ACM SIG AI - SEM 4 Recruitment Tasks

Task 1: Neural Network for Image Classification with Preprocessing

Objective:

Develop a deep learning pipeline that preprocesses images and classifies them using a neural network.

Requirements:

Preprocessing, Model Development, Training and Evaluation

Bonus:

- Implement Grad-CAM to visualize which parts of an image influenced the classification.
- Compare CNN performance with a Vision Transformer (ViT) model.

Task 2: Named Entity Recognition (NER) Using Transformers

Objective:

Fine-tune a transformer model for Named Entity Recognition (NER) to identify entities like names, locations, and organizations.

Dataset:

- Use the **CoNLL-2003** dataset or a custom dataset with labeled entities.

Requirements:

Data Preprocessing:

1. Tokenize the text using a transformer-based tokenizer (e.g., BERT tokenizer).
2. Convert labels to a format compatible with the model (e.g., IOB format).

3. Apply padding and truncation to handle different sequence lengths.

Model Development and **Evaluation**

Bonus:

- Build a **Streamlit app** where users can enter text and get highlighted named entities in real time.

Task 3: Chatbot Using Retrieval-Augmented Generation (RAG)

Objective:

Develop a **Retrieval-Augmented Generation (RAG) chatbot** that retrieves relevant documents before generating a response using a large language model (LLM).

Dataset:

- Use **Wikipedia dumps**, **product manuals**, **FAQs**, or any domain-specific knowledge base.

Requirements:

Data Processing & Indexing:

1. Store documents in an efficient retrieval system using **FAISS** or **ChromaDB**.
2. Preprocess documents: remove stopwords, lowercase, tokenize.
3. Convert text into embeddings using **Sentence Transformers** or OpenAI's embeddings API.

Model Development:

1. Implement **retrieval** using FAISS or ChromaDB to fetch relevant passages.
2. Use a **pre-trained LLM** (GPT-3.5, LLaMA, or T5) to generate responses based on retrieved knowledge.
3. Fuse the retrieved information with the generated response using a prompt-engineering strategy.

Evaluation:

1. Use **BERTScore** or **manual evaluation** to assess response quality.
2. Log chatbot interactions and analyze failure cases.

3. Compare performance with and without retrieval (i.e., knowledge-augmented vs. vanilla LLM).

Bonus:

- Add **speech-to-text (STT) and text-to-speech (TTS)** for a voice-based chatbot experience.
- Deploy the chatbot using **Gradio or Streamlit**.

Task 4: AI-Powered Code Assistant for Code Editors

Objective:

Build an AI-powered coding assistant that integrates with a **code editor (VS Code, Jupyter, etc.)**, offering real-time code suggestions and simplifications.

Requirements:

Core Features:

1. Implement an **autocompletion engine** using OpenAI Codex, CodeT5, or StarCoder.
2. Provide **code refactoring suggestions** for improving efficiency.
3. Offer **error detection and auto-fix suggestions**.
4. Summarize code logic in natural language.

Deployment:

1. Package as a **VS Code extension** or a **Jupyter Notebook plugin**.
2. Use **Streamlit or Flask** to build a web-based interactive demo.

Evaluation:

1. Compare AI suggestions with **ground truth implementations**.
2. Measure **latency (response time)** and **user satisfaction (manual review, Likert scale)**.

Bonus:

- Implement **context-aware suggestions** based on previous lines of code.
- Add **multi-language support (Python, Java, JavaScript, C++)**.

Bonus Task: End-to-End Facial Recognition Attendance System

Objective:

Develop a **facial recognition-based attendance system** that detects and logs attendance in real time.

Dataset:

- Use **LFW (Labeled Faces in the Wild)** or collect a **custom dataset**.

Requirements:

Preprocessing:

1. Detect and align faces using **MTCNN or OpenCV**.
2. Extract facial embeddings using **FaceNet or DeepFace**.

Model Development:

1. Train a **Siamese network** or use **pre-trained FaceNet** for recognition.
2. Store facial embeddings in a **database (SQLite, MongoDB, or PostgreSQL)**.
3. Match incoming faces with stored embeddings to mark attendance.

Deployment:

1. Build a **Streamlit/Flask app** with a webcam-based interface.
2. Display real-time logs of attendance.

Evaluation:

1. Use **accuracy, false acceptance rate (FAR), and false rejection rate (FRR)**.
2. Compare recognition time and accuracy against traditional ID-based attendance.

Submission Guidelines

Code:

- Use **Python** for all implementations.
- Organize tasks in **separate folders** (Task1, Task2, etc.).
- Use **Jupyter Notebooks** or **Python scripts**.

Documentation:

- Each task must include a **README.md** explaining:
    - The problem statement
    - Steps taken
    - How to run the code
    - Results summary

Visualizations:

- Include **graphs, confusion matrices, and evaluation reports** where applicable.

GitHub Repository:

- Name the repo: yourname_ACM_SIGAI_Recr_S4.
- Make it **public** and ensure proper **folder structuring**.

Demonstration is Mandatory

Good luck!
LOKESH YARRAMALLU, SIG AI LEAD