AI Assignment from Vijayi WFH Technologies Pvt Ltd

Submission of both the tasks - Task 1 & Task 2 is compulsory.

TASK – 1

Problem Statement

Objective:
Develop a machine learning pipeline that classifies customer support tickets by their issue type and urgency level, and extracts key entities (e.g., product names, dates, complaint keywords). The file (ai_dev_assignment_tickets_complex_1000) is provided.

Assignment Tasks

1. Data Preparation

- You are provided with an Excel file (ai_dev_assignment_tickets_complex_1000) containing anonymized customer support ticket data with the following columns:
    - ticket_id
    - ticket_text
    - issue_type (label)
    - urgency_level (label: Low, Medium, High)
    - product (ground truth for entity extraction)
- Clean and preprocess the data, including but not limited to:
    - Text normalization (lowercase, removing special characters)
    - Tokenization, stopword removal, lemmatization
    - Handling missing data

2. Feature Engineering

- Create meaningful features using traditional NLP techniques:
    - Bag-of-Words, TF-IDF, or similar
    - Extract additional features such as ticket length, sentiment score, etc.
- Justify your feature selection in the documentation.

3. Multi-Task Learning

- Build and train **two separate models**:
    1. **Issue Type Classifier**: Predict the ticket's issue_type (multi-class classification)
    2. **Urgency Level Classifier**: Predict the urgency_level (multi-class classification)
- Use any classical ML models (e.g., Logistic Regression, SVM, Random Forest, etc.).

4. Entity Extraction

- Using traditional NLP, extract key entities from ticket_text:
    - **Product names** (from a provided product list or using regex/rule-based methods)
    - **Dates** mentioned in the text
    - **Complaint keywords** (e.g., “broken”, “late”, “error”)
- Return extracted entities as a dictionary.

5. Integration

- Write a function that takes raw ticket_text as input and returns:
    - Predicted issue_type
    - Predicted urgency_level
    - Extracted entities (as a JSON/dictionary)

6. Gradio Interface (Optional)

- Build an interactive Gradio web app where users can:
    - Input raw ticket text
    - See the predicted issue type, urgency, and extracted entities as output

7. Documentation & Submission (Each of the following points is mandatory)

- Briefly document your code: key design choices, model evaluation (with metrics), and limitations (README or notebook markdown).
- A short **screen recording demo video** explaining the code and the output coming from Gradio. You can upload the video on Google Drive (and share access) or any of the many sites that are available these days.
- Submit:
    - All code (Jupyter notebook or .py scripts)
    - Demo Video
    - README with instructions to run the code and app

Bonus (Optional)

- Add visualizations of ticket distributions, feature importances, etc.
- Add confusion matrices and classification report.
- Allow batch processing of multiple tickets via Gradio.

Evaluation Criteria

- Code quality, clarity, and organization
- Soundness of data preprocessing and feature engineering
- Correct use of traditional NLP and ML models
- Robustness and accuracy of predictions (use cross-validation or a test set)
- Usability and look of the Gradio app
- Ability to explain your approach and choices
- Documentation and demo video.

TASK – 2

RAG-Based Semantic Quote Retrieval and Structured QA with Model Training

Problem Statement

Objective:

You are tasked with building a **semantic quote retrieval system** powered by RAG (Retrieval Augmented Generation) using the above dataset.
The workflow includes **fine-tuning a model** on the dataset, integrating with a RAG pipeline, evaluating RAG and deploying an interactive **Streamlit** app.

Assignment Instructions

1. Data Preparation

- Download and explore the Abirate/english_quotes (https://huggingface.co/datasets/Abirate/english_quotes) dataset from HuggingFace.
- Preprocess and clean data as needed for model training (tokenization, lowercasing, handling missing values, etc.).

2. Model Fine-Tuning

- Fine-tune a sentence embedding model (e.g., sentence-transformers) or a relevant transformer (e.g., DistilBERT, MiniLM, or similar) on the quotes.
    - Task: Given a query (e.g., "quotes about hope by Oscar Wilde"), the model should retrieve relevant quote, author, and tags.
- Save the fine-tuned model.

3. Build the RAG Pipeline

- Implement a **Retrieval Augmented Generation** pipeline:
    - Use your fine-tuned model to encode and index the quotes (e.g., with FAISS, ChromaDB, etc.).
    - Use a Large Language Model (e.g., OpenAI GPT-3.5/4, Llama2/3, or open-source) to answer natural language queries using retrieved quote context.

4. RAG Evaluation

- Testing a RAG system consists of running a set of queries against the tool and evaluating the output.
- Use **any ONE** of the framework for RAG evaluation –
    - RAGAS
    - Quotient
    - Arize Phoenix

5. Streamlit Application

- Build a user-friendly **Streamlit app** that allows:
    - User to input natural language queries (e.g., “Show me quotes about courage by women authors”)
    - System retrieves relevant entries from the fine-tuned + indexed dataset.
    - Structured JSON response displayed (quotes, authors, tags, summary).
    - Optionally, display source quotes and their similarity scores.

6. Deliverables

- **Jupyter/Colab/Kaggle Notebook(s) or .py files** for:
    - Data prep & model fine-tuning
    - RAG pipeline implementation
    - RAG evaluation
    - Streamlit app
- **Evaluation results**: a short discussion.
- **README**: Clear instructions on running your code, model architecture, design decisions, and challenges.
- A short video of a code walkthrough and testing.

Example queries for evaluation:

- “Quotes about insanity attributed to Einstein”
- “Motivational quotes tagged ‘accomplishment’”
- “All Oscar Wilde quotes with humor”

Evaluation Criteria

- Model training – as model gets well adapted to dataset
- RAG retrieval – extracting most relevant chunks or not
- RAG Evaluation scores
- Proper response coming from Stramlit application
- Documentation and demo video.

Required Dataset

- Use: Abirate/english_quotes

Bonus (Optional):

- Try multi-hop queries (e.g., “Quotes tagged with both ‘life’ and ‘love’ by 20th-century authors”).
- Provide download of JSON results.
- Add visualizations of quote/author/tag distributions.

PLEASE NOTE –

For any task you submit, a short video of a code walkthrough and testing is MANDATORY. You can upload the video on Google Drive (and share access) or any of the many sites that are available these days. In case you share the video(s) on Google drive, ensure you give the right kind of access.

For any task you submit, a link to your code or your code itself (Jupyter notebook or Python script with all steps, well- documented) is required. Include modular functions for preprocessing, task extraction, and categorization. Validate results with a small, manually curated sample. Include insights or challenges faced during the task.

If you are unsure of the any instruction or requirement, make an assumption and document it.

It's preferred that everything is uploaded on the Drive and the link of the Drive is shared.

NOTES:

1. **You are not allowed to use any LLM or any other AI tool to generate code for this assignment. However, you can use LLMs (such as Chat GPT to understand the concepts.)**
2. **This assignment is for interview purposes only.**