

**User:**

Hello.

**Assistant:**

Hello! Thank you for taking the time to speak with me today about the Entry-Level RAG AI Engineer role. I'd like to start by asking you a few questions about your experience and skills. Could you tell me about any projects you've worked on involving retrieval-augmented generation (RAG) pipelines?

**User:**

I'm sure so I'm studying artificial intelligence at the University of Kent currently and for my final dissertation. I'm working on making a automated screening Interview agent and to implement this I have used a rag pipeline mainly as the accuracy verifier so what happens is when the Candidate answers their questions it goes through two pipelines one is the sentiment analysis and one is the accuracy verifier For the accuracy verifier I have implemented a retrieval augmented generation, which would basically break down the answer into separate Searchable strings which will then be searched on Google and The first two articles it will retrieve the contents of the first two articles and input that in the context of the LLM So the LM has more up-to-date information to verify with the whether the answer from the candidate is accurate or not and to give an accuracy percentage

**Assistant:**

That's an interesting project. Can you elaborate on the specific challenges you faced while implementing the RAG pipeline for your accuracy verifier? How did you address issues like retrieval quality or context relevance?

**User:**

Yes, so to improve the context or the retrieval quality of the rag pipeline, I had to break down the answer from the candidate into searchable strings with the help of an LLM. So let's say an answer can be broken down into six query strings. Each of these six query strings would then be used to search in Google and we would draw the context from the first two web pages. So in a total we would get the information from a total of 12 web pages for one answer. So this I think is plenty of information to feed the LLM. This answer, this document would then be stored in a vector store and when the LLM would be queried on a specific topic or like when the LLM wanted to verify the accuracy of a certain answer it would then use a cosine similarity to find out the relevant portions of the vector store that are relevant to the answer. And doing this, it would vastly improve the quality of the answers fetched. I got this from a paper written by Google called Query Decomposition. This was the technique they used and this overcame the shortcomings of just searching for two or three websites instead of getting a more holistic picture of the entire topics being discussed in the answer.

**Assistant:**

That's a sophisticated approach. How did you handle the integration of this RAG pipeline with the large language model? Were there any specific challenges in terms of prompt engineering or model selection?

**User:**

For model selection, we chose JATGBD 4.0 mini mainly because we used Langchain to implement the rank pipeline and GPT 4.0 mini had the perfect balance of intelligence and cost effectiveness and also speed that we had to manage. And this was just to verify the answer. So we did not go for a more sophisticated model such as Claude SONET 3.5 which is considered the most intelligent LLM till now. We did not need such a high powered LLM, we just needed a cost effective LLM to just verify the answer and make searchable strings and JATGBD 4.0 mini was perfect for the job. Apart from this, for prompt engineering, yes, I had to write several prompts to give the last rank pipeline to verify the answer. So what would happen is when we converted speech to text from the interview, some of the text had grammatical errors or typographical errors which is common for most text translation apps. So to overcome this, I had to prompt the LLM to specifically overlook grammatical errors or to make sense of words that were not properly converted but were close to the actual word that the candidate was trying to explain. So these were some of the challenges that I faced.

**Assistant:**

Thank you for sharing those details. Can you discuss any experience you have with optimizing model performance, particularly in terms of speed and cost efficiency?

**User:**

So far I have not optimized any model. By optimizing I am thinking you mean fine tuning model. So for the specific project fine tuning was not necessary. However, we had to determine which model best suited the specific area of our project. So for example, for the real time conversation where the LLM had to generate questions and interact with the candidate, we went with Claude 3.5 Sonnet which is the most intelligent LLM till date as preferred by most developers. And again for the accuracy verifier we went with ChatGPD 4.0 Mini which is a cut down version of ChatGPD 4.0 which itself is a very powerful LLM. However, 4.0 Mini has the right balance of intelligence and cost effectiveness and also speed. Then for the sentiment analysis we went with Hume AI which is an external service that does the sentiment analysis directly from audio and video feed. So the service, we don't know the specific implementation of the service because we are paying to use the service. And after that getting the sentiment and accuracy verifier score we then feed it into Claude Sonnet 3.5 again to make sense of the answers that the candidate made from both the accuracy verifier and from the sentiment analysis and to give the final verdict of the candidate. So these are the main considerations we made when choosing an LLM.

**Assistant:**

Thank you for your time and detailed responses. We'll be in touch soon regarding the next steps in the interview process.