



VISHWAKARMA
UNIVERSITY
Maximising Human Potential

Prompt Engineering

Project 2: AI Lab Report Assistant

Name: Pravesh Jain

Roll Number: 18

Division: A

SRN: 31240451

Project Title

AI Lab Report Assistant Using LLMs via OpenRouter API

Abstract

The AI Lab Report Assistant is an intelligent web-based application designed to automate the generation of structured laboratory reports using natural language processing. It utilizes the 'mistralai/mistral-7b-instruct' Large Language Model (LLM) accessed via the OpenRouter API to interpret user prompts and produce formatted lab reports. Users can input the lab topic and optionally specify the aim, tools, and notes. The system then returns a multi-section report with clearly defined segments such as Aim, Tools, Code, Output, and Conclusion. With a responsive user interface, local login simulation, live validation, and export options (PDF, TXT), this tool significantly improves efficiency and consistency in academic reporting.

Problem Statement

Manual writing of lab reports often results in:

- Inconsistencies in structure and formatting
- Time-consuming drafting processes
- Repetitive and error-prone efforts across student submissions

There is a need for an intelligent assistant that:

- Accepts structured and natural language inputs
- Produces logically formatted, context-aware reports
- Supports exportable document formats
- Provides an interactive and secure user experience

Objectives

- Build a full-stack application that generates lab reports using AI.
- Integrate Mistral-7B LLM through OpenRouter for language generation.
- Allow user input of topic, aim, tools, and notes via a frontend interface.
- Return reports in a standardized format ready for submission.
- Enable export to PDF and text formats directly from the browser.

Prompt Documentation

Prompts Used:

Frontend Prompt "Create a basic HTML page with internal CSS and JavaScript that interacts with a Flask API endpoint to submit user input and display the server's response."

Backend Prompt "Write a simple Flask backend in Python that defines an API route (/process) to accept POST requests with JSON data and return a JSON response."

API Integration Prompt "Modify the JavaScript in the HTML page to send an asynchronous fetch request to the Flask backend's /process endpoint and handle the response."

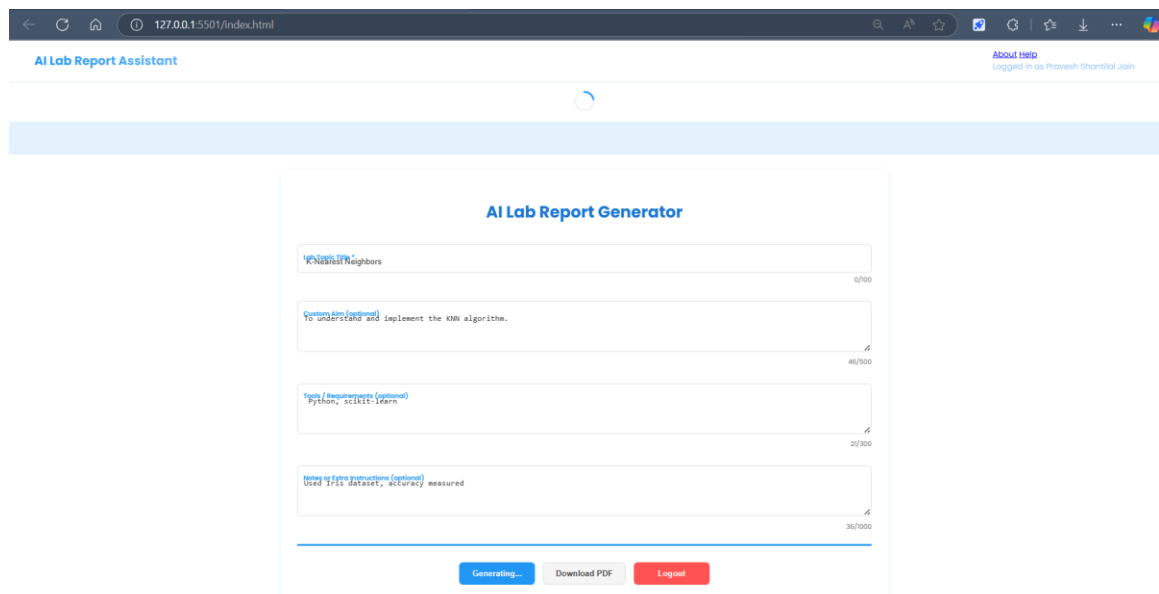
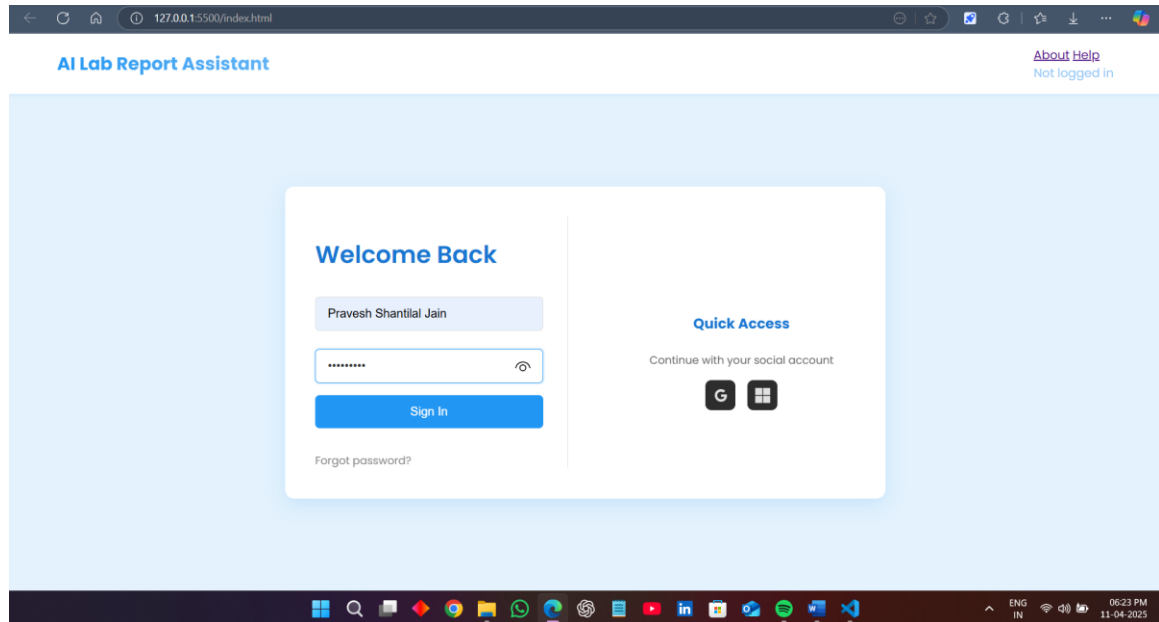
Brief Explanation of Design:

Prompt 1 (Frontend): Focused on creating a self-contained UI using internal CSS/JS, making the project easy to deploy and test without external dependencies. It ensures modularity for small-scale projects or prototypes.

Prompt 2 (Backend): Sets up a minimal Flask API, making it simple for beginners to understand request handling, JSON parsing, and response generation.

Prompt 3 (Integration): This bridges the frontend and backend using AJAX/fetch API, which is essential in any API-based application. It ensures real-time interaction and demonstrates asynchronous request handling clearly.

Sample



36/1000

Report generated successfully! You can now download it as PDF.

Generate Report

Download PDF

Logout

Title: A Comprehensive Study on K-Nearest Neighbors (KNN) Algorithm Using Python and scikit-learn: An Analysis of the Iris Dataset

1. Aim
The objective of this lab was to understand the K-Nearest Neighbors (KNN) algorithm, implement it from scratch, and evaluate its performance using the popular Iris dataset in Python using the scikit-learn library. The accuracy of the KNN model was the primary metric of interest.

2. Tools
- Programming Language: Python
- Libraries: scikit-learn (version 0.24.1)
- Dataset: Iris dataset (available in scikit-learn)

3. Code

```
python
from sklearn import datasets
from sklearn.metrics import classification_error_rate
import math


# Load Iris dataset
iris = datasets.load_iris()
X = iris.data
y = iris.target


# Function to calculate Euclidean distance between two points
def euclidean_distance(a, b):
    return math.sqrt(sum((a - b) ** 2))

# KNN function
def knn(X, y, new_data, k=3):
    dist = []
    predictions = []
    for new_point in new_data:
        # Calculate the distance between the new data point and each data point in X
        for i in range(len(X)):
```


— □ ×


🔍 🔊 ☆

 ⚙️ | ☆ ⬇️ ⋮



Downloads

 🔍 ⋮ 📌

 -k-nearest-neighbors.pdf

[Open file](#)

See more

0/100

Tools and Technologies

Frontend: HTML5, CSS3, JavaScript (vanilla)

Backend: Python with Flask

API Platform: OpenRouter API (Mistral-7B Model)

LLM Model Used: mistralai/mistral-7b-instruct

Authentication: Simulated login (Google/Microsoft UI)

Environment Config: python-dotenv for .env handling

Deployment Port: Localhost (http://127.0.0.1:5000)

PDF Export: jsPDF JavaScript Library

Libraries/Modules:

- Flask – for server-side development
- flask_cors – to enable CORS for frontend-backend communication
- openai – to call LLMs via OpenRouter API
- dotenv – to securely manage environment variables

System Architecture

User Interface → Prompt Construction → Flask Backend → OpenRouter API (LLM) → Report Generation → Display / Export PDF

Module Breakdown

Frontend (index.html + JavaScript)

- **Login Box:** Simulates Google/Microsoft/local login.
- **Form Inputs:**
 - topic (required)

- aim, tools, notes (optional)
- **Client Logic:**
 - Builds prompt dynamically
 - Sends POST request to Flask server
 - Displays report and allows download
 - Shows live character counters and validation messages

Backend (app.py)

- **Routes:**
 - /generate: Handles POST, communicates with OpenRouter, returns AI response
 - /health: Returns server and API key status
- **Error Handling:**
 - Missing prompt
 - API key not configured
 - Invalid or null API response

API (OpenRouter + Mistral-7B)

- Sends structured messages with system and user roles
- Temperature: 0.7
- Token limit: 2000 tokens
- Responds with structured lab report text

Security Considerations

- API key never exposed to the frontend
- CORS enabled only for required local endpoints
- Login is simulated and does not store credentials

- Backend handles all external API communication securely

Sample Prompt

Create a detailed AI lab report for the topic 'Machine Learning with K-Means'.

Aim: Understand the working of K-Means clustering in unsupervised learning.

Tools: Python, Scikit-learn, Jupyter Notebook

Additional Notes: Include dataset loading, clustering code, and plots.

Format the report in sections: Aim, Tools, Code, Output, Conclusion

Key Features

- Natural language interface
- Structured prompt builder
- LLM-based text generation
- Modular frontend/backend architecture
- PDF export using jsPDF
- Responsive design with accessibility support

Sample Output Snippet

Aim: To demonstrate K-Means clustering using Scikit-learn in Python.

Tools: Python 3.x, Scikit-learn, Pandas, Jupyter Notebook

Code: `from sklearn.cluster import KMeans ...`

Output: Clusters visualized with colors, cluster centers marked.

Conclusion: K-Means effectively identifies natural groupings within datasets.

Future Enhancements

- OAuth-based login
- Multi-model selection
- Cloud-based report storage
- Diagram/chart generation
- Admin panel and analytics

Conclusion

The AI Lab Report Assistant is a robust prototype showcasing the capabilities of LLMs in educational automation. With minimal input, it generates complete and structured lab reports in seconds. By integrating Mistral-7B via OpenRouter, the application provides a seamless AI experience with real-world utility in academic contexts. The project is modular, scalable, and positioned well for future enhancements.