Lesson 3 Demo 7

AI-Powered IT Support with Multi-Agent Interaction

Objective: To demonstrate how multiple AI agents interact, share knowledge, and dynamically delegate tasks to resolve IT issues efficiently

IT support systems often require collaboration across different expertise areas, such as diagnostics, troubleshooting, and escalation. A single agent may not be sufficient to handle complex IT issues effectively. By enabling multi-agent interaction, specialized agents can work together to improve efficiency and resolution time.

The system includes:

- **User agent**: Engages with users and collects issue details
- **Diagnostic agent**: Analyzes the problem and identifies possible causes
- **Resolution agent**: Provides troubleshooting steps and solutions
- **Escalation agent**: Transfers unresolved issues to human support

This structured approach ensures faster issue resolution and a scalable IT support framework.

Prerequisites:

1. Create a virtual environment
2. Install dependencies

Steps to be followed:

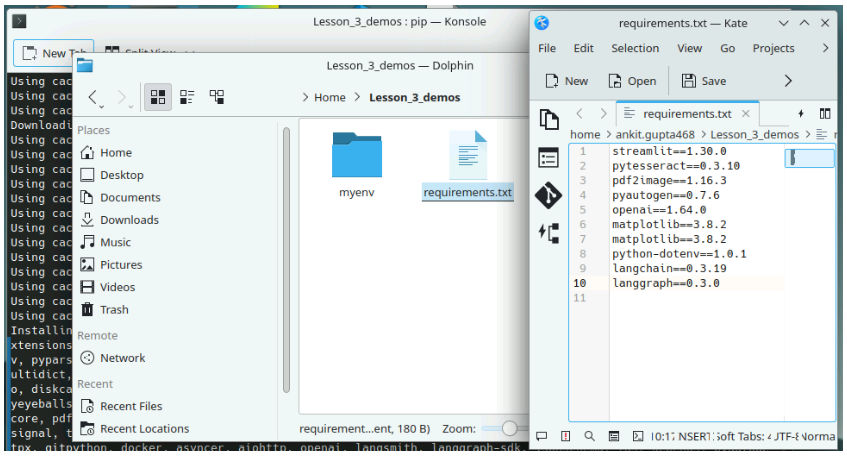
1. Setup the environment
2. Define the Diagnostic Agent
3. Define the Resolution Agent
4. Define the Escalation Agent
5. Build the Streamlit Interface
6. Run the code

Step 1: Set up the environment

1.1 Open command prompt and go to the “**Lesson_3_demos**” folder (which we created in Demo_1) using the command given below:
mkdir Lesson_3_demos (not needed if the folder is already created in Demo1)
cd Lesson_3_demos

1.2 After this, activate the virtual environment using the command below:
python3 -m venv venv (not needed if the virtual env. is already created in Demo1)
source venv/bin/activate

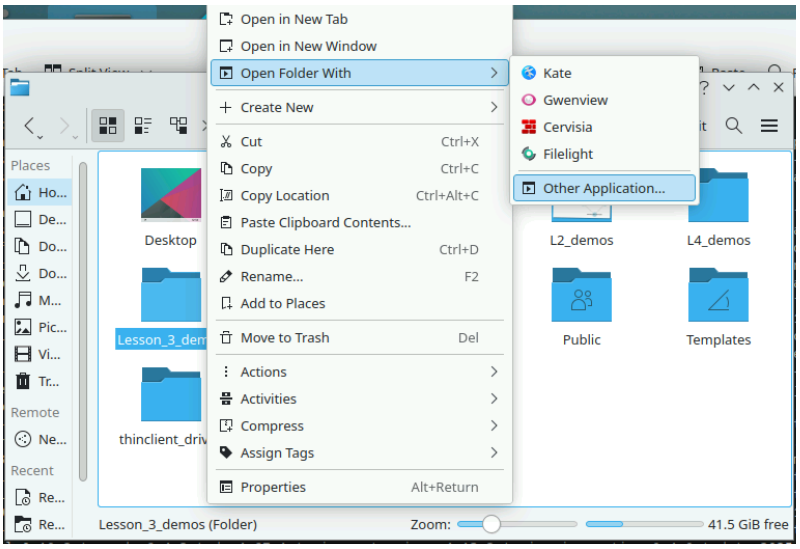
1.3 Now, create a `requirements.txt` file inside the folder with required libraries (not needed if already done in Demo1):



```
streamlit==1.30.0
pytesseract==0.3.10
pdf2image==1.16.3
pyautogen==0.7.6
openai==1.64.0
matplotlib==3.8.2
matplotlib==3.8.2
python-dotenv==1.0.1
langchain==0.3.19
langgraph==0.3.0
```

1.4 Install all the required libraries using the command below:
pip install -r requirements.txt (not needed if already done in Demo1)

1.5 Now, open the folder using VS Code editor:



After this, open a new Python file using the “**New File**” option and name it as “**Demo7**”.

Step 2: Define the Diagnostic Agent

2.1 The **Diagnostic Agent** classifies IT issues into three categories: simple, critical, and complex. Simple issues, such as password resets, slow internet, or software updates, are categorized as "simple.

2.2 Critical issues, such as network failures or security breaches, are classified as "critical. Complex issues, which require advanced troubleshooting, are categorized as "complex."

```
import streamlit as st

import autogen

import openai

from dotenv import load_dotenv

import os


client = openai.AzureOpenAI(


api_key="2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AAABACOGi
3sC",

    api_version="2023-12-01-preview",

    # azure_endpoint="https://openai-api-management-gw.azure-api.net/"

    azure_endpoint="https://openai-api-management-gw.azure-api.net/deployments/gpt-4o-
mini/chat/completions?api-version=2023-12-01-preview"

)


# Define the Diagnostic Agent

class DiagnosticAgent(autogen.AssistantAgent):

    def __init__(self, name="DiagnosticAgent"):

        super().__init__(name=name)


    def analyze_issue(self, issue):

        """Analyzes the issue and decides whether to resolve or escalate."""

        if any(keyword in issue.lower() for keyword in ["password", "slow internet",
"software update"]):

            return "simple"

        elif any(keyword in issue.lower() for keyword in ["network failure", "hardware
crash", "security breach"]):

            return "critical"

        else:
```

Step 3: Define the Resolution Agent

3.1 The **Resolution Agent** addresses complex issues by using GPT to generate troubleshooting steps. For complex issues, the agent provides step-by-step instructions to help the user resolve the problem.

3.2 Simple issues are resolved immediately by offering basic troubleshooting solutions.

```
# Define the Resolution Agent

class ResolutionAgent(autogen.AssistantAgent):

    def __init__(self, name="ResolutionAgent", model="gpt-4o-mini"):

        super().__init__(name=name)

        self.model = model


    def resolve_issue(self, issue):

        """Uses GPT to troubleshoot complex problems."""

        response = client.chat.completions.create(

            model=self.model,

            messages=[

                {"role": "system", "content": "You are an IT support specialist providing solutions to technical problems."},

                {"role": "user", "content": f"Troubleshoot the following IT issue: {issue}"}

            ]

        )

        return response.choices[0].message.content.strip()
```

Step 4: Define the Escalation Agent

4.1 The **Escalation Agent** handles critical issues. When the Diagnostic Agent classifies an issue as critical, the Escalation Agent escalates it to human IT support. The agent informs the user that the issue requires human intervention.

```
# Define the Escalation Agent

class EscalationAgent(autogen.AssistantAgent):

    def __init__(self, name="EscalationAgent"):

        super().__init__(name=name)


    def escalate_issue(self, issue):

        """Escalates the issue to human IT support."""

        return f"The issue '{issue}' requires human intervention. Escalating to IT 
support."
```

Step 5: Build the Streamlit Interface

5.1 The **Streamlit interface** allows users to describe their IT issue and trigger the agent actions.

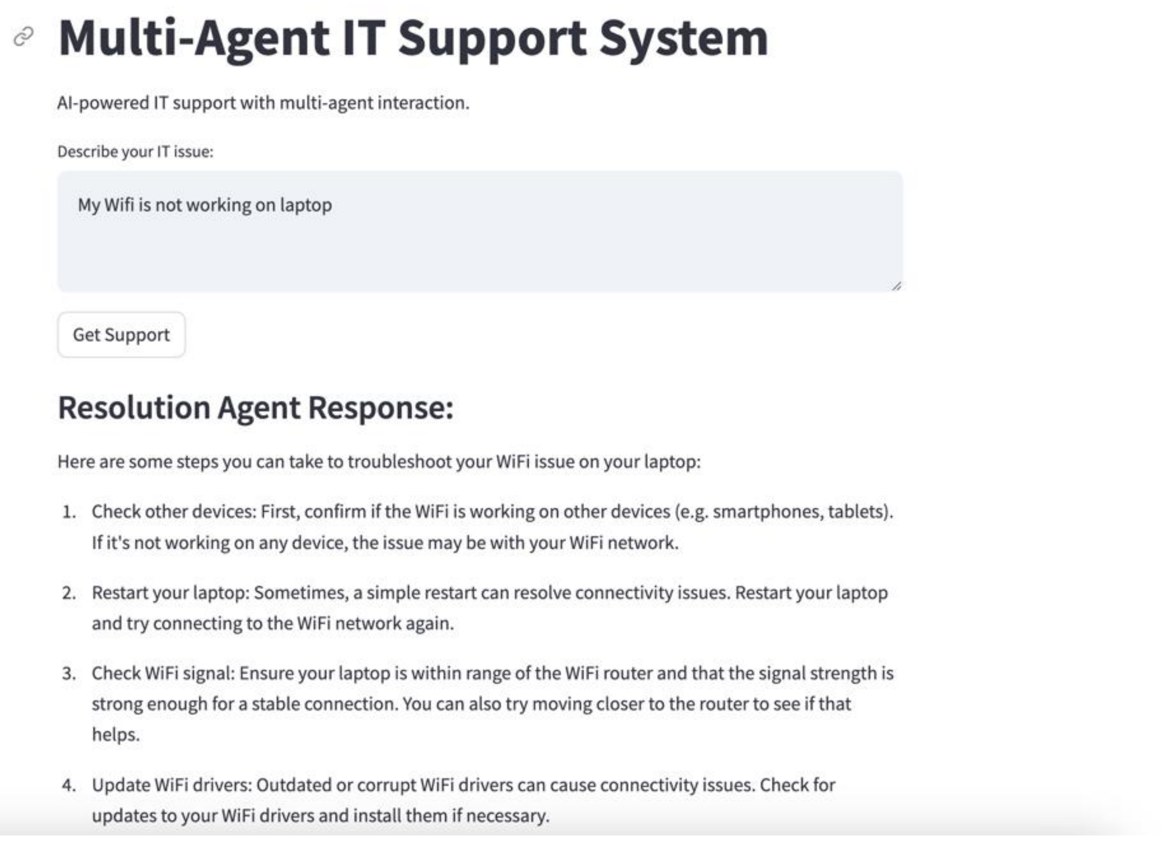
5.2 By clicking "Get Support," the agents classify the issue and take the appropriate actions based on its complexity. Simple issues receive immediate resolutions, complex issues get advanced troubleshooting, and critical issues are escalated.

```python
# Streamlit UI
st.title("Multi-Agent IT Support System")
st.write("AI-powered IT support with multi-agent interaction.")

# User Input
user_issue = st.text_area("Describe your IT issue:")

if st.button("Get Support"):
    if user_issue.strip():
        diagnostic_agent = DiagnosticAgent()
        resolution_agent = ResolutionAgent()
        escalation_agent = EscalationAgent()

        issue_type = diagnostic_agent.analyze_issue(user_issue)

        if issue_type == "simple":
            st.success("The issue has been resolved with basic troubleshooting.")
        elif issue_type == "complex":
            response = resolution_agent.resolve_issue(user_issue)
            st.subheader("Resolution Agent Response:")
            st.write(response)
        else:
            response = escalation_agent.escalate_issue(user_issue)
            st.subheader("Escalation Agent Response:")
            st.write(response)
    else:
        st.warning("Please enter a valid IT issue.")
```

Step 6: Run the code

6.1 Save the file and then run the streamlit webapp from command prompt using the command given below:
streamlit run Demo7.py

Output:



Multi-Agent IT Support System

AI-powered IT support with multi-agent interaction.

Describe your IT issue:

My Wifi is not working on laptop

Get Support

Resolution Agent Response:

Here are some steps you can take to troubleshoot your WiFi issue on your laptop:

1. Check other devices: First, confirm if the WiFi is working on other devices (e.g. smartphones, tablets). If it's not working on any device, the issue may be with your WiFi network.
2. Restart your laptop: Sometimes, a simple restart can resolve connectivity issues. Restart your laptop and try connecting to the WiFi network again.
3. Check WiFi signal: Ensure your laptop is within range of the WiFi router and that the signal strength is strong enough for a stable connection. You can also try moving closer to the router to see if that helps.
4. Update WiFi drivers: Outdated or corrupt WiFi drivers can cause connectivity issues. Check for updates to your WiFi drivers and install them if necessary.

By following the above-mentioned steps, you have successfully demonstrated multi-agent interaction in an IT support system, showcasing how AI agents collaborate, share information, and make decisions collectively. By leveraging specialized agents for different roles, the system improves efficiency, reduces resolution time, and ensures a structured approach to IT troubleshooting.