TUMJA ScienceHackathon

Purpose of This Handout

This document is a technical primer for participants in our upcoming hackathon. You will receive a pre-built React application designed to simulate conflict between AI agents with configurable character traits. The goal of this handout is to help you familiarize yourself with:

- Large Language Models (LLMs)
- AI agents and how they are built
- APIs (specifically OpenAI's API)
- The structure of the React-based app you will receive

This foundational knowledge will enable you to dive straight into modifying or extending the codebase, rather than learning the basics during the hackathon itself.

---

1. Large Language Models (LLMs)

Large Language Models (LLMs) are a type of deep learning model trained on enormous corpora of human-generated text. These models are built using transformer-based architectures, which allow them to capture complex relationships between words, phrases, and contexts over long passages of text. Training an LLM involves exposing it to billions (or even trillions) of tokens from sources such as books, articles, websites, and dialogue transcripts.

Once trained, an LLM can:

- Understand and process natural language input
- Generate fluent, context-aware responses
- Complete text, summarize, translate, or analyze tone

These models are not hardcoded with specific facts. Instead, they learn statistical patterns in language, which allows them to produce human-like responses.

Popular examples of LLMs include:

- **OpenAI GPT-3.5 / GPT-4**: Known for strong reasoning and conversation quality
- **Anthropic Claude**: Focuses on safety-aligned interactions
- **Mistral**: Open-weight models for research and open development
- **Cohere Command R**: Optimized for retrieval-augmented generation

When using an LLM, you interact with it by sending structured prompts. These usually consist of system messages (that set the behavior), user messages (queries or dialogue), and assistant messages (previous replies). The model generates output token-by-token based on the current conversation history.

These interactions power the conversational agents in your project. You'll be building systems that wrap around LLMs to give them memory, personality, goals, and reactivity — turning them into dynamic AI agents capable of conflict simulation and emotional nuance. Examples include:

- **OpenAI GPT-4 / GPT-3.5**
- **Anthropic Claude**
- **Mistral**
- **Cohere Command R**

LLMs operate via text prompts. The model takes in a structured input (e.g., system instructions, user queries) and returns text-based output. This interaction forms the basis of the AI agents you'll be working with.

---

2. What Are AI Agents?

AI agents are software entities powered by LLMs, designed to act based on a set of traits, goals, or rules. In this project, agents:

- Have **configurable personality traits** (e.g., aggressive, calm, defensive)
- Respond to text input using LLM-generated output
- Are given specific "system prompts" that define how they should behave
- Can remember previous parts of the conversation during a session (via chat history management)

These agents don't just chat—they simulate intent, tone, and adaptive behavior.

---

3. Understanding the React Application

You will be provided a ready-to-run React app with the following key features:

- **Agent Configuration Panel**: Set traits for each AI agent before starting a session
- **Scenario Selection**: Choose a conflict context (e.g., workplace disagreement)
- **Conversation Interface**: Agents take turns speaking to simulate realistic interaction
- **User Participation Mode**: Allows a human to replace one of the agents

Technologies used:

- **React**: Frontend framework
- **OpenAI API**: For agent communication

You will extend or adapt this project to explore deeper conflict logic, emotions, or other use cases.

---

4. How APIs Work

An API (Application Programming Interface) allows software systems to communicate with each other. When using an LLM via API:

- You send a **prompt** (text input)
- The API returns a **completion** (text output)
- You can control the behavior using parameters like temperature, max_tokens, and stop

Basic API workflow:

```
fetch("https://api.openai.com/v1/chat/completions", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "Authorization": `Bearer YOUR_API_KEY`
  },
  body: JSON.stringify({
    model: "gpt-4",
    messages: [
      { role: "system", content: "You are a defensive person in a conflict." },
      { role: "user", content: "Why are you always late to meetings?" }
    ]
  })
})
.then(res => res.json())
.then(data => console.log(data.choices[0].message.content));
```

You can define the tone and style of the AI's responses by modifying the system message.

---

5. How OpenAI's Chat API Works

Key components:

- model: Use "gpt-3.5-turbo" or "gpt-4"
- messages: A list of message objects with roles system, user, or assistant
- temperature: Controls randomness (0 = deterministic, 1 = more creative)
- max_tokens: Controls how long the response can be

OpenAI charges per token. A "token" is roughly 4 characters or ¾ of a word. GPT-4 is more expensive but more nuanced.

---

6. What You'll Do During the Hackathon

When you receive the source code, you are expected to:

- Modify prompts and traits to change agent behavior
- Add new conflict scenarios or traits
- Enhance the realism of AI dialogue
- Potentially integrate voice or emotion analysis
- Discuss and mitigate ethical risks (bias, manipulation)
- Solve challenges provided at the event

This is a sandbox for experimentation and creativity.

Tip: You don't need to build everything from scratch. Focus on how you can improve the logic, adapt the interface, or make interactions more human-like.

We hope this primer gives you a head start. Come ready to build, learn, and explore!