# Agent-Based Task Execution System

## Project Overview

This project implements an agent-based system designed to leverage language models and web search APIs to perform complex tasks. The architecture consists of specialized agents that interact and collaborate to gather, analyze, and synthesize information based on a task description provided by the user. The primary objective is to replicate the collaborative approach human experts take when solving multifaceted problems across various domains.

## Features

- \*\*Google Search Integration\*\*: Uses SerpAPI to execute Google searches and retrieve the top 10 relevant links.  
- \*\*YouTube Video Analysis\*\*: Utilizes the YouTube Data API to fetch and present video information related to the user's query, including titles, descriptions, and URLs.  
- \*\*Domain Expert Advice\*\*: Employs OpenAI's GPT-4 model to generate expert insights and advice in legal and medical fields.  
- \*\*User-Friendly Interface\*\*: Developed with Streamlit to provide a seamless and interactive user experience.

## Setup and Installation

### Prerequisites

- Python 3.7 or higher  
- API keys for OpenAI, SerpAPI, and YouTube Data API

### Installation Steps

1. Clone the repository:  
 ```bash  
 git clone https://github.com/your-username/agent-based-task-execution.git  
 cd agent-based-task-execution  
 ```  
2. Install the required packages:  
 ```bash  
 pip install -r requirements.txt  
 ```  
3. Set up your API keys in the `.env` file:  
 - `OPENAI\_API\_KEY`: Your OpenAI API key  
 - `SERPAPI\_KEY`: Your SerpAPI key  
 - `YOUTUBE\_API\_KEY`: Your YouTube API key

## Running the Application

1. Use Streamlit to run the Python script:  
 ```bash  
 streamlit run main.py  
 ```  
2. Enter a task description in the input field and click "Execute Task" to see the results.

## Approach and Implementation

### Agent Design

The system adopts a modular design, where each agent is responsible for executing a specific type of task. This modularity allows for easy expansion and maintenance as more functionality is integrated.  
- \*\*GoogleSearch Agent\*\*: This agent performs web searches using SerpAPI, retrieving the top search results and presenting them to the user.  
- \*\*YouTubeVideo Agent\*\*: Responsible for searching YouTube and fetching details about relevant videos using the YouTube Data API.  
- \*\*LegalAdvisor Agent\*\*: Provides detailed legal advice based on the user's query by generating responses with OpenAI's GPT model.  
- \*\*MedicalAdvisor Agent\*\*: Offers medical insights and advice similarly using OpenAI's GPT model to simulate expert consultations.

### Techniques and Models

The project leverages advanced AI models and external APIs to fulfill its objectives:  
- \*\*OpenAI GPT-4\*\*: Used for its ability to generate coherent, context-aware text that simulates expert advice in various fields, ensuring users receive high-quality insights.  
- \*\*SerpAPI\*\*: Facilitates access to structured Google search results, providing a programmatic means to acquire and analyze top web resources.  
- \*\*YouTube Data API\*\*: Used to search and extract metadata from YouTube, enabling the system to present relevant multimedia content alongside textual information.

### Error Handling

The system includes robust error handling mechanisms to manage API request failures and other runtime issues gracefully. This includes retry logic for API calls, user-friendly error messages, and fallbacks to ensure continuity of service.

## Best Practices

- \*\*Code Organization\*\*: Functions and classes are defined clearly, following industry best practices to enhance readability and maintainability.  
- \*\*Documentation\*\*: In-line comments and documentation are provided to clarify the purpose and functionality of each component.  
- \*\*Scalability\*\*: The modular design ensures that the system can be easily extended with additional agents and features without major refactoring.

## Dependencies

The project requires several dependencies to function correctly:  
- `openai`: For interacting with OpenAI's language model API.  
- `requests`: To facilitate HTTP requests to external services and APIs.  
- `streamlit`: To build an interactive and intuitive user interface.  
- `google-search-results`: For accessing Google search results through SerpAPI.  
- `python-dotenv`: For managing environment variables securely.

## Future Enhancements

The project roadmap includes several enhancements:  
- \*\*Expanding Agent Capabilities\*\*: Adding agents for new domains (such as finance and education) and improving existing ones to handle more complex queries and scenarios.  
- \*\*Enhanced UI/UX\*\*: Integrating more interactive elements in the user interface, such as visualization tools and dynamic feedback systems, to improve usability and engagement.  
- \*\*Performance Optimization\*\*: Fine-tuning the system's performance through optimized API usage, efficient data handling, and faster processing times.  
- \*\*Advanced NLP Features\*\*: Incorporating advanced natural language processing techniques to improve the understanding and contextual relevance of user queries.  
- \*\*Automated Testing\*\*: Implementing automated testing procedures to ensure reliability and stability of the system as new features are added.