Lab 5: Building AI-Integrated Web Applications

Revon Moore

Fall 2025

Contents

1 Client-Side AI with TensorFlow.js

This lab begins with an exploration of client-side machine learning using TensorFlow.js. By running inference directly in the browser, applications can achieve low-latency predictions without relying on a server-side backend. The pre-trained MobileNet model was used to perform image classification tasks in real time.

1.1 Task 1: Image Classification with a Different Image

For Task 1, the original example image was replaced with a different image in order to verify correct model generalization. A new image (a dog) was added to the `img/` directory and referenced in `index.html`. The MobileNet model successfully classified the image and returned a prediction along with a confidence score.

The user interface was customized using CSS to improve readability and presentation. Screenshots were captured to document both the rendered webpage and the model's prediction output, confirming that the classification pipeline functioned as expected.

2 Introduction to Large Language Models with OpenAI

This section explores the use of OpenAI's JavaScript SDK to interact with large language models (LLMs). Environment variables were used to securely store the API key, and the application was executed from the terminal.

2.1 Task 2: Generating Different Model Responses

To complete Task 2, multiple prompts were tested to observe how changes in input affect model output. Both non-streaming and streaming response modes were implemented. While API quota limitations prevented full response generation during execution, terminal logs confirm that the request structure, model configuration, and response handling logic were correctly implemented.

2.2 Task 3: Exploring OpenAI Example Code

The official OpenAI GitHub examples repository was reviewed to better understand common usage patterns and supported modalities. Examples related to text generation and structured

prompts were examined. Relevant source files were inspected and screenshots were collected to demonstrate engagement with the provided examples.

3 LangChain and Retrieval-Augmented Generation

LangChain was introduced to demonstrate advanced prompt orchestration techniques and Retrieval-Augmented Generation (RAG). RAG enhances language model responses by grounding outputs in external documents rather than relying solely on model parameters.

3.1 Task 4: Prompt Engineering Experiments

Multiple prompt variations were tested to observe differences in response structure and content. These experiments demonstrate how prompt design influences model behavior. Results were documented through screenshots and code inspection.

3.2 Task 5: RAG Pipeline Implementation

A Retrieval-Augmented Generation pipeline was implemented using a PDF resume as the source document. The pipeline included document loading, text chunking, embedding generation, and contextual response creation. Terminal output confirms successful initialization of the pipeline and correct execution flow.

4 Task 6: Chatbot User Interface

A web-based chatbot interface was developed to allow users to upload PDF documents for analysis. The application generated two perspectives for each document: a hiring manager review and a critical evaluation. Successful server startup and browser rendering confirm that the interface and backend logic were implemented correctly.

5 Exploring the OpenAI Agents SDK

The OpenAI Agents SDK was explored by implementing a simple Recipe Agent. A custom tool was defined to return ingredients for a specific dish, and an agent was configured to invoke this tool when appropriate. The agent successfully handled structured input and tool execution logic.

Although API quota limits prevented live model responses during execution, the agent architecture, tool definition, and execution flow were fully implemented and verified through code inspection and terminal output.

GitHub Repositories

All source code for this lab is publicly available on GitHub:

- **AI Web Lab (TensorFlow.js, OpenAI API):**
  `https://github.com/revonmoore/RM_ai-web-lab`
- **OpenAI Agents SDK Lab:**
  `https://github.com/revonmoore/agents-lab`

Conclusion

This lab provided practical experience building AI-integrated web applications using TensorFlow.js, OpenAI APIs, LangChain, Retrieval-Augmented Generation, and the OpenAI Agents SDK. Each task reinforced key concepts related to client-side inference, prompt engineering, document-grounded generation, and agent-based architectures. Despite API quota limitations, all required components were implemented, tested, and documented in accordance with the lab requirements.