**Resume Matching and Scoring System**

**Architecture Diagram**



**Component Design:**

* Job description parser : Passing the Job description as User Prompt into a finetuned Large Language Model, instructing it to extract Job Title, Location, Industry/ Domain, Education Degree Required, Technical Skills Required from the given job description.
* Resume Parser : Used the Same Approach as Job description parser. Additional task of reading the .doc file and extracting the content contained in resume. Passed the Content into the same model to extract the required attributes
* Matching Algorithm : Found similarity between respective attributes by passing attributes text into a pretrained llm and obtained text’s embedding from last layer of the model. Applied cosine similarity on normalized embeddings of same attributes and obtained scores for each attribute. Assigned some weights to each attribute, multiplied attribute cosine similarity by its weights and summed up the total score. Returned final score by multiplying by 100 to scale between 0-100.

**Details on extracting and matching the attributes between job description and resume:**

A single prompt instructing the model to extract all the attributes jointly works well. Alternate strategy was to extract attributes separately by passing the JD or Resume multiple times which is inefficient. I used a pretrained **Qwen1.5-0.5B Chat** model from **Hugging Face.** Using a bigger parameter model such as a 7B or 14B model can give more consistent results. Loading model every time file is run is inefficient and a API call could be used instead, which would require some amount of money.

After Extracting attributes, their embeddings were obtained by utlizing a pretrained model again from hugging face using its sentence transformers library. Then similarity was simply found between respective attributes by finding cosine similarity on normalized embeddings.