênfase no processamento de linguagem natural e nos modelos de linguagem. Além disso, foram abordadas técnicas de contextualização com RAG e ferramentas como LangChain e Ollama, que viabilizam a integração de LLMs locais. Os trabalhos correlatos, como os de Müller e Zamberlan (2020), de Maurer e Zamberlan (2024) e de Barros e Martins (2024), forneceram uma base sólida ao destacarem, respectivamente, a relevância do sistema SISGEP-COMIC, a implementação de LLMs locais e a avaliação de respostas geradas por modelos de linguagem. Esses estudos reforçam tanto a viabilidade quanto os desafios envolvidos no desenvolvimento de um módulo de interação textual para a geração de relatórios.

A implementação do módulo proposto envolveu a integração de um servidor Ollama, executando o modelo Llama 3.1 8B, configurado para operar localmente ou na nuvem (via *Google Colab*), utilizando chamadas remotas de procedimento (RPC) para gerar consultas SQL no sistema web SISGEP-COMIC. Nesse sistema, a classe *Conecta* foi desenvolvida para gerenciar a interação com o servidor Ollama, validar consultas SQL e exibir os resultados em tabelas HTML. No servidor Ollama, o *framework* LangChain, com suporte ao *PromptTemplate*, permitiu a conversão de *prompts* em linguagem natural em consultas SQL, utilizando o RAG para aumentar a precisão com base no esquema do banco de dados, descrito em JSON.

Os resultados apresentados na Tabela 1 indicam que o módulo foi capaz de gerar consultas SQL com precisão adequada e consistência para perguntas simples, com tempos de execução entre

três e seis segundos, sem ocorrência de falhas. No entanto, perguntas mais complexas apresentaram altas taxas de falhas e ausência de precisão, o que evidencia limitações na interpretação de consultas mais ambíguas.

Com o objetivo de aprimorar os resultados, recomenda-se para trabalhos futuros: i) testar diferentes LLMs locais mais recentes, como *DeepSeek R1* ou versões mais robustas do Llama, visando melhorar a precisão em perguntas mais complexas; ii) aprofundar o treinamento das regras do *PromptTemplate*, ajustando o contexto do RAG para melhor lidar com ambiguidades; iii) avaliar o módulo utilizando assinaturas de serviços de LLMs terceirizados, como *OpenAI GPT-4* ou *xAI Grok-3*, para superar as limitações de hardware do servidor local.

## REFERÊNCIAS

AMAZON LANGCHAIN. **What is LangChain?** 2024. Disponível em: https://aws.amazon.com/what-is/langchain/. Acesso em: 31 out. 2024.

AMAZON RAG. **What is Retrieval-Augmented Generation?** 2024. Disponível em: https://aws.amazon.com/what-is/retrievalaugmented-generation/. Acesso em: 30 out. 2024.

BARROS, G. R. C.; MARTINS, M. O. **Comparative of source code generated by principal LLM generators for Python and Lua languages**. 2024. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Franciscana (UFN), Santa Maria, RS, Brasil, 2024. Disponível em: https://tfgonline.lapinf.ufn.edu.br.

CHASE, H. **LangChain**. 2024. Disponível em: (https://www.langchain.com/). Acesso em: 28 out. 2024.

CLOUDFLARE. **What is a large language model (LLM)?** 2024. Disponível em: https://www.cloudflare.com/en-gb/learning/ai/what-is-large-language-model/. Acesso em: 29 out. 2024.

D'SOUZA, J. **A Review of Transformer Models.** 2023. DOI: 10.48366/r640001. Disponível em: https://www.researchgate.net/publication/373757234_A_Review_of_Transformer_Models. Acesso em: 24 out. 2024.

JALOTE, P. **Fault Tolerance in Distributed Systems.** PTR Prentice Hall, 1994. ISBN: 9780133013672. Disponível em: https://books.google.com.br/books?id=PZzSngEACAAJ.

JÚNIOR, A. K. **Computação em nuvem**. 1. ed. São Paulo: Contentus, 2020. E-book.

McAULEY, D. **AI and LLMs in Cloud Computing: Challenges and Opportunities.** Advances in Computer Sciences, v. 7, n. 1, 2024.

MAURER, P. G. G.; ZAMBERLAN, A. O. **Protótipo de Interação Humano-Computador para Processamento da Língua Natural em LLMs.** 2024. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Franciscana (UFN), Santa Maria, RS, Brasil, 2024. Disponível em: https://tfgonline.lapinf.ufn.edu.br.

META. **Introducing Llama 3.1: Our most capable models to date.** 2024. Disponível em: https://ai.meta.com/blog/meta-llama-3-1/. Acesso em: 9 out. 2024.

MERRITT, R. **What Is Retrieval-Augmented Generation, aka RAG?** 2024. Disponível em: https://blogs.nvidia.com/blog/what-is-retrieval-augmented-generation/. Acesso em: 30 out. 2024.

MICROSOFT LEARN A. **What is generative AI?** 2024. Disponível em: https://learn.microsoft.com/en-us/training/modules/fundamentals-generative-ai/2-what-is-generative-ai. Acesso em: 20 out. 2024.

MICROSOFT LEARN B. **What are language models?** 2024. Disponível em: https://learn.microsoft.com/en-us/training/modules/fundamentals-generative-ai/3-language%20models. Acesso em: 29 out. 2024.

MISHRA, B. K.; KUMAR, R. **Natural language processing in artificial intelligence**. CRC Press, 2020.

MÜLLER, P. T.; ZAMBERLAN, A. O. **Módulo Sisgep Comic: gestão de avaliação de projetos**. 2020. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) - Universidade Franciscana (UFN), Santa Maria, RS, Brasil, 2020. Disponível em: https://tfgonline.lapinf.ufn.edu.br.

NELSON, B. J. **Remote Procedure Call.** Also CMUCS-81-119. Tese (Doutorado) - Xerox Palo Alto Research Center, 1981.

NGROK. **All your traffic. One gateway.** 2026. Disponível em: https://ngrok.com/. Acesso em: 18 mai. 2026.

OLLAMA. **Ollama - Get up and running with large language models.** 2024. Disponível em: https://ollama.com/. Acesso em: 10 out. 2024.

OLLAMA PYTHON. **Ollama Python Library.** 2024. Disponível em: https://github.com/ollama/ollama-python. Acesso em: 28 out. 2024.

OPENAI A. **ChatGPT - Overview.** 2024. Disponível em: https://openai.com/chatgpt/overview/. Acesso em: 5 out. 2024.

OPENAI B. **OpenAI - Privacy policy.** 2024. Disponível em: https://openai.com/policies/row-privacy-policy/. Acesso em: 22 out. 2024.

ORACLE. **What is machine learning?** 2024. Disponível em: https://www.oracle.com/artificial-intelligence/machine-learning/what-is-machine-learning/. Acesso em: 30 set. 2024.

PYTHON. **Welcome to Python.org.** 2024. Disponível em: https://www.python.org/. Acesso em: 2 out. 2024.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. Pearson, 2016.

STRYKER, C.; HOLDSWORTH, J. **What is NLP (natural language processing)?** 2024. Disponível em: https://www.ibm.com/topics/natural-language-processing. Acesso em: 21 set. 2024.

VASWANI, A. et al. **Attention Is All You Need.** 2017. Disponível em: https://arxiv.org/abs/1706.03762. Acesso em: 16 abr. 2024.

XU, C. Z. **Scalable and Secure Internet Services and Architecture**. Chapman & Hall/CRC Computer and Information Science Series. CRC Press, 2005. ISBN: 9781420035209. Disponível em: https://books.google.com.br/books?id=QnXLBQAAQBAJ.