Document: Understanding AI Agents and Building Them for Specific Use Cases

---

1. AI Agents Overview

An **AI Agent** is a system designed to perform tasks autonomously or semi-autonomously based on defined objectives. These agents leverage advanced AI technologies, such as natural language processing (NLP) and machine learning, to analyze data, make decisions, and interact with users effectively.

---

2. Difference Between AI Agents and LLMs

- **AI Agents**:
  - Task-specific systems with defined goals and workflows.
  - Leverage multiple tools, APIs, and machine learning models for context and execution.
  - Example: A customer support bot answering queries or processing claims.

- **LLMs (Large Language Models)**:

- Broad-purpose AI models trained on vast datasets to understand and generate text.
- Serve as foundational components for tasks like text generation, summarization, or answering queries.
- Example: OpenAI's GPT-3 or GPT-4.

Key Difference: LLMs provide raw intelligence (understanding language and reasoning), while AI Agents are goal-oriented applications that combine LLMs with workflows and domain-specific knowledge.

---

3. Building AI Agents for Banking, Finance, Security, and Insurance with CrewAI

Step 1: Define the Problem Statement

- Identify the specific task, such as fraud detection, customer service, or portfolio management.
- Ensure the agent's goals align with organizational requirements.

Step 2: Design Tasks and Workflows

- Create subtasks based on the problem. For example, in banking:
  - Task 1: Analyze customer complaints using sentiment analysis.
  - Task 2: Recommend solutions or escalate unresolved cases.

Step 3: Select Tools and Models

- Use tools like **CrewAI** to orchestrate tasks and integrate pre-trained LLMs (e.g., GPT).
- Add APIs and domain-specific datasets for real-time functionality.

Step 4: Build and Test

- Leverage CrewAI to define agents, goals, and tasks.
- Test workflows using sample inputs and iterate to refine accuracy and efficiency.

Step 5: Deploy

- Deploy the agent within applications, ensuring compatibility with existing systems like CRM, cybersecurity platforms, or customer portals.

---

4. Problem Statement and Overview of the Code

Problem Statement

Doctors and healthcare providers often struggle to quickly generate comprehensive diagnosis and treatment plans based on patient symptoms and medical history. This can result in delays, errors, and inconsistent recommendations.

Overview of the Code

The code builds an AI-driven solution that automates the diagnosis and treatment plan process for healthcare providers. By leveraging AI agents (via CrewAI), the application provides:

1. Preliminary diagnoses based on symptoms and medical history.
2. Tailored treatment recommendations that consider best practices and patient-specific needs.

This empowers doctors with quick, accurate, and detailed assistance.

---

5. Process

How the Code Works

1. **Input Collection**:
   - Users provide patient-specific data, including age, gender, symptoms, and medical history.
2. **Agent Tasks**:
   - **Medical Diagnostician Agent**: Processes the input to provide a diagnosis.
   - **Treatment Advisor Agent**: Recommends a treatment plan based on the diagnosis.

3. **Execution**:
   - The agents are orchestrated using CrewAI to execute tasks sequentially.
4. **Output**:
   - The results are displayed in the application and provided as a downloadable Word document.

---

6. Sample Input and Output

Sample Input

- Gender: Male
- Age: 45
- Symptoms: Fever, persistent cough, chest pain
- Medical History: Hypertension, past history of pneumonia

Sample Output

- **Diagnosis**: Possible conditions include pneumonia, bronchitis, or early-stage lung infection.

- **Treatment Plan**: Includes prescribed antibiotics, rest, hydration, follow-up tests, and specialist referral.

---

7. Tools and Libraries Used

1. **Streamlit**: Provides a user-friendly interface for input collection and result display.
2. **CrewAI**: Orchestrates AI agents, defines workflows, and processes tasks.
3. **LangChain OpenAI**: Interacts with GPT models to generate intelligent responses.
4. **SerperDevTool**: Enables search and information retrieval from web sources.
5. **Python Libraries**
   :
   - **docx**: Creates Word documents for downloadable reports.
   - **dotenv**: Manages environment variables for secure API keys.
   - **io**: Handles in-memory file operations for generating documents.

---

8. Code for Reference

```
import streamlit as st
from crewai import Agent, Task, Crew, Process
import os
from crewai_tools import ScrapeWebsiteTool, SerperDevTool
from dotenv import load_dotenv
```

```
from langchain_openai import ChatOpenAI
from docx import Document
from io import BytesIO
import base64


load_dotenv()


# LLM object and API Key
os.environ["OPENAI_API_KEY"] = os.getenv("OPENAI_API_KEY")
os.environ["SERPER_API_KEY"] = os.getenv("SERPER_API_KEY")


def generate_docx(result):
  doc = Document()
  doc.add_heading('Healthcare Diagnosis and Treatment Recommendations', 0)
  doc.add_paragraph(result)
  bio = BytesIO()
  doc.save(bio)
  bio.seek(0)
  return bio


def get_download_link(bio, filename):
  b64 = base64.b64encode(bio.read()).decode()
  return f'<a href="data:application/vnd.openxmlformats-officedocument.wordprocessingml.document;base64,{b64}"
download="{filename}">Download Diagnosis and Treatment Plan</a>'


st.set_page_config(
  layout="wide"
)
```

```
# Title
st.title("AI Agents to Empower Doctors")


# Text Inputs
gender = st.selectbox('Select Gender', ('Male', 'Female', 'Other'))
age = st.number_input('Enter Age', min_value=0, max_value=120, value=25)
symptoms = st.text_area('Enter Symptoms', 'e.g., fever, cough, headache')
medical_history = st.text_area('Enter Medical History', 'e.g., diabetes, hypertension')


# Initialize Tools
search_tool = SerperDevTool()
scrape_tool = ScrapeWebsiteTool()


llm = ChatOpenAI(
  model="gpt-3.5-turbo-16k",
  temperature=0.1,
  max_tokens=8000
)


# Define Agents
diagnostician = Agent(
  role="Medical Diagnostician",
  goal="Analyze patient symptoms and medical history to provide a preliminary diagnosis.",
  backstory="This agent specializes in diagnosing medical conditions based on patient-reported symptoms and medical history. It uses
advanced algorithms and medical knowledge to identify potential health issues.",
  verbose=True,
  allow_delegation=False,
  tools=[search_tool, scrape_tool],
```

```
  llm=llm
)


treatment_advisor = Agent(
  role="Treatment Advisor",
  goal="Recommend appropriate treatment plans based on the diagnosis provided by the Medical Diagnostician.",
  backstory="This agent specializes in creating treatment plans tailored to individual patient needs. It considers the diagnosis, patient history,
and current best practices in medicine to recommend effective treatments.",
  verbose=True,
  allow_delegation=False,
  tools=[search_tool, scrape_tool],
  llm=llm
)


# Define Tasks
diagnose_task = Task(
  description=(
      "1. Analyze the patient's symptoms ({symptoms}) and medical history ({medical_history}).\n"
      "2. Provide a preliminary diagnosis with possible conditions based on the provided information.\n"
      "3. Limit the diagnosis to the most likely conditions."
  ),
  expected_output="A preliminary diagnosis with a list of possible conditions.",
  agent=diagnostician
)


treatment_task = Task(
  description=(
      "1. Based on the diagnosis, recommend appropriate treatment plans step by step.\n"
      "2. Consider the patient's medical history ({medical_history}) and current symptoms ({symptoms}).\n"
      "3. Provide detailed treatment recommendations, including medications, lifestyle changes, and follow-up care."
```

```
  ),
  expected_output="A comprehensive treatment plan tailored to the patient's needs.",
  agent=treatment_advisor
)


# Create Crew
crew = Crew(
  agents=[diagnostician, treatment_advisor],
  tasks=[diagnose_task, treatment_task],
  verbose=2
)


# Execution
if st.button("Get Diagnosis and Treatment Plan"):
  with st.spinner('Generating recommendations...'):
      result = crew.kickoff(inputs={"symptoms": symptoms, "medical_history": medical_history})
      st.write(result)
      docx_file = generate_docx(result)


      download_link = get_download_link(docx_file, "diagnosis_and_treatment_plan.docx")

      st.markdown(download_link, unsafe_allow_html=True)
```