Prueba tecnica

Andrés Santiago Usma

August 2025

1 Solución

El repositorio se puede encontrar aquí. El agente IA fue diseñado con LangGraph, el grafo se puede observar en la figura 1.

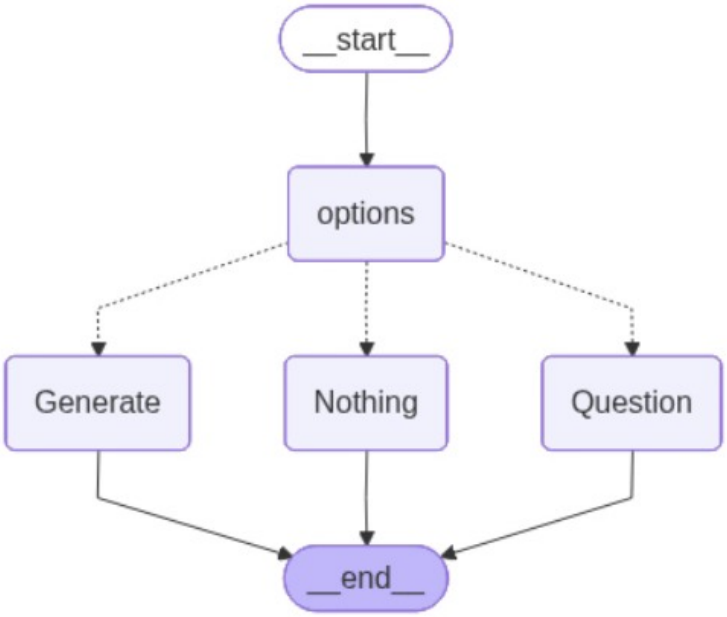

Figure 1: Grafo del agente IA diseñado con LangGraph.

El diseño está pensado como un Advanced RAG, donde la request inicial del usuario es analizado y procesado previo al Retrieval. En este caso, se proponen tres opciones, "Question" cuando el usuario realiza una pregunta sobre matemáticas, "Generate" cuando el usuario pide una guía de estudio, y "Nothing" cuando el usuario pide algo que no está relacionado con conceptos de matemáticas. El nodo "options" es un nodo condicional y la instrucción se observa en la figura 2.

```
sys_msg = """Eres un asistente que ayuda a definir que esta pidiendo el usuario. Esta petición puede ser una de las siguientes:
1. Question: El usuario esta preguntando sobre un concepto de matemáticas.
2. Generate: El usuario esta pidiendo que se genere un documento de ejercicios o explicativo.
3. Nothing: El usuario esta pidiendo algo que no tiene que ver con conceptos matematicos.

Responde solo con el nombre de la opción que corresponde (Question, Generate o Nothing) y nada más.
"""
```

Figure 2: Instrucciones del nodo "options".

Los detalles de cada opción y sus respectivas instrucciones son explicadas a continuación.

1.1 Question

El nodo "Question" realiza dos acciones, hacer retrieve del prompt del usuario a la base de datos Chroma, y contestar la pregunta del usuario. El nodo "Question" se puede observar en la Figura 3.

```
def question_node(state: State):
    retrieved_docs = vector_store.similarity_search(state["question"])
    docs_content = "\n\n".join(doc.page_content for doc in retrieved_docs)
    messages = prompt.invoke({"question": state["question"], "context": docs_content})
    response = llm.invoke(messages)
    print("question: ",response.content)
    return {"answer": response.content, "context": retrieved_docs}
```

Figure 3: Diseño del nodo "Question".

En este nodo se realiza el similarity_search que dará los chunks del documento que mayor similitud semántica tenga con el request del usuario. Luego, se usarán estos documentos para darle contexto al modelo. El "prompt" es obtenido de el repositorio de LangGraph y se puede observar en la Figura 4.

```
HUMAN
You are an assistant for question-answering tasks. Use the following pieces of retrieved context to answer the question. If you don't know the
answer, just say that you don't know. Use three sentences maximum and keep the answer concise.
Question: {question}
Context: {context}
Answer:
```

Figure 4: Prompt del retreival.

Una vez se tiene el prompt completo con el contexto necesario y la pregunta del usuario, se invoca el LLM y se responde la pregunta del usuario.

1.2 Generate

El nodo "Generate" realiza dos acciones, hacer retrieve del prompt del usuario a la base de datos Chroma, y generar un documento que sea una guía educativa. El nodo "Generate" se puede observar en la Figura 5.

```python
def generate_node(state: State):
    retrieved_docs = vector_store.similarity_search(state["question"])
    docs_content = "\n\n".join(doc.page_content for doc in retrieved_docs)
    messages = prompt.invoke({"question": state["question"], "context": docs_content}).to_messages()
    print("messages: ",messages, "hm_generate_msg: ",hm_generate_msg)
    response = llm.invoke(messages + [hm_generate_msg])
    string_to_pdf(response.content, "guia_estudio.pdf")
    return {"answer": "El documento se ha generado en guia_estudio.pdf", "context": retrieved_docs}
```

Figure 5: Diseño del nodo "Generate".

La logica del retrieval es igual que la del nodo "Question". Sin embargo, ya no se busca responder una pregunta sino generar un documento, por lo que se agrega la siguiente instrucción:

"El usuario esta pidiendo generar una guia de estudio. Dale un texto detallado que contenga las definiciones de los conceptos claves, algunos ejemplos y ejercicios, incluye cualquier detalle que el usuario haya pedido."

Cualquier texto que el modelo retorne, se guardará en un pdf.

1.3 Nothing

En este nodo se busca rechazar la petición del usuario para controlar las capacidades del modelo. De esta forma, el modelo de lenguaje no contestará cualquier petición que se la haga, y tendremos mayor control de lo que puede o no hacer el agente. El diseño del nodo se observa en la Figura 6 y la instrucción del nodo es la siguiente:

"Eres un asistente que ayuda a responder preguntas generales que no tienen que ver con conceptos matemáticos. Explícame brevemente porque mi petición no tiene que ver con conceptos matemáticos."

```python
def nothing_node(state: State):
    x={"answer": llm.invoke([state["question"]] + [hm_nothing_msg]).content}
    print("nothing: ",x["answer"])
    return x
```

Figure 6: Diseño del nodo "Nothing".

En este nodo se le proporciona la pregunta al modelo y la instrucción del rechazo para que la respuesta sea más directa.

2 Aplicación

La interfaz fue diseñada con streamlit. La interfaz puede observarse en la Figura 7. Adicionalmente, de la Figura 8 a la Figura 10 se pueden observar las respuestas del modelo y como son mostradas en la interfaz.

Figure 7: Diseño de la interfaz en Streamlit.

El documento PDF generado por el modelo se encuentra adjunto con este documento y en el repositorio.

3 Bonus

3.1 Base de datos vectorial

La base de datos vectorial que usa mi código es Chroma. Esta base de datos funciona localmente, por lo que se requiere un código adicional para dividir, procesar y guardar los documentos en la base de datos vectorial. Esta función ya permite cargar múltiples documentos en una misma base de datos. Sin embargo, si deseamos tener una base de datos vectorial en la nube, existen varias soluciones, yo he trabajado con Qdrant que te permite tener una instancia y guardar la base de datos vectorial en la nube. LangGraph permite la conexión con esta nube para subir o pedir documentos usando la API key.

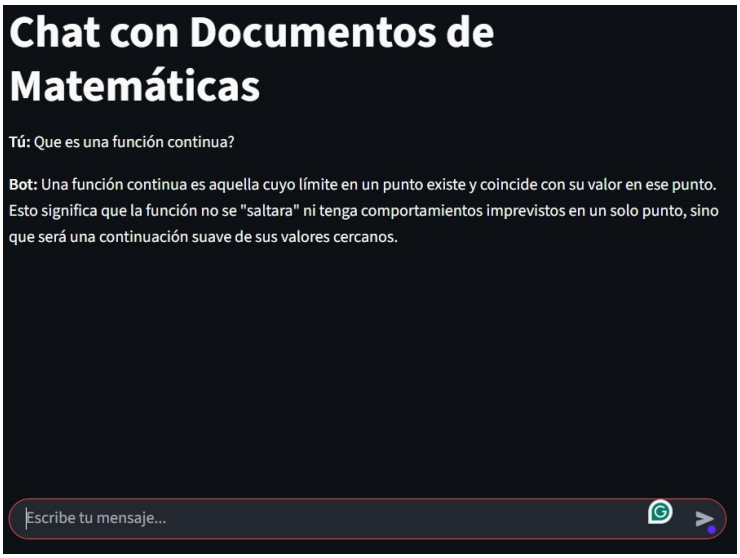

Figure 8: Respuesta del modelo a una pregunta del usuario.

3.2 Pruebas de carga

El mayor tiempo de generación se da en la petición de "Generate". La mayor cantidad de tokens a generar y la posible mayor cantidad de documentos a analizar es la razón de este delay. Adicionalmente, el preprocesamiento del request del usuario es un paso adicional que requiere análisis del modelo, aproximadamente 3 segundos en mi computador, sin embargo, es un procesamiento necesario para controlar mejor el flujo de generación.

3.3 Métrica de predicción

No estoy al tanto de métricas de predicción para modelos de lenguaje para problemas matemáticos, sobre todo métricas automáticas. Aun así, una idea que propongo es el uso de otro LLM para analizar las respuestas del primero agente. Este nuevo modelo sería capaz de obtener información de la base de datos o de internet para analizar y validar la primera respuesta del modelo, algo similar a un Speculative RAG.

3.4 Seguimiento

LangGraph tiene la posibilidad de supervisar el flujo de pensamiento del agente y su tiempo de procesamiento usando LangSmith. Esta aplicación permite observar la respuesta del LLM en cada nodo para mejorar el desarrollo del agente y supervisar las respuestas que el mismo da. Dentro de mi código no agregué la posibilidad de conectar con LangSmith pero agregué algunos prints que permiten monitorizar la ejecución del agente en la terminal.

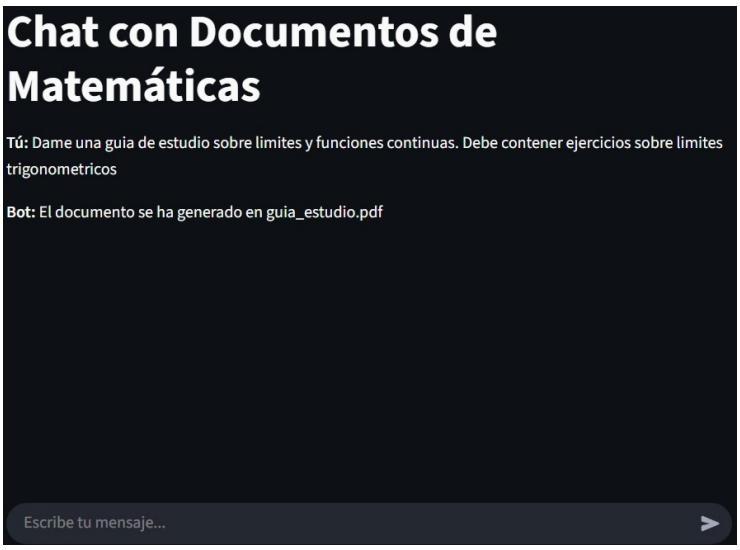

Figure 9: Respuesta del modelo a la petición de generación de una guía educativa.

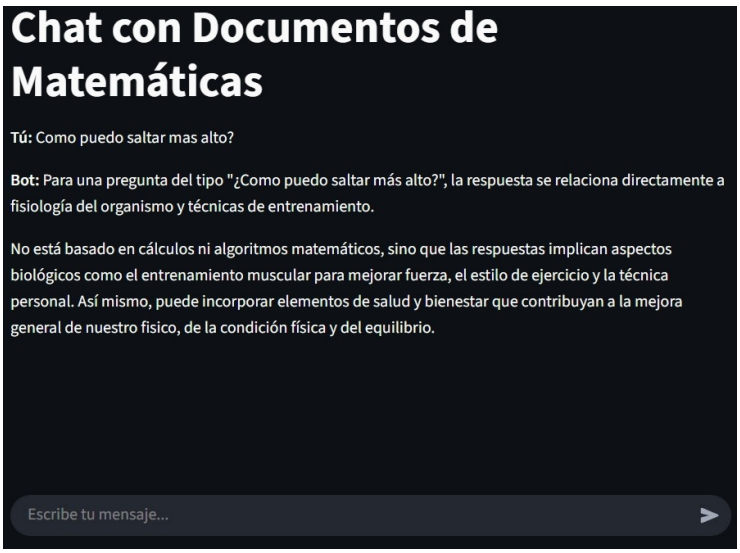

Figure 10: Respuesta del modelo a la petición no relacionada con temas matemáticos.