Retrieval Augmented Generation

Carlos Eduardo Aleixo[1], Henrique Crispin de Aguiar[1], João Victor Carvalho[1]

[1]Departamento de Sistemas e Computação
Universidade Luterana do Brasil(Ulbra) – Palmas, TO – Brasil

Carlos Eduardo Costa Aleixo Dos Santos, Henrique Crispin De Aguiar,
Joao Victor Da Silva Carvalho

Abstract. *In the field of Natural Language Processing (NLP), one of the ongoing challenges is improving the accuracy and relevance of responses generated by language models. Retrieval-Augmented Generation (RAG) models have emerged as a promising solution, combining language generation with the retrieval of information from external databases. This study aims to implement a RAG model in a local Small Language Model (SLM) using tools such as Ollama, Phi3, ChromaDB, and Nomic Embedding. The results indicate that chunk size did not cause significant changes in the outcomes, and the issue appears to lie in the number of documents in the database and the model's ability to find fragments truly similar to the user's prompt.*

Resumo. *No campo do Processamento de Linguagem Natural (PLN), um dos desafios contínuos é melhorar a precisão e relevância das respostas geradas por modelos de linguagem. Modelos de Recuperação-Aumentada por Geração (RAG) surgiram como uma solução promissora, combinando a geração de linguagem com a recuperação de informações de bases de dados externas. Este estudo visa implementar um modelo RAG em um Small Language Model (SLM) local, utilizando ferramentas como Ollama, Phi3, ChromaDB e Nomic Embedding. Os resultados indicam que o tamanho dos fragmentos não causou mudanças relevantes nos resultados e que o problema parece residir na quantidade de documentos no banco de dados e na capacidade do modelo de encontrar os fragmentos realmente similares ao prompt do usuário.*

1. Introdução

No campo do Processamento de Linguagem Natural (PLN), um dos desafios contínuos é melhorar a precisão e relevância das respostas geradas por modelos de linguagem. Modelos de grande porte, como GPT-3, têm demonstrado capacidades impressionantes, mas enfrentam problemas significativos, incluindo a geração de informações imprecisas ou alucinações, a necessidade de enormes quantidades de dados e recursos computacionais para treinamento e inferência, e dificuldades em manter as informações atualizadas (LEWIS et al., 2020).

Por outro lado, modelos de linguagem menores (Small Language Models - SLMs) oferecem várias vantagens. Eles são mais leves e exigem menos recursos computacionais, o que os torna mais acessíveis e viáveis para uma ampla gama de usuários. Além disso, rodar um modelo localmente traz benefícios significativos em termos de privacidade e segurança, permitindo que os dados sensíveis permaneçam no dispositivo do usuário sem a necessidade de transmissão para servidores externos.

A implementação de modelos de Recuperação-Aumentada por Geração (RAG) em uma SLM local combina os benefícios dos SLMs com as vantagens da recuperação de informações de bases de dados externas (LEWIS et al., 2020). O RAG melhora a precisão e relevância das respostas geradas ao buscar dados relevantes em tempo real, incorporando essas informações no processo de geração de texto. Isso é particularmente poderoso quando rodado localmente, pois permite que qualquer pessoa tenha uma ferramenta de análise e geração de texto operando em seu próprio dispositivo, utilizando seus dados pessoais para gerar respostas privadas e personalizadas.

Este estudo visa implementar um modelo RAG em uma SLM local, utilizando ferramentas como Ollama, Phi3, ChromaDB e Nomic Embedding. A configuração do ambiente de teste foi projetada para analisar a performance do RAG sob diferentes configurações de chunking. Os resultados indicam que o tamanho dos fragmentos não causou mudanças relevantes nos resultados e que o problema parece residir na quantidade de documentos no banco de dados e na capacidade do modelo de encontrar os fragmentos realmente similares ao prompt do usuário.

2. Referencial Teórico

Para implementar nosso sistema RAG em uma SLM local, selecionamos ferramentas que combinam facilidade de uso, alta qualidade e que são open-source. Essas ferramentas não apenas oferecem robustez e eficiência, mas também permitem que a comunidade de desenvolvedores contribua e melhore continuamente as suas funcionalidades.

2.1. RAG (Retrieval-Augmented Generation)

(Retrieval-Augmented Generation (RAG) é uma abordagem que combina técnicas de recuperação de informações (retrieval) com a geração de texto (generation) para melhorar a qualidade e a relevância das respostas geradas por modelos de linguagem sem a necessidade de treinar novamente o modelo (LEWIS, 2020). Este método aborda uma das principais limitações dos modelos de linguagem tradicionais: a dependência do conhecimento incorporado durante o treinamento, que pode ficar desatualizado ou ser insuficiente para responder a consultas específicas.

Em primeiro lugar, o modelo utiliza um mecanismo de recuperação de informações para buscar dados em uma base de conhecimento externa. Esta base pode ser composta por documentos, artigos, bancos de dados ou qualquer fonte de informação estruturada ou não estruturada. A ideia é encontrar fragmentos de texto que possam ajudar a responder a pergunta ou enriquecer o contexto da consulta.

Em seguida, o modelo de geração de linguagem utiliza os fragmentos recuperados para produzir uma resposta mais coesa e informativa. A geração de texto é orientada pelo conteúdo recuperado, permitindo que o modelo produza respostas mais precisas e atualizadas.

2.2. Ollama

Ollama é uma ferramenta de código aberto para executar e gerenciar modelos de linguagem de grande porte (LLMs) na sua máquina local. Esta ferramenta é projetada para facilitar a execução eficiente de LLMs e é especialmente útil quando combinada com técnicas de Recuperação-Augmented Generation (RAG). O Ollama serve como a base para sistemas de geração de linguagem, proporcionando a capacidade de produzir respostas coesas e contextualmente apropriadas. Esta característica é essencial para aplicações que exigem precisão e clareza na comunicação.

2.3. Phi3

Phi 3 é um modelo de linguagem avançado, open-source, que oferece capacidades robustas de processamento de linguagem natural. Projetado para suportar tarefas complexas de geração de texto, o Phi 3 utiliza técnicas de aprendizado profundo para melhorar a compreensão e a geração de texto. O modelo utilizado neste estudo possui 3.8 bilhões de parâmetros e ocupa 2.4 GB de memória, o que o torna relativamente leve e eficiente. Sua arquitetura otimizada permite uma integração eficaz em sistemas RAG, onde a precisão e a relevância das respostas são importantes. A flexibilidade do Phi 3 para lidar com diversos tipos de entradas textuais e sua capacidade de adaptação a diferentes contextos são aspectos fundamentais para o desempenho do nosso sistema RAG.

2.4. Nomic

Nomic é uma ferramenta de embedding que oferece representações vetoriais para dados textuais. Embeddings são representações numéricas que capturam o significado semântico das palavras ou frases em um espaço vetorial, permitindo a comparação de similaridades e relações entre diferentes fragmentos de texto. O próprio Nomic é utilizado para gerar embeddings de consultas e documentos. Essas representações vetoriais são fundamentais para a etapa de recuperação de informações do RAG, permitindo que o sistema encontre e compare fragmentos de texto com alta precisão. Com embeddings de alta qualidade, o sistema RAG pode identificar documentos ou partes deles que são mais relevantes para a consulta do usuário, melhorando a precisão das informações recuperadas.

2.5. ChromaDB

O ChromaDB é uma solução essencial para o armazenamento e gerenciamento de embeddings em ambientes de código aberto, oferecendo uma plataforma flexível e eficiente para a manipulação de dados estruturados e não estruturados. Utilizado em conjunto com a abordagem Retrieval-Augmented Generation (RAG), o ChromaDB permite a busca e organização de documentos baseando-se na similaridade semântica, o que é fundamental para a recuperação de informações precisas e relevantes. A capacidade de moldar e buscar documentos com base em embeddings de alta qualidade contribui significativamente para a eficácia do RAG, tornando o ChromaDB uma ferramenta importante para nossa pesquisa.

3. Materiais e Métodos

A implementação do modelo RAG utilizando Ollama Phi 3, Nomic e ChromaDB foi realizada em um ambiente controlado, utilizando várias ferramentas e bibliotecas. O ChromaDB foi configurado para rodar dentro de um container Docker, enquanto os códigos Python, incluindo a utilização das bibliotecas LangChain, foram executados dentro do WSL2 (Windows Subsystem for Linux). Esta configuração permitiu um ambiente de desenvolvimento flexível, garantindo a privacidade e o processamento local dos dados.

3.1 Fluxo de Execução

A implementação do modelo RAG utilizando Ollama Phi 3, Nomic e ChromaDB seguiu o seguinte fluxo:

- Divisão de Documentos: Os documentos são primeiro quebrados em fragmentos, chamados de chunks, utilizando a ferramenta de divisão de texto.
- Criação de Vetores: Após a divisão, os vetores correspondentes a esses chunks são gerados utilizando o Nomic.
- Armazenamento em ChromaDB: Os vetores criados são então inseridos no ChromaDB. Esse processo só precisa acontecer quando novos documentos precisarem ser adicionados à memória não paramétrica do modelo.
- Configuração do Modelo Phi 3: O modelo Phi 3 é rodado localmente utilizando a plataforma Ollama.
- Recebimento de Consultas: Uma função chamada query recebe o prompt do usuário.
- Busca de Documentos Relevantes: Uma função procura no ChromaDB por documentos que tenham similaridade com o prompt do usuário.
- Seleção dos Documentos Mais Relevantes: A busca retorna os documentos mais similares ao prompt.
- Construção do Prompt Final: Os documentos encontrados são adicionados ao prompt final que será passado para o Phi 3. O template do prompt utilizado é o seguinte:

```
PROMPT_TEMPLATE = """
Answer the question based only on the following context:

{context}

---

Answer the question based on the above context: {question}
"""
```

Neste template, {context} representa os documentos encontrados e {question} é o prompt do usuário. O modelo gera uma resposta baseada exclusivamente no contexto fornecido pelos documentos recuperados.

4. Testes

Os testes foram realizados em três fases distintas para avaliar a eficácia e a precisão do modelo RAG implementado:

Primeira Fase: Utilizando a configuração de chunk_size=512 e chunk_overlap=64, foram adicionados cinco documentos ao banco de dados vetorial. Para cada documento recém-adicionado, foi feita uma pergunta relacionada. Caso o RAG falhasse na primeira pergunta, uma segunda pergunta era feita.

Segunda Fase: O banco de dados foi reiniciado e repovoado com todos os documentos de uma vez, utilizando a configuração de chunk_size=2048 e chunk_overlap=128. As mesmas perguntas da primeira fase foram repetidas para avaliar a consistência dos resultados.

Terceira Fase: O banco de dados foi novamente reiniciado e repovoado com todos os documentos utilizando a configuração inicial de chunk_size=512 e chunk_overlap=64. Nesta fase, foram repetidas algumas perguntas que haviam obtido bons resultados na primeira fase, mas agora com todos os documentos já no banco, para verificar a reação do modelo quando há vários documentos presentes.

4.1 Processo de Teste:

Adição de Documentos: Em cada fase, os documentos foram divididos em chunks, vetorizados e inseridos no ChromaDB.

Consultas de Teste: Perguntas específicas foram feitas ao sistema para testar a capacidade do modelo de recuperar e gerar respostas precisas. As perguntas foram formuladas para cobrir tópicos presentes nos documentos.

Avaliação dos Resultados: As respostas geradas pelo modelo foram comparadas manualmente com as respostas corretas. Em caso de falha na recuperação de informações relevantes, foram anotadas as discrepâncias e analisadas as possíveis causas.

Exemplos de perguntas utilizadas:

"Onde Henrique Crispin de Aguiar nasceu?"

"Qual foi o ponto inicial da Segunda Guerra Mundial?"

"Quando a Batalha de Crécy aconteceu?"

"Quando Bordéus caiu para os franceses?"

"Quais foram as perdas humanas do Paraguai na Guerra do Paraguai?"

Exemplo de teste bem sucedido:

Pergunta: “Qual foi o ponto inicial da segunda guerra mundial?”

Resposta: “O ponto inicial da Segunda Guerra Mundial foi 1 de setembro de 1939, quando a Alemanha invadiu a Polônia.”

Exemplo de teste mal sucedido:

Pergunta: “Quando o Saturno V AS-506 lançou a Apollo 11?”

Resposta: “Não há informação no texto fornecido sobre quando o Saturno V AS-506 lançou a Apollo 11. O texto apresenta uma variedade de eventos históricos, mas não menciona esse assunto especificamente.”

5. Resultados

5.1 Primeira Fase

Os testes realizados com chunks de tamanho 512 e overlap de 64 demonstraram que o modelo de Recuperação de Respostas (RAG) teve um desempenho misto. Em perguntas específicas, como "Onde Henrique Crispin de Aguiar nasceu?", o modelo respondeu corretamente que ele nasceu em Guaraí, Tocantins. Da mesma forma, a pergunta "Qual foi o ponto inicial da Segunda Guerra Mundial?" foi respondida corretamente, indicando que a guerra começou em 1 de setembro de 1939, com a invasão da Polônia pela Alemanha.

No entanto, o modelo falhou em questões que exigiam informações não presentes no contexto fornecido. Por exemplo, na pergunta sobre a data da Batalha de Crécy, o modelo não conseguiu fornecer uma resposta, indicando que o contexto não incluía essa informação. Perguntas sobre eventos históricos menos conhecidos ou com menor contexto disponível, como as perdas humanas do Paraguai na Guerra do Paraguai ou o número de homens treinados no início da guerra, também resultaram em falhas do modelo.

5.2 Segunda Fase

Os testes com chunks de tamanho 2048 e overlap de 128 revelaram desafios semelhantes. Em perguntas repetidas, como sobre a Batalha de Crécy, o modelo novamente falhou ao indicar que o contexto fornecia informações sobre eventos espaciais e não sobre a batalha. Além disso, questões sobre as perdas humanas do Paraguai durante a guerra e o número de homens treinados também não obtiveram respostas satisfatórias, isso ocorreu porque o modelo foi incapaz de fornecer documentos relevantes para o contexto.

5.3 Terceira Fase

Todas as perguntas que foram bem sucedidas antes falharam aqui, mesmo estando com as mesmas configurações de chunk_size e overlap da primeira fase, mostrando que o modelo tem maior dificuldade em encontrar documentos relevantes quando sua base de dados possui mais vetores.

5.4 Visão Geral

Os resultados mostram que as configurações do tamanho e overlap dos documentos inseridos no banco de dados teve pouca ou nenhuma relevância nos nossos testes, porém a quantidade de documentos presentes no banco de dados foi um fator muito relevante para o sucesso do RAG, sendo que, aparentemente, quanto mais documentos presentes, menor a chance de fragmentos relevantes para o prompt do usuário serem recuperados (LEWIS et al., 2020). Em resumo, nossa análise constata que a função que procura vetores do chromaDB que são similares ao prompt foi a maior causadora de problemas, já que na maioria dos casos ela foi incapaz de retornar fragmentos similares ao prompt.

6. Considerações finais

Os resultados dos experimentos demonstram um padrão claro de desempenho do RAG, onde perguntas com respostas diretas e bem definidas dentro do contexto são respondidas com maior precisão. No entanto, a capacidade do modelo de lidar com perguntas que exigem inferência ou que tratam de tópicos não cobertos diretamente pelo contexto é limitada. O aumento do tamanho do chunk não necessariamente melhora a precisão das respostas, mas pode ajudar a capturar um contexto mais amplo, ainda que isso não garanta uma maior taxa de acertos.

Outro padrão observado é a dependência do modelo em relação à qualidade e à especificidade do contexto fornecido. Contextos ricos e diretamente relacionados à pergunta aumentam significativamente as chances de uma resposta correta (AMAZON WEB SERVICES, 2024). Esse padrão sugere que uma abordagem mais refinada na seleção e preparação do contexto pode ser uma chave para melhorar o desempenho do RAG. A precisão do RAG em responder perguntas factuais diretas é promissora, mas a capacidade de lidar com questões que exigem inferências ou detalhes não incluídos no contexto é limitada.

Um ponto positivo observado foi que o modelo não apresentou alucinações quando o RAG falhou. Em vez disso, ele corretamente indicou que o contexto fornecido não era suficiente para responder à pergunta, o que é uma característica importante para manter a confiabilidade do sistema.

Para futuras melhorias, é essencial focar na qualidade e relevância do contexto fornecido. Algumas direções possíveis para desenvolvimento incluem:

Aprimoramento dos Métodos de Seleção de Contexto: Desenvolver técnicas mais refinadas para selecionar e preparar o contexto, garantindo que informações críticas sejam incluídas de forma clara e acessível. Isso pode envolver o uso de algoritmos de aprendizado de máquina para identificar os fragmentos mais relevantes.

Melhoria da Função de Busca no ChromaDB: Melhorar os algoritmos de similaridade utilizados pelo ChromaDB para garantir que os documentos recuperados sejam altamente relevantes para o prompt do usuário. Isso pode incluir a implementação de novos métodos de cálculo de similaridade ou a incorporação de feedback iterativo.

Técnicas de Pós-Processamento: Explorar técnicas de pós-processamento das respostas geradas para refinar e validar as respostas antes de apresentá-las como finais. Isso pode envolver a revisão manual ou automatizada das respostas para corrigir possíveis erros e melhorar a clareza.

Integração de Fontes de Dados Adicionais: Ampliar a base de dados com mais fontes de informações relevantes pode ajudar a melhorar a qualidade do contexto disponível para o modelo, aumentando a precisão das respostas.

Em resumo, enquanto o RAG demonstra um potencial significativo, há áreas claras para melhorias que podem aumentar sua eficácia e precisão em aplicações práticas. Focar na qualidade do contexto e na precisão da recuperação de informações será fundamental para o desenvolvimento de modelos RAG mais robustos e confiáveis.

7. Referências

LEWIS, Patrick et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. [S. l.], p. 1-10, 12 dez. 2020. Disponível em: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf. Acesso em: 27 jun. 2024.

AMAZON WEB SERVICES. Retrieval-Augmented Generation. Disponível em: https://aws.amazon.com/pt/what-is/retrieval-augmented-generation/. Acesso em: 25 jun. 2024.

DUTRA, Marianne. Local RAG. Disponível em: https://github.com/mariannedutra/local-rag. Acesso em: 29 jun. 2024.