Juan Matias Rossi

El primer día busqué información sobre LangChain, cómo aplicar RGA y como podía utilizar la api LangServe. La documentación en un principio me había parecido bastante sencilla, pero cuando me informé más mediante videos y documentación no oficial me confundí un poco con la cantidad de opciones que se pueden utilizar a la hora de crear Chains o Runnables.
El segundo día había planeado utilizar AWS Amplify para el front, AWS Secrets para alojar mi OpenAI key y AWS Lambda para la API. Después estuve tres días básicamente entendiendo lo que quería hacer, más o menos explico la idea en el repositorio. Al tener tan poco contexto aplicar RAG es limitar a la inteligencia artificial generativa a ese precario contexto, si especificas un prompt para solamente el contexto no das opciones de generar otra respuesta en base a su posterior entrenamiento. Entonces me decidí a implementar una solución con multi cadenas, especificando los prompts y ajustando la temperatura del modelo. El punto negativo que le encontre a esta solución es que es muy difícil mantener una coherencia si se le aplica un historial de chat, la solución pedida en la tarea es aplicar una Conversational Retrieval Chain y obviamente que un chatbot tenga acceso al historial de respuestas es importante. En el código del repositorio tengo comentado cómo sería la cadena en caso de aplicar Conversational Retrieval Chain, en el despliegue utilice la solución comentada anteriormente.
Solamente quería aclarar este punto, soy consciente de ambas opciones, soy capaz de aplicar ambas, pero para casos en los que hay poco contexto esta solución me pareció más interesante. El sexto día tuve varios problemas con AWS Lambda por la dificultad a la hora de hacer debugging y monitorear la instancia. Rápidamente me decidí a utilizar Koyeb por primera vez, muy comprensible, sencillo de monitorear y gratis. También tuve un pequeño problema con AWS Amplify, una variable de entorno (url de la API) era detectada sin problemas pero no me dejaba utilizarla para un fetch porque podía ser “riesgoso”, se solucionó al poner plenamente el string.
En general fue una experiencia muy divertida, a pesar de que LangChain está en constante evolución uno puede encontrar información en un margen de hace 4 meses a 5 días. RAG es una metodología bastante barata, flexible y confiable. Me dejó con ganas de ver más tipo de soluciones y qué características comparten o difieren.

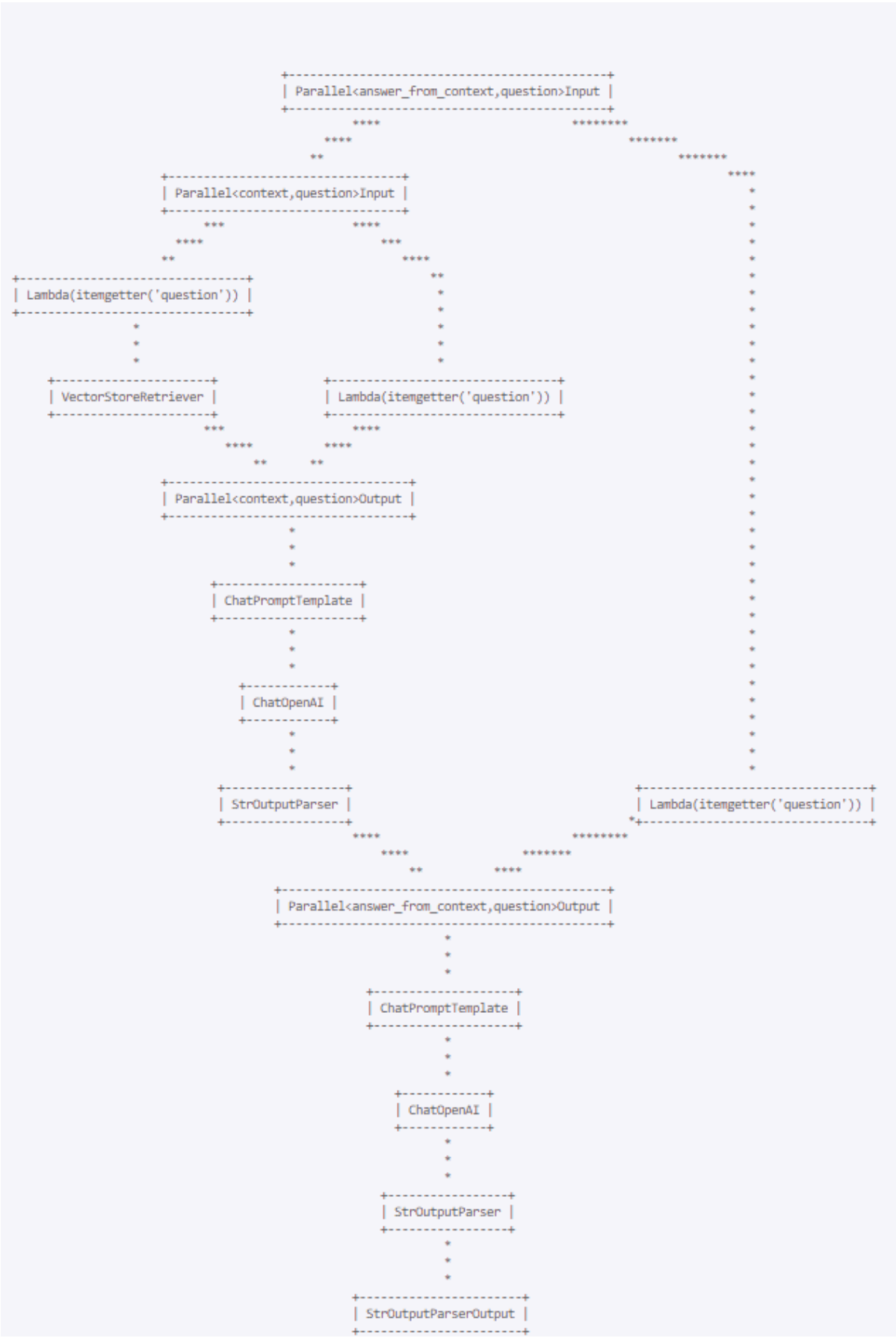
Github: https://github.com/SchneiderSix/LangChain-RAG-LangServe
Deployment: AWS Amplify (frontend), Koyeb(backend)
https://master.d3bilk665bedk0.amplifyapp.com/

Componentes:

Primero pasa por los middlewares (CORS, tocken_bucket rate limiter), luego pasa por el endpoint invoke de /qp con el header input, key-value question. Esa ruta llama a la cadena **re_do_chain**, esta tiene dos argumentos, "answer_from_context" (el valor de este argumento se consigue llamando a la cadena **context_chain**) y "question" (el valor es la pregunta que pasa por el header de la llamada). En el esquema podemos apreciar que el camino de la izquierda es la ruta que se sigue al obtener un valor para "answer_from_context". En esta cadena **context_chain** el argumento "context" es una mezcla entre la pregunta en sí y nuestro retriever (nuestro contexto "procesado"). El parámetro "question" se obtiene también del header de la llamada, con estos argumentos la primera cadena utiliza el prompt (prompt1) especificando que se busca una respuesta de determinado contexto.

Luego se llama a la API de OpenAI y se utiliza un parser (función StrOutputParser) para convertir la respuesta en String. Este primer resultado se utiliza para el argumento "answer_from_context", el argumento "question" es el mismo que se obtiene inicialmente del header, se utiliza otro prompt (prompt2) para esta segunda llamada a la API de OpenAI y al final obtenemos el resultado que es enviado al usuario.

Abajo se aprecia el esquema de la cadena **re_do_chain**.


Parallel<answer_from_context,question>Input
Parallel<context,question>Input
Lambda(itemgetter('question'))
VectorStoreRetriever
Lambda(itemgetter('question'))
Parallel<context,question>Output
ChatPromptTemplate
ChatOpenAI
StrOutputParser
Lambda(itemgetter('question'))
Parallel<answer_from_context,question>Output
ChatPromptTemplate
ChatOpenAI
StrOutputParser
StrOutputParserOutput