Relatório do Agente

Entrega II

1. **Qual o framework?**

A solução utiliza o LangChain como a principal framework para desenvolver o agente de inteligência artificial. O LangChain é uma biblioteca Python projetada para facilitar a construção de aplicações baseadas em modelos de linguagem (LLMs), permitindo a integração com dados externos, ferramentas, e memória contextual. No caso do seu código, o LangChain é combinado com outras bibliotecas complementares para criar um agente que processa perguntas em linguagem natural e retorna resultados de consultas SQL em notas fiscais.

**2. Como a solução foi estruturada:**

Arquitetura do Sistema de Gerenciamento de Documentos Fiscais

* Agente 1 (Aquisição): Obtém notas fiscais eletrônicas (NF-e) em formatos de imagem ou PDF por meio de upload manual ou download automático de sistemas governamentais.
* Agente 2 (Extração e Treinamento): Processa os documentos adquiridos, aplicando OCR para extrair dados e utilizando um Large Language Model (LLM) para aprender novos layouts. Armazena os dados extraídos em formato estruturado, otimizando a precisão na captura de informações fiscais.
* Base de Conhecimento: Armazena os dados estruturados gerados pelo Agente 2, servindo como repositório central para consultas.
* Agente 3 (Interação e Resposta): Integra um LLM para responder a perguntas dos usuários, acessando a Base de Conhecimento e retornando informações fiscais em linguagem natural.

Interface do Usuário: Permite interação com o Agente 3, recebendo respostas baseadas nos dados processados.

Essa arquitetura automatiza o ciclo completo de aquisição, extração e consulta de documentos fiscais, garantindo eficiência e precisão.

Detalhes técnicos da framework:

• **LangChain:**

• Versão: Usa langchain-google-genai, que integra o modelo Gemini da Google (ChatGoogleGenerativeAI).

• **Componentes utilizados:**

• PromptTemplate: Define modelos de prompts para estruturar as entradas do LLM, com variáveis como {pergunta} e {df}.

• JsonOutputParser com Pydantic: Valida as saídas do LLM, garantindo que sigam formatos estruturados (ex.: Resposta com campo resposta: str ou Query com campo query: str).

• Cadeias (|): Combina PromptTemplate, LLM, e parser em pipelines para processar entradas e saídas de forma encadeada.

**• Modelo de linguagem:** ChatGoogleGenerativeAI com o modelo gemini-1.5-pro, configurado com temperature=0.5 para respostas balanceadas entre precisão e criatividade.

* **Outras bibliotecas complementares:**

• Pandas: Manipula dataframes para carregar arquivos .csv, validar dados, e retornar resultados de queries SQL.

• SQLAlchemy: Gerencia conexões com o banco de dados SQLite (nfs\_data.db) e executa queries SQL.

• Pydantic: Define modelos de dados (Query, Resposta) para validação estruturada das saídas do LLM.

• filetype: Possivelmente usado para validar arquivos (não explícito, mas incluído para compatibilidade).

• zipfile e re: Bibliotecas padrão do Python para descompactar arquivos .zip e usar expressões regulares na validação de nomes de arquivos.

**• Integração com Gemini:**

• A API do Gemini é acessada via ChatGoogleGenerativeAI, com a chave da API armazenada em Colab Secrets.

**• Ambiente de execução:**

• O código é projetado para rodar no Google Colab, com ajustes para caminhos de arquivos (usando os.path.join) e gerenciamento de chaves via google.colab.userdata.

• Há suporte para integração com um frontend, que fornece pergunta e diretorio, sugerindo que o agente pode ser usado em aplicações web ou desktop.

(Preencher aqui sobre o frontend)

**4 Perguntas e respostas:**

**Link para o Github:** https://github.com/ajndantas/I2A2-Grupo\_01/tree/master/Atividade dia 18/agente\_nfs

***Ocultar chaves utilizadas***