UNIVERSIDADE FEDERAL DE ALAGOAS
INSTITUTO DA COMPUTAÇÃO
CURSO SUPERIOR DE BACHALERADO EM ENGENHARIA DE COMPUTAÇÃO

Cicero Rogério Lima Tenório Filho

Virória Maria Santana Bigi
Antônio Wagner

RELATÓRIO PARCIAL LISTA 3

Maceió, AL
2025

RESUMO

RELATÓRIO PARCIAL LISTA 3

Cicero Rogério Lima Tenório Filho

Virória Maria Santana Bigi
Antônio Wagner
ORIENTADOR: Evandro de Barros Costa

Este relatório apresenta a análise conceitual e prática sobre agentes inteligentes, com foco em modelos baseados em LLM (Large Language Models) e RAG (Retrieval-Augmented Generation). A primeira parte discute os fundamentos teóricos de agentes inteligentes conforme a literatura clássica e aplica esses conceitos a exemplos reais, como assistentes jurídicos e chatbots. A segunda parte descreve o desenvolvimento de sistemas baseados em RAG com auxílio de LLMs, utilizando programação em pares com modelos como GPT-3.5 e Gemini-Pro. São detalhadas as fases de extração, indexação vetorial, recuperação de contexto e geração de respostas a partir de documentos PDF, além da comparação entre os dois modelos em termos de precisão, robustez e desempenho.

Palavras-chave: Agente Inteligente; LLM; RAG; GPT-3.5; Gemini-Pro; Processamento de Linguagem Natural; Recuperação de Informação; FAISS; sentence-transformers; PyMuPDF; Chatbot; Engenharia de Computação.

SUMÁRIO

1 QUESTÃO 1

1 - Descreva conceitualmente e apresente um exemplo de agente inteligente, conforme especificado no capítulo 2 do livro AIMA. Além disso, faça uma discussão conceitual e exemplique os conceitos envolvidos um agente baseado em LLM e RAG. Ademais, explique as semelhanças e diferenças entre essas duas noções de agente inteligente. Mostre e discute uma aplicação construída com cada umas das duas abordagens.

a) A definição apresentada por Stuart Russell e Peter Norvig de um agente inteligente pode ser vista conceitualmente como qualquer coisa que possa ser vista como percebendo seu ambiente por meio de sensores e atuando no ambiente por meio de atuadores. Já um exemplo de um agente inteligente pode ser um robô aspirador de pó, que teria seus senso- res de proximidade, sujeira, giroscópio e acelerômetro responsáveis por fazê-lo perceber o ambiente, e seus motores responsáveis por movê-lo e limpar o espaço.

b) Um agente baseado em LLMs e RAG combina a capacidade de processamento de linguagem natural de modelos de larga escala com a recuperação de informações externas. A LLM atua como núcleo do raciocínio, interpretando entradas textuais e gerando respostas. No entanto, como seu conhecimento é limitado ao treinamento, o mecanismo RAG complementa essa limitação ao buscar em tempo real documentos relevantes em uma base externa. O agente, ao receber uma pergunta, utiliza a etapa de recuperação para localizar trechos informativos que são então combinados com a entrada e processados pela LLM para produzir uma resposta fundamentada. Esse processo torna o agente mais preciso, atualizado e útil em tarefas que exigem acesso a conhecimento recente ou específico. Um exemplo é um assistente jurídico que, ao receber uma dúvida sobre prazos legais, recupera o artigo correspondente do código processual e gera uma resposta clara com base nesse conteúdo

c)Tanto o agente baseado apenas em LLM quanto o agente com RAG utilizam um modelo de linguagem para interpretar entradas e gerar respostas em linguagem natural. Ambos percebem o ambiente através de texto, processam essa informação e respondem com base em um raciocínio textual.

A principal diferença está na fonte de conhecimento utilizada. O agente baseado so- mente em LLM depende exclusivamente do que foi aprendido durante seu treinamento, o que o limita a informações disponíveis até a data de corte do modelo. Já o agente com RAG possui um mecanismo de recuperação que permite buscar informações em tempo real em uma base externa, como documentos, bancos de dados ou páginas da web, e fornece essas informações como contexto adicional para o modelo gerar uma resposta.

d) Se utilizarmos uma abordagem baseada apenas em uma LLM (Large Language Model), então podemos construir, por exemplo, um assistente de redação. Esse tipo de aplicação recebe instruções como “escreva uma introdução sobre mudanças climáticas” e,

com base no conhecimento aprendido durante o treinamento, gera textos coesos e bem estruturados. Como não depende de consultas externas, essa abordagem oferece respostas rápidas e fluidas. No entanto, se o tema exigido envolve informações recentes ou em constante atualização, então a aplicação pode apresentar dados desatualizados ou imprecisos, já que seu conhecimento é fixo no tempo do treinamento.

Por outro lado, se adotarmos a abordagem RAG (Retrieval-Augmented Generation), então podemos construir um chatbot de suporte técnico para uma empresa de software. Quando o usuário pergunta, por exemplo, “como configuro o backup automático no sistema X?”, o sistema primeiro recupera documentos técnicos e manuais atualizados da base de conhecimento da empresa e, em seguida, passa esses trechos relevantes para a LLM gerar uma resposta contextualizada e precisa. Se a aplicação precisa lidar com conteúdo específico, atualizado com frequência ou que não está presente no treinamento da LLM, então o uso de RAG permite maior confiabilidade e flexibilidade sem a necessidade de re-treinar o modelo.

Portanto, se o objetivo da aplicação é gerar textos criativos, bem estruturados ou com foco em conhecimento geral, então a abordagem com LLM pura é mais apropriada. Mas se o objetivo é fornecer respostas precisas baseadas em dados atualizados ou personalizados, então a abordagem RAG é a mais indicada.

ALGUNS OUTROS EXEMPLOS

1. **Aplicação com Agente Baseado em LLM (GPT-4, Gemini, etc.)**
   Exemplo: Chatbot de Suporte ao Cliente em um E-commerce

   Funcionamento Detalhado

   label=0. **Entrada do Usuário:**

   - “Meu pedido #12345 ainda não chegou. O que devo fazer?”

   lbbel=0. **Processamento pelo LLM:**

   - O modelo já foi pré-treinado em milhares de conversas de suporte.
   - Reconhece a intenção: “rastreamento de pedido + reclamação sobre atraso”.
   - Gera uma resposta baseada em seu conhecimento interno (sem consulta externa).

   lcbel=0. **Resposta Gerada:**

   - “Pedidos geralmente chegam em 5–7 dias úteis. Verifique o rastreamento [aqui]. Se o prazo passou, entre em contato com nosso time pelo email `suporte@loja.com`.”

2. **Aplicação com Agente Baseado em RAG (Retrieval-Augmented Generation)**
   Exemplo: Assistente Médico para Diagnóstico com Base em Pesquisas

   Funcionamento Detalhado

   label=0. **Entrada do Usuário:**

   - "Quais são os tratamentos mais recentes para diabetes tipo 2?"

   lbbel=0. **Busca em Banco de Dados (Retrieval):**

   - O sistema consulta fontes confiáveis (PubMed, diretrizes da OMS, artigos médicos).
   - Recupera os 3–5 documentos mais relevantes publicados nos últimos 2 anos.

   lcbel=0. **Geração da Resposta (Generation):**

   - O LLM sintetiza as informações em uma resposta clara:

   label=. Inibidores de SGLT2 (ex.: empagliflozina) para redução de risco cardiovascular.

   lbbel=. GLP-1 receptor agonists (ex.: semaglutida) para perda de peso e controle glicêmico.

   - **Fonte:** American Diabetes Association (2023).

2 QUESTÃO 2

2 - Considere os projetos a seguir e faça a escolha de dois deles para você (ou sua equipe) desenvolver completamente uma solução, usando um processo de desenvolvimento com a técnica de programação em pares, onde o outro par é um LLM (de sua livre escolha), gerando sistemas de software que funcionem. No desenvolvimento de cada projeto escolhido, realize o seguinte:

DETALHAMENTO DAS FASES

Fase 1 – Coleta e Processamento do Documento

O sistema começa com o upload de um arquivo PDF pelo usuário. Utiliza-se a biblioteca PyMuPDF para extrair o texto de cada página. O conteúdo textual completo é segmentado em trechos (chunks) de até 500 caracteres, permitindo melhor desempenho no processamento semântico posterior.

Fase 2 – Geração de Embeddings

Cada chunk é convertido em uma representação vetorial densa utilizando o modelo `all-MiniLM-L6-v2` da biblioteca `sentence-transformers`, que preserva informações semânticas necessárias para comparação e busca.

Fase 3 – Indexação Vetorial

Os embeddings são organizados em um índice vetorial com a biblioteca FAISS. Esse índice permite a recuperação eficiente de trechos mais relevantes a partir de uma consulta vetorial.

Fase 4 – Recuperação de Contexto (RAG)

A partir de uma pergunta feita pelo usuário, seu embedding é comparado com o índice FAISS para recuperar os trechos mais semanticamente próximos. Esses trechos formam o contexto usado pela LLM.

Fase 5 – Geração da Resposta

Com base no contexto recuperado, o sistema monta um prompt e o envia a uma LLM, podendo ser o `GPT-3.5-turbo` (OpenAI) ou o `Gemini-Pro` (Google). O prompt segue o padrão:

> Com base no seguinte conteúdo: [contexto recuperado], responda: [pergunta do usuário]

Fase 6 – Interação Conversacional

O usuário interage com o sistema via uma interface feita com `ipywidgets`, podendo selecionar entre os dois modelos LLM disponíveis. O agente retorna a resposta de forma

direta e textual, conforme o conteúdo extraído do PDF.

COMPARAÇÃO DAS LLMS: GPT-3.5 VS GEMINI-PRO

1. Precisão da Resposta

O GPT-3.5 tende a fornecer respostas mais diretas e estruturadas. Já o Gemini-Pro é mais descritivo e por vezes prolixo.

2. Relevância com o Conteúdo do PDF

Ambos os modelos conseguem gerar respostas coerentes com base nos trechos recuperados via RAG. No entanto, o GPT-3.5 apresenta maior precisão ao focar em informações pontuais.

3. Robustez e Tolerância a Erros

GPT-3.5 é mais robusto a perguntas mal formuladas. Gemini-Pro solicita reformulações com maior frequência.

4. Tempo de Resposta

O GPT-3.5 apresenta desempenho mais estável e respostas mais rápidas. O Gemini-Pro tem latência variável.

RESUMO COMPARATIVO

Critério	**GPT-3.5 (OpenAI)**	**Gemini-Pro (Google)**
Clareza da resposta	Alta	Média-alta
Detalhamento	Médio	Alto
Aderência ao contexto	Excelente	Boa
Tolerância a ruído	Alta	Média
Tempo de resposta	Rápido	Variável

Tabela 2.1 – Comparação entre os dois LLMs utilizados