**Task 1: Defining your Problem and Audience**

An organization does not have a common system for their employees to raise the query or support tickets for different departments like HR, IT, Transportation etc.

With the current system in place, an employee must contact the respective departments to get the answer for their queries or raise the support ticket. Likewise, the representatives of these departments do not have a centralized system to address the users query or support ticket. Due to this managing the support ticket and answering user’s queries has become difficult.

**Task 2: Propose a Solution**

In this project I am designing a Customer Support Agent that can be used by an employee of a company to answer questions related to different departments like Human Resource, IT Support, Transportation etc. This is a generic system which has knowledge of the policies of all departments within an organization so that employees do not have to reach out to the departments separately for their queries.

The employee can also raise a support ticket for their queries using this system. The Customer Support Agent also classifies the ticket into different categories like HR, IT, Transportation and others. This helps the representative of these departments to answer the tickets raised for their respective departments. If the users query has negative sentiment, the query is routed to human agent.

*Describe the tools you plan to use in each part of your stack. Write one sentence on why you made each tooling choice:*

1. LLM: I am using gpt-4o-mini as a model. I have chosen gpt-4o-mini for efficiency and speed, cost-effectiveness, lower resource requirements
2. Embedding Model: I am using Open AI’s text-embedding-3-small embedding model. I have chosen to use text-embedding-3-small embedding model for high efficiency, reduces latency, cost-effectiveness and good performance. I have fine-tuned the snowflake-arctic-embed-I embedding model. Open source embedding model provides several advantages like cost effectiveness, flexibility and customization option.
3. Orchestration: I am using Lang Graph because the workflow in my application is hierarchical or tree-like structure with conditions. This can get complex as I add more features to my application. LangGraph is a good choice for such workflows.
4. Vector Database: I have chosen Qdrant as a vector store because it is highly performant for similarity searches, can handle large volumes of vectors and indexing high dimensional data.
5. Monitoring: LangSmith because it provides a comprehensive platform to track and analyze the performance of LLM
6. Evaluations: Using RAGAS for RAG evaluation metrics. RAGAS can handle large datasets effectively and it pinpoints areas where improvement is needed.
7. User Interface: Streamlit because of it simplicity. Streamlit allows us to build full-featured web applications using just Python.
8. Inference & Serving: Hugging Face because it is fast and cheap to host. I have used HF to host the application and the fine-tuned embedding model.

*Where will you use an agent or agent? What will you use “agentic reasoning” for in your app?*

I have not used agents or agents in the app but plan to use it in future. I plan to create agents for different departments in the organization.

**Task 3: Dealing with the Data**

*Describe all of your data sources and external APIs and describe what you’ll use them for.*

Data source: I will be using the HR Policy Manual, IT Department Policy Manual and Transportation Policy Manual for the RAG to answer users’ queries about these departments. I am not using any external API.

External APIs: I haven’t used external APIs in my app, but plan to use Tavily API or other API in future versions

Chunking strategy: I am using RecursiveCharacterTextSplitter for chunking because the RAG system needs long documents and overlapping chunks. RecursiveCharacterTextSplitter recursively tries to split using multiple delimiters. Using this splitter helps ensure chunks contain complete ideas, improving retrieval accuracy.

**Task 4: Building a Quick End-to-End Prototype**

I have deployed my application to Hugging Face but I am seeing an error when I upload PDF file using Load Documents tab. I was not able to resolve it. My application runs fine locally without an error.

HF app link: <https://huggingface.co/spaces/deepali1021/Midterm-streamlit>

**Task 5: Creating a Golden Test Data Set**

*Assess your pipeline using the RAGAS framework including key metrics faithfulness, response relevance, context precision, and context recall. Provide a table of your output results*.

| **Metric** | **Value** |
| --- | --- |
| Context Recall | 0.8735 |
| Faithfulness | 0.8249 |
| Factual Correctness | 0.6620 |
| Answer Relevancy | 0.9557 |
| Context Entity Recall | 0.5015 |
| Noise Sensitivity (Relevant) | 0.2298 |

Context Recall:   The pipeline is quite effective at recalling relevant context from the input data. The system is good at understanding and retaining the important parts of the input.

Faithfulness: Pipeline shows high value of faithfulness to the input. The output generated is accurate and true to the source material.

Factual Correctness: This score is low. There needs to be an improvement in response generated so that it aligns with the reference.

Answer Relevancy: This is a very high score, indicating that the answers generated by the pipeline are highly relevant to the questions asked.

Context Entity Recall: This is low and needs to be improved. It is a measure of what fraction of entities are recalled from reference

Noise Sensitivity (Relevant): Noise Sensitivity (Relevant) is a metric that evaluates how robust the retrieval system is to irrelevant or noisy data. A low score means it’s robust against noise.

**Task 6: Fine-Tuning Open-Source Embeddings**

Link to the fine-tuned embedding model on Hugging Face: <https://huggingface.co/deepali1021/finetuned_arctic_ft-v2>

**Task 7: Assessing Performance**

*How does the performance compare to your original RAG application? Test the fine-tuned embedding model using the RAGAS frameworks to quantify any improvements. Provide results in a table.*

| **Metric** | **Value (Non fine tuned)** | **Value (Fine tuned)** |
| --- | --- | --- |
| Context Recall | 0.7833 | 0.9500 |
| Faithfulness | 1.0000 | 0.9870 |
| Factual Correctness | 0.7067 | 0.7267 |
| Answer Relevancy | 0.8480 | 0.8492 |
| Context Entity Recall | 0.5771 | 0.5092 |
| Noise Sensitivity (Relevant) | 0.1745 | 0.1389 |

*Articulate the changes that you expect to make to your app in the second half of the course. How will you improve your application?*

I plan to use agents to handle support tickets for different departments and use an external API in my app. I also plan to refactor the code to make it maintainable and readable.

**Lang graph diagram:**

<https://github.com/Deepali-Khalkar/Midterm-project/blob/main/Langgraph_diagram.png>

**Link:**

Github link: <https://github.com/Deepali-Khalkar/Midterm-project>

Loom: <https://www.loom.com/share/9fa2d8e54f8048cf860a8f777199fc64?sid=4e75e574-200d-46ff-8805-caa5e4ddd3f1>

RAGAS evaluation notebook: <https://github.com/Deepali-Khalkar/Midterm-project/blob/main/RAGAS_evaluation_Midterm.ipynb>

Finetuning embeddings notebook: <https://github.com/Deepali-Khalkar/Midterm-project/blob/main/Fine_tuning_Embeddings_Midterm.ipynb>

HF app link: <https://huggingface.co/spaces/deepali1021/Midterm-streamlit>

HF fine-tuned embedding model link: <https://huggingface.co/deepali1021/finetuned_arctic_ft-v2>