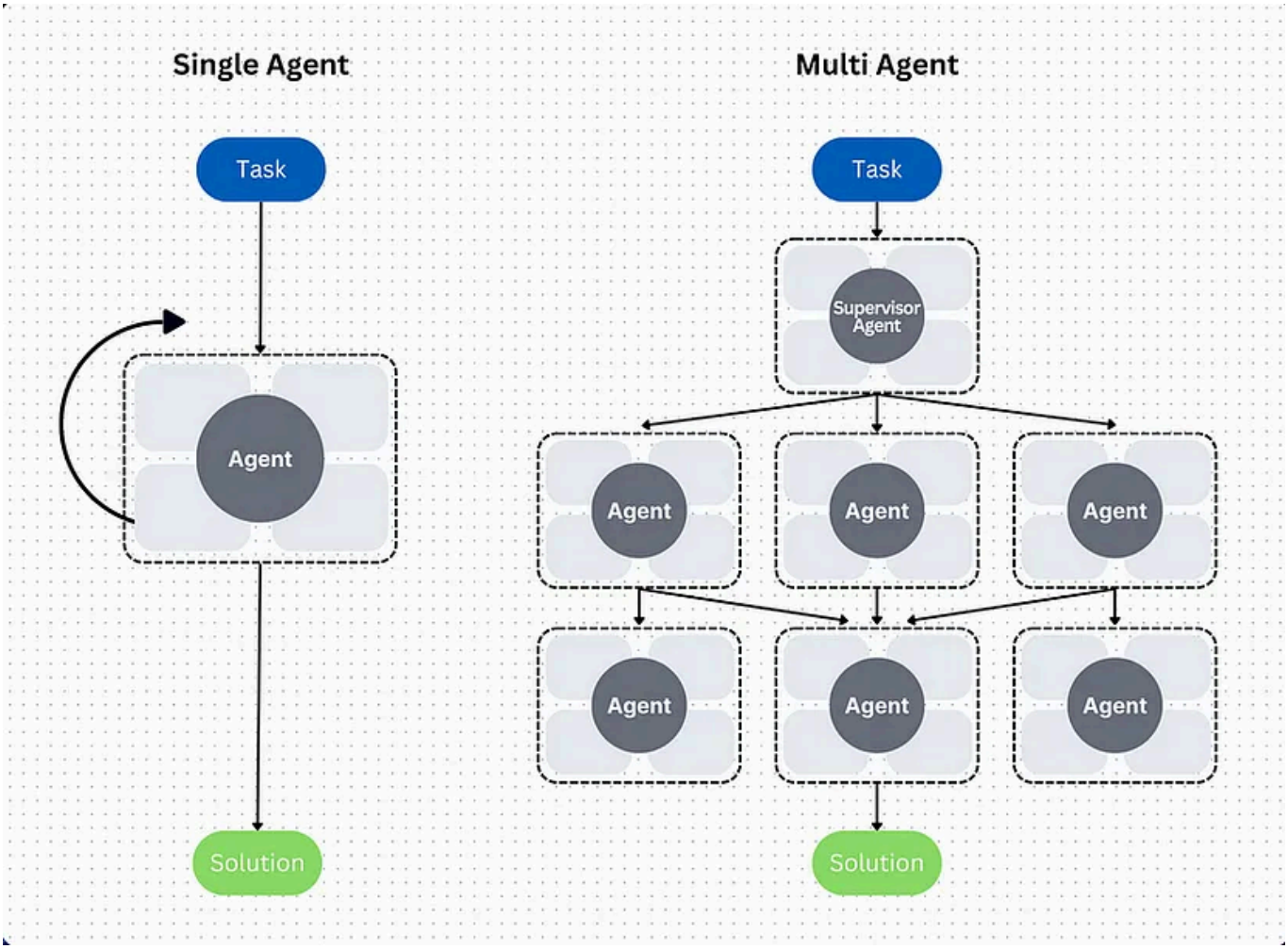
Member-only story

Building Multi-Agent AI Systems From Scratch: OpenAI vs. Ollama

Isuru Lakshan Ekanayaka · Follow

Published in Towards AI · 20 min read · Nov 17, 2024



Source: https://langfuse.com/blog/2024-07-ai-agent-observability-with-langfuse

In the dynamic realm of artificial intelligence, multi-agent systems have emerged as a transformative approach for addressing complex tasks through collaboration and specialization. By distributing responsibilities among distinct agents such as summarizing texts, generating content, and ensuring data privacy these systems enhance efficiency, accuracy, and reliability. This comprehensive guide explores the creation of two robust multi-agent AI systems from scratch using Python: one leveraging OpenAI's GPT-4 model and the other utilizing Ollama's open-source LLaMA 3.2:3b model. Both implementations are designed without relying on existing agent frameworks, offering a foundational understanding for developers eager to master AI agent architectures.

Table of Contents

Introduction

Multi-agent systems in AI involve multiple specialized agents working collaboratively to achieve intricate objectives. By distributing tasks among agents with distinct roles — such as summarizing texts, generating content, and ensuring data privacy — these systems enhance efficiency, accuracy, and reliability. This guide delves into the development of two such systems: one powered by OpenAI's GPT-4 and the other by Ollama's LLaMA 3.2:3b model. Both implementations prioritize transparency and educational value, enabling beginners to grasp the fundamentals of AI agent construction without relying on high-level orchestration frameworks.

OpenAI-Based Multi-Agent System

Source:https://openai.com/

Overview

The **Multi-Agents AI System from Scratch** is a Python-based application that harnesses OpenAI's GPT-4 model to perform specialized tasks through a collaborative multi-agent architecture. Built with Streamlit for an intuitive web interface, this system encompasses agents responsible for summarizing medical texts, writing research articles, and sanitizing medical data (Protected Health Information — PHI). Each primary agent is complemented by a validator agent to ensure output quality and accuracy. Designed with beginners in mind, this project demonstrates that AI agents can be developed without relying on orchestration frameworks like Crew AI, AutoGen, or LangChain.

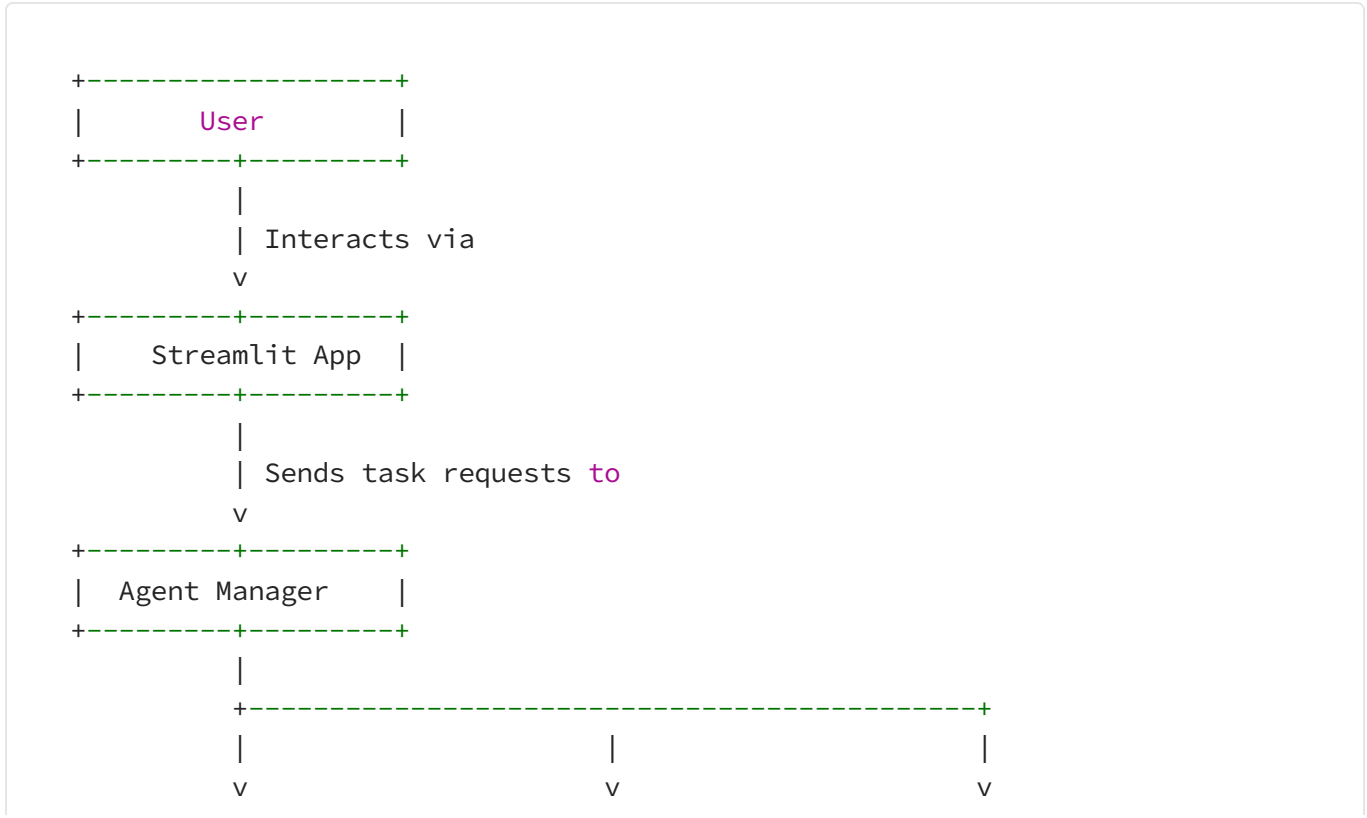
Features

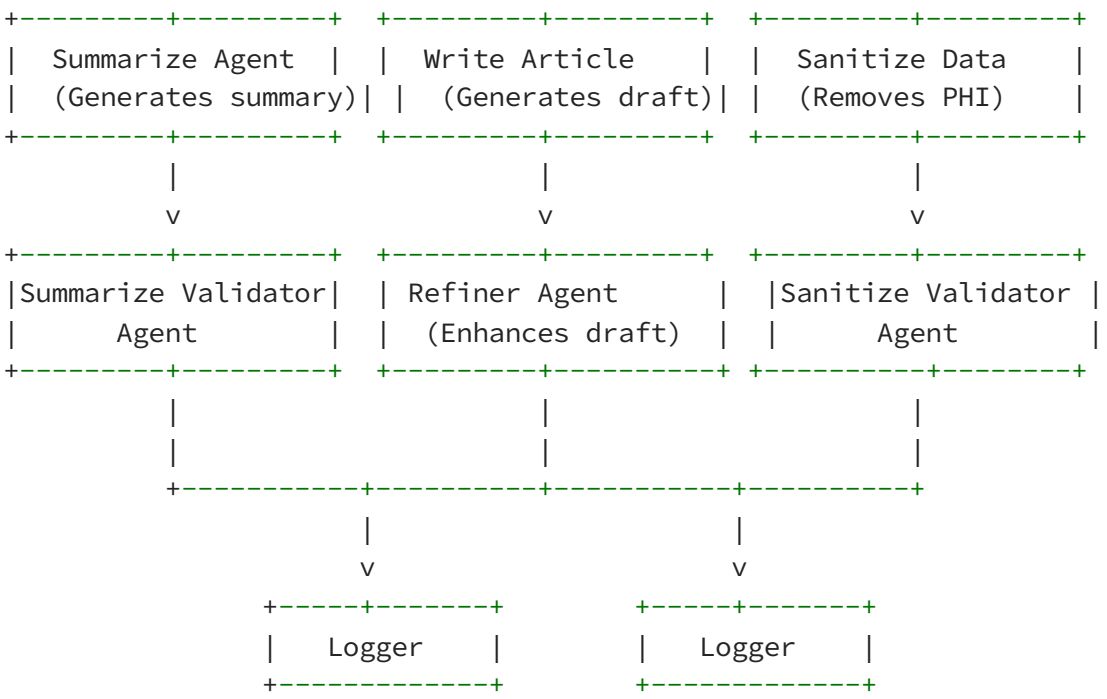
- **Summarize Medical Texts:** Generate concise summaries of extensive medical documents.

- **Write Research Articles:** Create detailed research articles based on a given topic and optional outline.
- **Sanitize Medical Data (PHI):** Remove sensitive health information from medical datasets.
- **Quality Validation:** Each primary task is paired with a validator agent to assess and ensure output quality.
- **Robust Logging:** Comprehensive logging facilitates monitoring and debugging.
- **User-Friendly Interface:** Streamlit-based web app allows easy interaction and task management.

Architecture

The system follows a modular architecture, ensuring clarity and ease of maintenance. Below is a high-level overview:





Project Structure (OpenAI)

```
multi-agent-system/
├── agents/
│   ├── __init__.py
│   ├── agent_base.py
│   ├── refiner_agent.py
│   ├── sanitize_data_tool.py
│   ├── sanitize_data_validator_agent.py
│   ├── summarize_tool.py
│   ├── summarize_validator_agent.py
│   ├── validator_agent.py
│   ├── write_article_tool.py
│   └── write_article_validator_agent.py
├── logs/
│   └── multi_agent_system.log
├── utils/
│   ├── __init__.py
│   └── logger.py
├── .gitignore
├── LICENSE
├── README.md
├── app.py
├── env.example
```

```
├── logo.png
└── requirements.txt
```

Description of Key Components

- **agents/**: Contains various Python scripts for implementing agents, tools, and validators for specific tasks.

> `agent_base.py` *: Base class or interface for different agent types.*
>
> `refiner_agent.py` *: Implements an agent for refining data or tasks.*
>
> `sanitize_data_tool.py` *: A tool for cleaning or sanitizing data.*
>
> `sanitize_data_validator_agent.py` *: Validator agent for ensuring sanitized data meets criteria.*
>
> `summarize_tool.py` *: A tool for summarizing data or content.*
>
> `summarize_validator_agent.py` *: Validator agent for ensuring summaries are correct.*
>
> `validator_agent.py` *: A general agent for validation tasks.*
>
> `write_article_tool.py` *: Tool for generating or writing articles.*
>
> `write_article_validator_agent.py` *: Validator agent for validating written articles.*

- **logs/**: Stores log files for debugging or monitoring.

> *`multi_agent_system.log` : Detailed logs for the system.*

- **utils/**: Contains utility scripts and configuration files.
- `logger.py` : Configures and manages logging using the `loguru` library.
- **.gitignore**: Specifies which files/folders to ignore in version control.
- **LICENSE**: Licensing information for the project.
- **README.md**: Documentation or project overview.
- **app.py**: Entry point for the Streamlit application.
- **env.example**: Example environment configuration file for setting environment variables.
- **logo.png**: Project or company logo.
- **requirements.txt**: Lists dependencies required for the project.

Code Walkthrough (OpenAI)

app.py (OpenAI)

```
import streamlit as st
from agents import AgentManager
from utils.logger import logger
import os
from dotenv import load_dotenv

# Load environment variables from .env if present
load_dotenv()

def main():
    st.set_page_config(page_title="Multi-Agent AI System", layout="wide")
    st.title("Multi-Agent AI System with Collaboration and Validation")

    st.sidebar.title("Select Task")
```

```
    task = st.sidebar.selectbox("Choose a task:", [
        "Summarize Medical Text",
        "Write and Refine Research Article",
        "Sanitize Medical Data (PHI)"
    ])

    agent_manager = AgentManager(max_retries=2, verbose=True)

    if task == "Summarize Medical Text":
        summarize_section(agent_manager)
    elif task == "Write and Refine Research Article":
        write_and_refine_article_section(agent_manager)
    elif task == "Sanitize Medical Data (PHI)":
        sanitize_data_section(agent_manager)

def summarize_section(agent_manager):
    st.header("Summarize Medical Text")
    text = st.text_area("Enter medical text to summarize:", height=200)
    if st.button("Summarize"):
        if text:
            main_agent = agent_manager.get_agent("summarize")
            validator_agent = agent_manager.get_agent("summarize_validator")
            with st.spinner("Summarizing..."):
                try:
                    summary = main_agent.execute(text)
                    st.subheader("Summary:")
                    st.write(summary)
                except Exception as e:
                    st.error(f"Error: {e}")
                    logger.error(f"SummarizeAgent Error: {e}")
                    return

            with st.spinner("Validating summary..."):
                try:
                    validation = validator_agent.execute(original_text=text, sum
                    st.subheader("Validation:")
                    st.write(validation)
                except Exception as e:
                    st.error(f"Validation Error: {e}")
                    logger.error(f"SummarizeValidatorAgent Error: {e}")
        else:
            st.warning("Please enter some text to summarize.")

def write_and_refine_article_section(agent_manager):
    st.header("Write and Refine Research Article")
    topic = st.text_input("Enter the topic for the research article:")
    outline = st.text_area("Enter an outline (optional):", height=150)
    if st.button("Write and Refine Article"):
        if topic:
            writer_agent = agent_manager.get_agent("write_article")
```

```
            refiner_agent = agent_manager.get_agent("refiner")
            validator_agent = agent_manager.get_agent("validator")
            with st.spinner("Writing article..."):
                try:
                    draft = writer_agent.execute(topic, outline)
                    st.subheader("Draft Article:")
                    st.write(draft)
                except Exception as e:
                    st.error(f"Error: {e}")
                    logger.error(f"WriteArticleAgent Error: {e}")
                    return

            with st.spinner("Refining article..."):
                try:
                    refined_article = refiner_agent.execute(draft)
                    st.subheader("Refined Article:")
                    st.write(refined_article)
                except Exception as e:
                    st.error(f"Refinement Error: {e}")
                    logger.error(f"RefinerAgent Error: {e}")
                    return

            with st.spinner("Validating article..."):
                try:
                    validation = validator_agent.execute(topic=topic, article=re
                    st.subheader("Validation:")
                    st.write(validation)
                except Exception as e:
                    st.error(f"Validation Error: {e}")
                    logger.error(f"ValidatorAgent Error: {e}")
        else:
            st.warning("Please enter a topic for the research article.")

def sanitize_data_section(agent_manager):
    st.header("Sanitize Medical Data (PHI)")
    medical_data = st.text_area("Enter medical data to sanitize:", height=200)
    if st.button("Sanitize Data"):
        if medical_data:
            main_agent = agent_manager.get_agent("sanitize_data")
            validator_agent = agent_manager.get_agent("sanitize_data_validator")
            with st.spinner("Sanitizing data..."):
                try:
                    sanitized_data = main_agent.execute(medical_data)
                    st.subheader("Sanitized Data:")
                    st.write(sanitized_data)
                except Exception as e:
                    st.error(f"Error: {e}")
                    logger.error(f"SanitizeDataAgent Error: {e}")
                    return
```

```
            with st.spinner("Validating sanitized data..."):
                try:
                    validation = validator_agent.execute(original_data=medical_d
                    st.subheader("Validation:")
                    st.write(validation)
                except Exception as e:
                    st.error(f"Validation Error: {e}")
                    logger.error(f"SanitizeDataValidatorAgent Error: {e}")
        else:
            st.warning("Please enter medical data to sanitize.")

if __name__ == "__main__":
    main()
```

agents/__init__.py (OpenAI)

```
from .summarize_tool import SummarizeTool
from .write_article_tool import WriteArticleTool
from .sanitize_data_tool import SanitizeDataTool
from .summarize_validator_agent import SummarizeValidatorAgent
from .write_article_validator_agent import WriteArticleValidatorAgent
from .sanitize_data_validator_agent import SanitizeDataValidatorAgent
from .refiner_agent import RefinerAgent  # New import
from .validator_agent import ValidatorAgent  # New import

class AgentManager:
    def __init__(self, max_retries=2, verbose=True):
        self.agents = {
            "summarize": SummarizeTool(max_retries=max_retries, verbose=verbose)
            "write_article": WriteArticleTool(max_retries=max_retries, verbose=v
            "sanitize_data": SanitizeDataTool(max_retries=max_retries, verbose=v
            "summarize_validator": SummarizeValidatorAgent(max_retries=max_retri
            "write_article_validator": WriteArticleValidatorAgent(max_retries=ma
            "sanitize_data_validator": SanitizeDataValidatorAgent(max_retries=ma
            "refiner": RefinerAgent(max_retries=max_retries, verbose=verbose),
            "validator": ValidatorAgent(max_retries=max_retries, verbose=verbose
        }

    def get_agent(self, agent_name):
        agent = self.agents.get(agent_name)
        if not agent:
```

```
            raise ValueError(f"Agent '{agent_name}' not found.")
        return agent
```

agents/agent_base.py (OpenAI)

```
import openai
from abc import ABC, abstractmethod
from loguru import logger
import os
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

openai.api_key = os.getenv("OPENAI_API_KEY")

class AgentBase(ABC):
    def __init__(self, name, max_retries=2, verbose=True):
        self.name = name
        self.max_retries = max_retries
        self.verbose = verbose

    @abstractmethod
    def execute(self, *args, **kwargs):
        pass

    def call_openai(self, messages, temperature=0.7, max_tokens=150):
        retries = 0
        while retries < self.max_retries:
            try:
                if self.verbose:
                    logger.info(f"[{self.name}] Sending messages to OpenAI:")
                    for msg in messages:
                        logger.debug(f"  {msg['role']}: {msg['content']}")
                response = openai.chat.completions.create(
                    model="gpt-4",
                    messages=messages,
                    temperature=temperature,
                    max_tokens=max_tokens,
                )
                reply = response.choices[0].message
                if self.verbose:
                    logger.info(f"[{self.name}] Received response: {reply}")
```

```
                return reply
            except Exception as e:
                retries += 1
                logger.error(f"[{self.name}] Error during OpenAI call: {e}. Retr
        raise Exception(f"[{self.name}] Failed to get response from OpenAI after
```

agents/refiner_agent.py (OpenAI)

```
from .agent_base import AgentBase

class RefinerAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="RefinerAgent", max_retries=max_retries, verbose=v

    def execute(self, draft):
        messages = [
            {
                "role": "system",
                "content": "You are an expert editor who refines and enhances re
            },
            {
                "role": "user",
                "content": (
                    "Please refine the following research article draft to impro
                    f"{draft}\n\nRefined Article:"
                )
            }
        ]
        refined_article = self.call_openai(
            messages=messages,
            temperature=0.5,
            max_tokens=2048
        )
        return refined_article
```

agents/sanitize_data_tool.py (OpenAI)

```
from .agent_base import AgentBase

class SanitizeDataTool(AgentBase):
    def __init__(self, max_retries=3, verbose=True):
        super().__init__(name="SanitizeDataTool", max_retries=max_retries, verbo

    def execute(self, medical_data):
        messages = [
            {"role": "system", "content": "You are an AI assistant that sanitize
            {
                "role": "user",
                "content": (
                    "Remove all PHI from the following data:\n\n"
                    f"{medical_data}\n\nSanitized Data:"
                )
            }
        ]
        sanitized_data = self.call_openai(messages, max_tokens=500)
        return sanitized_data
```

agents/sanitize_data_validator_agent.py (OpenAI)

```
from .agent_base import AgentBase

class SanitizeDataValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="SanitizeDataValidatorAgent", max_retries=max_retr

    def execute(self, original_data, sanitized_data):
        system_message = "You are an AI assistant that validates the sanitizatio
        user_content = (
            "Given the original data and the sanitized data, verify that all PHI
            "List any remaining PHI in the sanitized data and rate the sanitizat
            f"Original Data:\n{original_data}\n\n"
            f"Sanitized Data:\n{sanitized_data}\n\n"
            "Validation:"
        )
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
```

```
        validation = self.call_openai(messages, max_tokens=512)
        return validation
```

agents/summarize_tool.py (OpenAI)

```
from .agent_base import AgentBase

class SummarizeTool(AgentBase):
    def __init__(self, max_retries=3, verbose=True):
        super().__init__(name="SummarizeTool", max_retries=max_retries, verbose=

    def execute(self, text):
        messages = [
            {"role": "system", "content": "You are an AI assistant that summariz
            {
                "role": "user",
                "content": (
                    "Please provide a concise summary of the following medical t
                    f"{text}\n\nSummary:"
                )
            }
        ]
        summary = self.call_openai(messages, max_tokens=300)
        return summary
```

agents/summarize_validator_agent.py (OpenAI)

```
from .agent_base import AgentBase

class SummarizeValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="SummarizeValidatorAgent", max_retries=max_retries

    def execute(self, original_text, summary):
        system_message = "You are an AI assistant that validates summaries of me
        user_content = (
```

```
            "Given the original text and its summary, assess whether the summary
            "Provide a brief analysis and rate the summary on a scale of 1 to 5,
            f"Original Text:\n{original_text}\n\n"
            f"Summary:\n{summary}\n\n"
            "Validation:"
        )
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
        validation = self.call_openai(messages, max_tokens=512)
        return validation
```

agents/validator_agent.py (OpenAI)

```
from .agent_base import AgentBase

class ValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="ValidatorAgent", max_retries=max_retries, verbose

    def execute(self, topic, article):
        messages = [
            {
                "role": "system",
                "content": "You are an AI assistant that validates research arti
            },
            {
                "role": "user",
                "content": (
                    "Given the topic and the research article below, assess whet
                    "Provide a brief analysis and rate the article on a scale of
                    f"Topic: {topic}\n\n"
                    f"Article:\n{article}\n\n"
                    "Validation:"
                )
            }
        ]
        validation = self.call_openai(
            messages=messages,
            temperature=0.3,         # Lower temperature for more deterministic
            max_tokens=500
```

```
        )
        return validation
```

agents/write_article_tool.py (OpenAI)

```
from .agent_base import AgentBase

class WriteArticleTool(AgentBase):
    def __init__(self, max_retries=3, verbose=True):
        super().__init__(name="WriteArticleTool", max_retries=max_retries, verbo

    def execute(self, topic, outline=None):
        system_message = "You are an expert academic writer."
        user_content = f"Write a research article on the following topic:\nTopic
        if outline:
            user_content += f"Outline:\n{outline}\n\n"
        user_content += "Article:\n"
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
        article = self.call_openai(messages, max_tokens=1000)
        return article
```

agents/write_article_validator_agent.py (OpenAI)

```
from .agent_base import AgentBase

class WriteArticleValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="WriteArticleValidatorAgent", max_retries=max_retr

    def execute(self, topic, article):
        system_message = "You are an AI assistant that validates research articl
        user_content = (
            "Given the topic and the article, assess whether the article compreh
```

```
            "Provide a brief analysis and rate the article on a scale of 1 to 5,
            f"Topic: {topic}\n\n"
            f"Article:\n{article}\n\n"
            "Validation:"
        )
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
        validation = self.call_openai(messages, max_tokens=512)
        return validation
```

requirements.txt

```
openai
streamlit
pandas
loguru
python-dotenv
```

Installation (OpenAI)

Prerequisites

- Python 3.8 or higher: Download Python
- OpenAI API Access: Sign up for OpenAI's API

Steps

1. Clone the Repository

```
git clone https://github.com/isurulkh/Multi-Agents-System-from-Scratch.git
```

```
cd Multi-Agents-System-from-Scratch
```

2. Create a Virtual Environment

```
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Install Dependencies

```
pip install -r requirements.txt
```

4.Set Up Environment Variables

Create a `.env` file in the project root:

```
OPENAI_API_KEY=your-api-key-here
```

Alternatively, set the environment variable directly:

- **Unix/MacOS:**

```
export OPENAI_API_KEY='your-api-key-here'
```

- **Windows:**

```
set OPENAI_API_KEY=your-api-key-here
```

Usage (OpenAI)

1. **Activate the Virtual Environment**

```
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

2. Run the Streamlit App

```
streamlit run app.py
```

3. Access the App

- Open the URL provided by Streamlit (usually `http://localhost:8501`) in your web browser.

4. Interact with the Tasks

- **Summarize Medical Text:** Input medical texts to receive concise summaries.
- **Write and Refine Research Article:** Provide a topic and optional outline to generate and refine research articles.
- **Sanitize Medical Data (PHI):** Input medical data to remove sensitive information.

Agents (OpenAI)

Main Agents

Summarize Agent

- **Function:** Generates summaries of provided medical texts.
- **Usage:** Input the text, and receive a concise summary.

Write Article Agent

- **Function:** Creates drafts of research articles based on a topic and optional outline.
- **Usage:** Provide a topic and outline to generate an initial draft.

Sanitize Data Agent

- **Function:** Removes Protected Health Information (PHI) from medical data.
- **Usage:** Input medical data containing PHI to receive sanitized data.

Validator Agents

Summarize Validator Agent

- **Function:** Validates the accuracy and quality of summaries.
- **Usage:** Receives the original text and its summary to assess quality.

Refiner Agent

- **Function:** Enhances and refines research article drafts for better clarity and coherence.
- **Usage:** Receives a draft article and returns an enhanced version.

Sanitize Validator Agent

- **Function:** Ensures that all PHI has been removed from sanitized data.
- **Usage:** Receives original and sanitized data to verify PHI removal.

Logging (OpenAI)

- **Location:** Logs are stored in the `logs/` directory.
- **Files:**
- `multi_agent_system.log` : Contains detailed logs for monitoring and debugging.
- **Configuration:** Logging is handled using the `loguru` library, configured in `utils/logger.py` .

Ollama-Based Multi-Agent System

Source: https://ollama.com/

Overview

The **Ollama-Based Multi-Agent AI App** is a Python-based application leveraging the open-source LLaMA 3.2:3b model via Ollama to perform specialized tasks through a collaborative multi-agent architecture. Built with Streamlit for an intuitive web interface, this system encompasses agents responsible for summarizing medical texts, writing research articles, and sanitizing medical data (Protected Health Information — PHI). Each primary agent is complemented by a validator agent to ensure output quality and accuracy. This implementation emphasizes the use of open-source models, providing flexibility and control over the AI infrastructure.

Features

- **Summarize Medical Texts:** Generate concise summaries of extensive medical documents.
- **Write and Refine Research Articles:** Create detailed research articles based on a given topic and optional outline, followed by refinement for

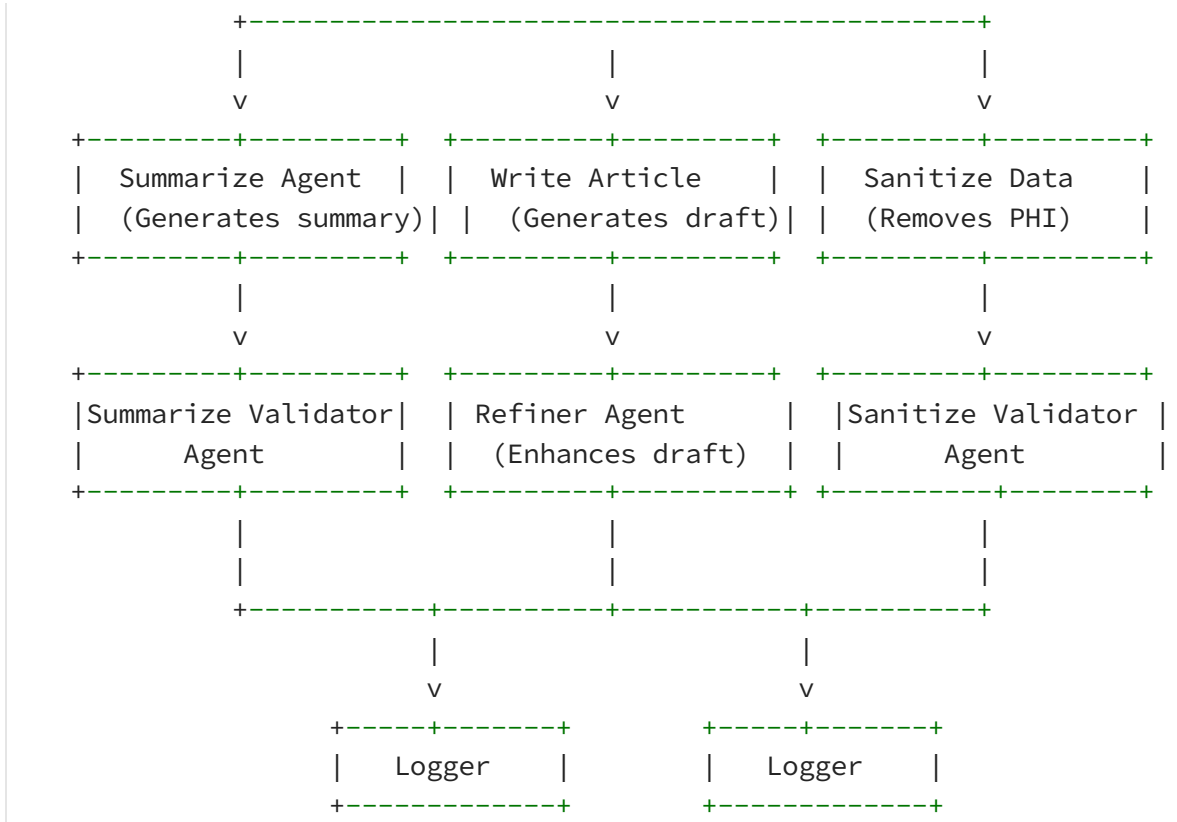
enhanced quality.

- **Sanitize Medical Data (PHI):** Remove sensitive health information from medical datasets to ensure privacy compliance.
- **Quality Validation:** Each primary task is paired with a validator agent to assess and ensure output quality.
- **Robust Logging:** Comprehensive logging facilitates monitoring and debugging.
- **User-Friendly Interface:** Streamlit-based web app allows easy interaction and task management.
- **Open-Source Flexibility:** Utilizes Ollama's LLaMA model for greater control and customization.

Architecture

The architecture mirrors that of the OpenAI-based system, with adjustments to accommodate the Ollama framework and the LLaMA model:

```
+--------------------+
|       User         |
+---------+----------+
          |
          | Interacts via
          v
+---------+----------+
|    Streamlit App   |
+---------+----------+
          |
          | Sends task requests to
          v
+---------+----------+
|  Agent Manager     |
+---------+----------+
          |
```



Project Structure (Ollama)

```
multi-agent-system-ollama/
├── agents/
│   ├── __init__.py
│   ├── agent_base.py
│   ├── refiner_agent.py
│   ├── sanitize_data_tool.py
│   ├── sanitize_data_validator_agent.py
│   ├── summarize_tool.py
│   ├── summarize_validator_agent.py
│   ├── validator_agent.py
│   ├── write_article_tool.py
│   └── write_article_validator_agent.py
├── logs/
│   └── multi_agent_system.log
├── utils/
│   ├── __init__.py
│   └── logger.py
├── .gitignore
├── LICENSE
├── README.md
```

```
├── app.py
├── env.example
├── logo.png
└── requirements.txt
```

Description of Key Components

> ***agents/****: Contains various Python scripts for implementing agents, tools, and validators for specific tasks.*

- `agent_base.py` : Base class or interface for different agent types, modified to interact with Ollama's LLaMA model.
- `refiner_agent.py` : Implements an agent for refining data or tasks.
- `sanitize_data_tool.py` : A tool for cleaning or sanitizing data.
- `sanitize_data_validator_agent.py` : Validator agent for ensuring sanitized data meets criteria.
- `summarize_tool.py` : A tool for summarizing data or content.
- `summarize_validator_agent.py` : Validator agent for ensuring summaries are correct.
- `validator_agent.py` : A general agent for validation tasks.
- `write_article_tool.py` : Tool for generating or writing articles.
- `write_article_validator_agent.py` : Validator agent for validating written articles.

> ***logs/****: Stores log files for debugging or monitoring.*

- `multi_agent_system.log` : Detailed logs for the system.

> ***utils/**: Contains utility scripts and configuration files.*
>
> - `logger.py` : Configures and manages logging using the `loguru` library.
>
> ***.gitignore**: Specifies which files/folders to ignore in version control.*
>
> ***LICENSE**: Licensing information for the project.*
>
> ***README.md**: Documentation or project overview.*
>
> ***app.py**: Entry point for the Streamlit application.*
>
> ***env.example**: Example environment configuration file for setting environment variables.*
>
> ***logo.png**: Project or company logo.*
>
> ***requirements.txt**: Lists dependencies required for the project.*

Code Walkthrough (Ollama)

app.py (Ollama)

```
import streamlit as st
from agents import AgentManager
from utils.logger import logger
import os
from dotenv import load_dotenv

# Load environment variables from .env if present
load_dotenv()
```

```
def main():
    st.set_page_config(page_title="Multi-Agent AI System", layout="wide")
    st.title("Multi-Agent AI System with Collaboration and Validation")

    st.sidebar.title("Select Task")
    task = st.sidebar.selectbox("Choose a task:", [
        "Summarize Medical Text",
        "Write and Refine Research Article",
        "Sanitize Medical Data (PHI)"
    ])

    agent_manager = AgentManager(max_retries=2, verbose=True)

    if task == "Summarize Medical Text":
        summarize_section(agent_manager)
    elif task == "Write and Refine Research Article":
        write_and_refine_article_section(agent_manager)
    elif task == "Sanitize Medical Data (PHI)":
        sanitize_data_section(agent_manager)

def summarize_section(agent_manager):
    st.header("Summarize Medical Text")
    text = st.text_area("Enter medical text to summarize:", height=200)
    if st.button("Summarize"):
        if text:
            main_agent = agent_manager.get_agent("summarize")
            validator_agent = agent_manager.get_agent("summarize_validator")
            with st.spinner("Summarizing..."):
                try:
                    summary = main_agent.execute(text)
                    st.subheader("Summary:")
                    st.write(summary)
                except Exception as e:
                    st.error(f"Error: {e}")
                    logger.error(f"SummarizeAgent Error: {e}")
                    return

            with st.spinner("Validating summary..."):
                try:
                    validation = validator_agent.execute(original_text=text, sum
                    st.subheader("Validation:")
                    st.write(validation)
                except Exception as e:
                    st.error(f"Validation Error: {e}")
                    logger.error(f"SummarizeValidatorAgent Error: {e}")
        else:
            st.warning("Please enter some text to summarize.")

def write_and_refine_article_section(agent_manager):
    st.header("Write and Refine Research Article")
```

```
        topic = st.text_input("Enter the topic for the research article:")
        outline = st.text_area("Enter an outline (optional):", height=150)
        if st.button("Write and Refine Article"):
            if topic:
                writer_agent = agent_manager.get_agent("write_article")
                refiner_agent = agent_manager.get_agent("refiner")
                validator_agent = agent_manager.get_agent("validator")
                with st.spinner("Writing article..."):
                    try:
                        draft = writer_agent.execute(topic, outline)
                        st.subheader("Draft Article:")
                        st.write(draft)
                    except Exception as e:
                        st.error(f"Error: {e}")
                        logger.error(f"WriteArticleAgent Error: {e}")
                        return

                with st.spinner("Refining article..."):
                    try:
                        refined_article = refiner_agent.execute(draft)
                        st.subheader("Refined Article:")
                        st.write(refined_article)
                    except Exception as e:
                        st.error(f"Refinement Error: {e}")
                        logger.error(f"RefinerAgent Error: {e}")
                        return

                with st.spinner("Validating article..."):
                    try:
                        validation = validator_agent.execute(topic=topic, article=re
                        st.subheader("Validation:")
                        st.write(validation)
                    except Exception as e:
                        st.error(f"Validation Error: {e}")
                        logger.error(f"ValidatorAgent Error: {e}")
            else:
                st.warning("Please enter a topic for the research article.")

    def sanitize_data_section(agent_manager):
        st.header("Sanitize Medical Data (PHI)")
        medical_data = st.text_area("Enter medical data to sanitize:", height=200)
        if st.button("Sanitize Data"):
            if medical_data:
                main_agent = agent_manager.get_agent("sanitize_data")
                validator_agent = agent_manager.get_agent("sanitize_data_validator")
                with st.spinner("Sanitizing data..."):
                    try:
                        sanitized_data = main_agent.execute(medical_data)
                        st.subheader("Sanitized Data:")
                        st.write(sanitized_data)
```

```
                except Exception as e:
                    st.error(f"Error: {e}")
                    logger.error(f"SanitizeDataAgent Error: {e}")
                    return

            with st.spinner("Validating sanitized data..."):
                try:
                    validation = validator_agent.execute(original_data=medical_d
                    st.subheader("Validation:")
                    st.write(validation)
                except Exception as e:
                    st.error(f"Validation Error: {e}")
                    logger.error(f"SanitizeDataValidatorAgent Error: {e}")
        else:
            st.warning("Please enter medical data to sanitize.")

if __name__ == "__main__":
    main()
```

agents/__init__.py (Ollama)

```
from .summarize_tool import SummarizeTool
from .write_article_tool import WriteArticleTool
from .sanitize_data_tool import SanitizeDataTool
from .summarize_validator_agent import SummarizeValidatorAgent
from .write_article_validator_agent import WriteArticleValidatorAgent
from .sanitize_data_validator_agent import SanitizeDataValidatorAgent
from .refiner_agent import RefinerAgent  # New import
from .validator_agent import ValidatorAgent  # New import

class AgentManager:
    def __init__(self, max_retries=2, verbose=True):
        self.agents = {
            "summarize": SummarizeTool(max_retries=max_retries, verbose=verbose)
            "write_article": WriteArticleTool(max_retries=max_retries, verbose=v
            "sanitize_data": SanitizeDataTool(max_retries=max_retries, verbose=v
            "summarize_validator": SummarizeValidatorAgent(max_retries=max_retri
            "write_article_validator": WriteArticleValidatorAgent(max_retries=ma
            "sanitize_data_validator": SanitizeDataValidatorAgent(max_retries=ma
            "refiner": RefinerAgent(max_retries=max_retries, verbose=verbose),
            "validator": ValidatorAgent(max_retries=max_retries, verbose=verbose
        }
```

```
    def get_agent(self, agent_name):
        agent = self.agents.get(agent_name)
        if not agent:
            raise ValueError(f"Agent '{agent_name}' not found.")
        return agent
```

agents/agent_base.py (Ollama)

```
import ollama
from abc import ABC, abstractmethod
from loguru import logger
import os

class AgentBase(ABC):
    def __init__(self, name, max_retries=2, verbose=True):
        self.name = name
        self.max_retries = max_retries
        self.verbose = verbose

    @abstractmethod
    def execute(self, *args, **kwargs):
        pass

    def call_llama(self, messages, temperature=0.7, max_tokens=150):
        """
        Calls the Llama model via Ollama and retrieves the response.

        Args:
            messages (list): A list of message dictionaries.
            temperature (float): Sampling temperature.
            max_tokens (int): Maximum number of tokens in the response.

        Returns:
            str: The content of the model's response.
        """
        retries = 0
        while retries < self.max_retries:
            try:
                if self.verbose:
                    logger.info(f"[{self.name}] Sending messages to Ollama:")
                    for msg in messages:
                        logger.debug(f"  {msg['role']}: {msg['content']}")
```

```
                # Call the Ollama chat API
                response = ollama.chat(
                    model='llama3.2:3b',  # Updated model name
                    messages=messages
                )

                # Parse the response to extract the text content
                reply = response['message']['content']

                if self.verbose:
                    logger.info(f"[{self.name}] Received response: {reply}")

                return reply
            except Exception as e:
                retries += 1
                logger.error(f"[{self.name}] Error during Ollama call: {e}. Retr
        raise Exception(f"[{self.name}] Failed to get response from Ollama after
```

agents/refiner_agent.py (Ollama)

```
from .agent_base import AgentBase

class RefinerAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="RefinerAgent", max_retries=max_retries, verbose=v

    def execute(self, draft):
        messages = [
            {
                "role": "system",
                "content": "You are an expert editor who refines and enhances re
            },
            {
                "role": "user",
                "content": (
                    "Please refine the following research article draft to impro
                    f"{draft}\n\nRefined Article:"
                )
            }
        ]
        refined_article = self.call_llama(
            messages=messages,
            temperature=0.5,
```

```
            max_tokens=2048
        )
        return refined_article
```

agents/sanitize_data_tool.py (Ollama)

```
from .agent_base import AgentBase

class SanitizeDataTool(AgentBase):
    def __init__(self, max_retries=3, verbose=True):
        super().__init__(name="SanitizeDataTool", max_retries=max_retries, verbo

    def execute(self, medical_data):
        messages = [
            {"role": "system", "content": "You are an AI assistant that sanitize
            {
                "role": "user",
                "content": (
                    "Remove all PHI from the following data:\n\n"
                    f"{medical_data}\n\nSanitized Data:"
                )
            }
        ]
        sanitized_data = self.call_llama(messages, max_tokens=500)
        return sanitized_data
```

agents/sanitize_data_validator_agent.py (Ollama)

```
from .agent_base import AgentBase

class SanitizeDataValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="SanitizeDataValidatorAgent", max_retries=max_retr

    def execute(self, original_data, sanitized_data):
        system_message = "You are an AI assistant that validates the sanitizatio
```

```
        user_content = (
            "Given the original data and the sanitized data, verify that all PHI
            "List any remaining PHI in the sanitized data and rate the sanitizat
            f"Original Data:\n{original_data}\n\n"
            f"Sanitized Data:\n{sanitized_data}\n\n"
            "Validation:"
        )
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
        validation = self.call_llama(messages, max_tokens=512)
        return validation
```

agents/summarize_tool.py (Ollama)

```
from .agent_base import AgentBase

class SummarizeTool(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="SummarizeTool", max_retries=max_retries, verbose=

    def execute(self, text):
        messages = [
            {"role": "system", "content": "You are an AI assistant that summariz
            {
                "role": "user",
                "content": (
                    "Please provide a concise summary of the following medical t
                    f"{text}\n\nSummary:"
                )
            }
        ]
        summary = self.call_llama(messages, max_tokens=300)
        return summary
```

agents/summarize_validator_agent.py (Ollama)

```
from .agent_base import AgentBase

class SummarizeValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="SummarizeValidatorAgent", max_retries=max_retries

    def execute(self, original_text, summary):
        system_message = "You are an AI assistant that validates summaries of me
        user_content = (
            "Given the original text and its summary, assess whether the summary
            "Provide a brief analysis and rate the summary on a scale of 1 to 5,
            f"Original Text:\n{original_text}\n\n"
            f"Summary:\n{summary}\n\n"
            "Validation:"
        )
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
        validation = self.call_llama(messages, max_tokens=512)
        return validation
```

agents/validator_agent.py (Ollama)

```
from .agent_base import AgentBase

class ValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="ValidatorAgent", max_retries=max_retries, verbose

    def execute(self, topic, article):
        messages = [
            {
                "role": "system",
                "content": "You are an AI assistant that validates research arti
            },
            {
                "role": "user",
                "content": (
                    "Given the topic and the research article below, assess whet
                    "Provide a brief analysis and rate the article on a scale of
```

```
                    f"Topic: {topic}\n\n"
                    f"Article:\n{article}\n\n"
                    "Validation:"
                )
            }
        ]
        validation = self.call_llama(
            messages=messages,
            temperature=0.3,         # Lower temperature for more deterministic
            max_tokens=500
        )
        return validation
```

agents/write_article_tool.py (Ollama)

```
from .agent_base import AgentBase

class WriteArticleTool(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="WriteArticleTool", max_retries=max_retries, verbo

    def execute(self, topic, outline=None):
        system_message = "You are an expert academic writer."
        user_content = f"Write a research article on the following topic:\nTopic
        if outline:
            user_content += f"Outline:\n{outline}\n\n"
        user_content += "Article:\n"
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
        article = self.call_llama(messages, max_tokens=1000)
        return article
```

agents/write_article_validator_agent.py (Ollama)

```
from .agent_base import AgentBase

class WriteArticleValidatorAgent(AgentBase):
    def __init__(self, max_retries=2, verbose=True):
        super().__init__(name="WriteArticleValidatorAgent", max_retries=max_retr

    def execute(self, topic, article):
        system_message = "You are an AI assistant that validates research articl
        user_content = (
            "Given the topic and the article, assess whether the article compreh
            "Provide a brief analysis and rate the article on a scale of 1 to 5,
            f"Topic: {topic}\n\n"
            f"Article:\n{article}\n\n"
            "Validation:"
        )
        messages = [
            {"role": "system", "content": system_message},
            {"role": "user", "content": user_content}
        ]
        validation = self.call_llama(messages, max_tokens=512)
        return validation
```

requirements.txt

```
ollama
streamlit
pandas
loguru
python-dotenv
```

Installation (Ollama)

Prerequisites

- **Python 3.7 or higher:** Download Python
- **Ollama Installed:** Ollama Installation Guide

- **LLaMA 3.2:3b Model:** Ensure the `llama3.2:3b` model is available and correctly configured in Ollama.

Steps

1. Clone the Repository

```
git clone https://github.com/isurulkh/AI-Agents-from-Scratch-using-Ollama.git
cd AI-Agents-from-Scratch-using-Ollama
```

2. Create a Virtual Environment

```
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. Install Dependencies

Ensure the `requirements.txt` file includes all necessary packages:

```
pip install -r requirements.txt
```

4. Set Up Ollama and the LLaMA Model

Install Ollama: Follow the Ollama Installation Guide to install Ollama on your system.

Download and Configure LLaMA 3.2:3b Model:

- Ensure that the `llama3.2:3b` model is downloaded and properly set up in Ollama.
- You can verify the model is available by running the test script or using the Ollama CLI.

Usage (Ollama)

1. **Activate the Virtual Environment**

```
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

2. Run the Streamlit App

```
streamlit run app.py
```

3. Access the App

- Open the URL provided by Streamlit (usually `http://localhost:8501`) in your web browser.

4. Interact with the Tasks

- **Summarize Medical Text:** Input medical texts to receive concise summaries.
- **Write and Refine Research Article:** Provide a topic and optional outline to generate and refine research articles.
- **Sanitize Medical Data (PHI):** Input medical data to remove sensitive information.

Agents (Ollama)

Main Agents

Summarize Agent

- **Function:** Generates summaries of provided medical texts.
- **Usage:** Input the text, and receive a concise summary.

Write Article Agent

- **Function:** Creates drafts of research articles based on a topic and optional outline.
- **Usage:** Provide a topic and outline to generate an initial draft.

Sanitize Data Agent

- **Function:** Removes Protected Health Information (PHI) from medical data.
- **Usage:** Input medical data containing PHI to receive sanitized data.

Validator Agents

Summarize Validator Agent

- **Function:** Validates the accuracy and quality of summaries.
- **Usage:** Receives the original text and its summary to assess quality.

Refiner Agent

- **Function:** Enhances and refines research article drafts for better clarity and coherence.
- **Usage:** Receives a draft article and returns an enhanced version.

Sanitize Validator Agent

- **Function:** Ensures that all PHI has been removed from sanitized data.
- **Usage:** Receives original and sanitized data to verify PHI removal.

Logging (Ollama)

- **Location:** Logs are stored in the `logs/` directory.

Files:

- `multi_agent_system.log` : Contains detailed logs for monitoring and debugging.
- **Configuration:** Logging is handled using the `loguru` library, configured in `utils/logger.py` .

Acknowledgements (Ollama)

- Ollama for providing the platform to run LLaMA models locally.
- LLaMA by Meta for the powerful open-source language model.
- Streamlit for the web application framework.
- Loguru for the logging library.
- Inspired by collaborative multi-agent system architectures and prompt engineering techniques like Chain-of-Thought (CoT) and ReAct.

Building multi-agent AI systems from scratch offers invaluable insights into the mechanics of AI-driven collaboration and task specialization. By developing these systems without relying on high-level frameworks, developers gain a deeper understanding of agent interactions, prompt engineering, and system orchestration. Whether leveraging proprietary models like OpenAI's GPT-4 or open-source alternatives like Ollama's LLaMA, the foundational principles remain consistent, emphasizing modularity, validation, and robust logging.

These systems not only enhance task efficiency and accuracy but also pave the way for scalable and adaptable AI solutions tailored to specific domains such as healthcare, research, and data management. By mastering the creation of multi-agent architectures, developers can contribute to more sophisticated and reliable AI applications, driving innovation across various industries.

OpenAI Ollama AI Artificial Intelligence Multi Agent Systems

Published in Towards AI

62K Followers · Last published 2 hours ago

Follow

The leading AI community and content platform focused on making AI accessible to all

Written by Isuru Lakshan Ekanayaka

118 Followers · 11 Following

Follow

AI R&D Engineer| Machine Learning Engineer | MLOps | GenerativeAI Engineer | Deep Learning | LLMOps | Data Scientist | Research & Development | NLP

More from Isuru Lakshan Ekanayaka and Towards AI

In Towards AI by Isuru Lakshan Ekanayaka

25 Must-Know Retrieval-Augmented Generation Models...

In Towards AI by Mohit Sewak, Ph.D.

LLM Agent Jailbreaking and Defense—101

Introduction

Nov 12 32

The Complete Guide to LLM Agent Security: Ways to Secure Your GenAI Agents

1d ago 213 1

In Towards AI by Surya Maddula

Feedback Loop Mechanisms in Retrieval Systems

Overview: Examine systems that learn from user interactions to enhance future retrieval...

5d ago 256 2

In Towards AI by Isuru Lakshan Ekanayaka

RAG Dictionary: Your A-to-Z Guide to Mastering Retrieval-Augmente...

In the rapidly evolving landscape of artificial intelligence, Retrieval-Augmented Generati...

Nov 20 44

See all from Isuru Lakshan Ekanayaka

See all from Towards AI

Recommended from Medium

In Level Up Coding by Md Monsur ali

OmniVision-968M: The World's Most Compact and Smallest...

How OmniVision-968M Outperforms Existing Vision-Language Models and Enables...

Nov 17 370 4

Alfredo Sone

AI Agents: How to build Digital Workers

Key learnings to understand and design intelligent digital workers.

Nov 18 368 9

Lists

AI Regulation

6 stories · 633 saves

Generative AI Recommended Reading

52 stories · 1526 saves

Natural Language Processing

1839 stories · 1461 saves

What is ChatGPT?

9 stories · 474 saves

In Stackademic by Crafting-Code

In Data Science in your pocket by Mehul Gupta

I Stopped Using Kubernetes. Our DevOps Team Is Happier Than Ever

Why Letting Go of Kubernetes Worked for Us

Nov 19 2.9K 94

Magentic-One, AutoGen, LangGraph, CrewAI, or OpenAI...

Pros and Cons of popular Multi-Agent Orchestration framework

Nov 13 465 11

In Towards Data Science by Dr. Leon Eversberg

Improved RAG Document Processing With Markdown

How to read and convert PDFs to Markdown for better RAG results with LLMs

Nov 19 596 4

In Cubed (we have moved to differ.... by Michael ...

The Insanity of Relying on Vector Embeddings: Why RAG Fails

In RAG, the goal is to locate the stored information that has the highest percentage...

Nov 21 828 21

See more recommendations