**Technical Assessment Project (AI Architect)**

We have been given a task to create an AI based system which can intake a huge number of resumes and process and store them and can select the best resume or set of resumes which match our Job description the best.

Solution: For this type of system the best approach can be using Sentence embeddings to capture the semantic meaning of the entire sentence or document as a whole. In a resume, you often find technologies (e.g., Python, Django, AWS) grouped together in specific contexts

For example:

-> Experienced in Nodejs, Express, AWS, and React

-> Worked on projects using Java, Spring Boot, and MySQL

These technologies are grouped together based on relatedness and context. We definitely need our system to perform document matchings based on semantic search with good performance, precision and accuracy.Sentence embeddings (like those from Sentence-BERT) are designed to understand these relationships across multiple words or phrases within a sentence, which makes them particularly good for our use case where technologies are often listed together.

**LLM Selection Approach:**

For this use case we will be using **distilbert-base-nli-stsb-mean-tokens** which is popular pre-trained model for semantic similarity and sentence-level embeddings, primarily fine-tuned for Natural Language Inference (NLI) and Semantic Textual Similarity (STS) tasks.Also It is based on Distillbert which is very good for being smaller and very good in performance while still retaining 97% of BERT's language understanding ability in NLP Tasks

**Why use mean-tokens?**

Averaging the tokens allows the model to capture the overall semantic meaning of the sentence in a compact, fixed-size vector (usually a 768-dimensional vector for DistilBERT), which is excellent for tasks like document matching, search ranking, and clustering, where you need to compare entire sentences or paragraphs.

**How to ensure Job Descriptions (JDs) with resumes are accurate and of high quality?**

**Normalisation**: normalize\_embeddings() function to normalise vectors, which helps maintain consistency in how embeddings are compared.

**Performance Analysis**: We are analysing F1 score, Mean Reciprocal Rank(MRR), precision and recall for getting a better idea about performance of our semantic search results.

**Better Filtering and Matching**: By combining cosine similarity scores with confidence thresholds, the system improves in terms of both quality (relevant matches) and user control (confidence).

**Index Cleaning**: The code ensures that duplicates or invalid entries are removed from the FAISS index, which helps in maintaining an efficient and clean index.

**Real Time Project Updates**

1. First we load the Resumes from our local store. For this we are using a folder “resume-store” to store all our resumes.We use Fitz library based on PyMuPDF library for pdf loading and data extraction with function => extract\_text\_from\_pdf(file\_path)
2. Next we sanitise text from PDF for generating embeddings for our cleaned out text.
3. We have to define a function to calculate cosine similarity between our JD and the resumes embeddings.
4. Sentence-BERT model (like distilbert-base-nli-stsb-mean-tokens) is one of the best models for semantic search because it’s designed for measuring sentence similarity. These embeddings represent the semantic content of each resumein a high-dimensional vector space and we can compare these resumes easily and accurately with great search performance.
5. Choosing a Vector Database : You can use vector databases like FAISS, Annoy, Milvus, or Pinecone to store and retrieve the embeddings. FAISS vector db is widely known for high-dimensional similarity search with fast and accurate results, so we’ll proceed with FAISS.
6. Storing Embeddings in FAISS will allow us to efficiently store vectors in an index and perform similarity searches.
7. When performing semantic search, the goal is to return the most relevant and accurate matches based on the semantic similarity between the job description (JD) and the resumes.
8. Here we are using Sentence-BERT to encode the raw text into a fixed-length vector.

DB selection thought process  
1. DB Selection - FAISS is optimised for speed and scalability and also support for flexible index types and is very good for performance with versatility in index types and built in ANN search and is also good for High dimensional vectors.

2. Index type - The vectors are stored in a flat array, and searches are conducted by simply calculating the cosine distance to all vectors in the index.We chose flat vector over HNSW, inverted file index

2.Deciding Distance search algorithm - We went with cosine similarity because cosine similarity is especially useful in semantic search vs L2 Distance which performs better for magnitude related search operations.

**Problems Encountered while developing the code:**

1. Initially started with BERT uncased, but eventually found better results and performance with using mean tokens based bert model with the help of sentence transformers.
2. Started with testing L2 distance implementation but had to switch to cosine similarity due to better semantic performance and results.
3. Initially had started with Chromadb but Faiss became a natural choice due to ease of use and better flexibility and support IVF (Inverted File System) for faster indexing with accuracy and HSNW (Hierarchical Navigable Small World graphs)