Table of Contents

1. Overview – Multi-Agent System 2
2. Layered Architecture 2
   2.1 User Interface Layer 3
      2.1.1 Purpose 3
      2.1.2 Components 3
   2.2 Agent Layer 4
      2.3.1 Purpose 4
      2.3.2 Components 4
   2.3 MCP Layer 5
      2.4.1 Purpose 5
      2.4.2 Components 5
3. Directory Structure 6
4. Execution and Results Interpretation 8
   4.1 Overview 8
   4.2 Starting the System 8
      Step 1: Launch all background processes 8
      Step 2: Running the Supervisor (Main Application) 9
      Step 3: Stopping the System 9
      Step 4: Monitoring Execution Logs 9
   4.6 Outputs and Results 10
   4.7 Interpreting a Successful Run 11
5. Summary 12

1. Overview – Multi-Agent System

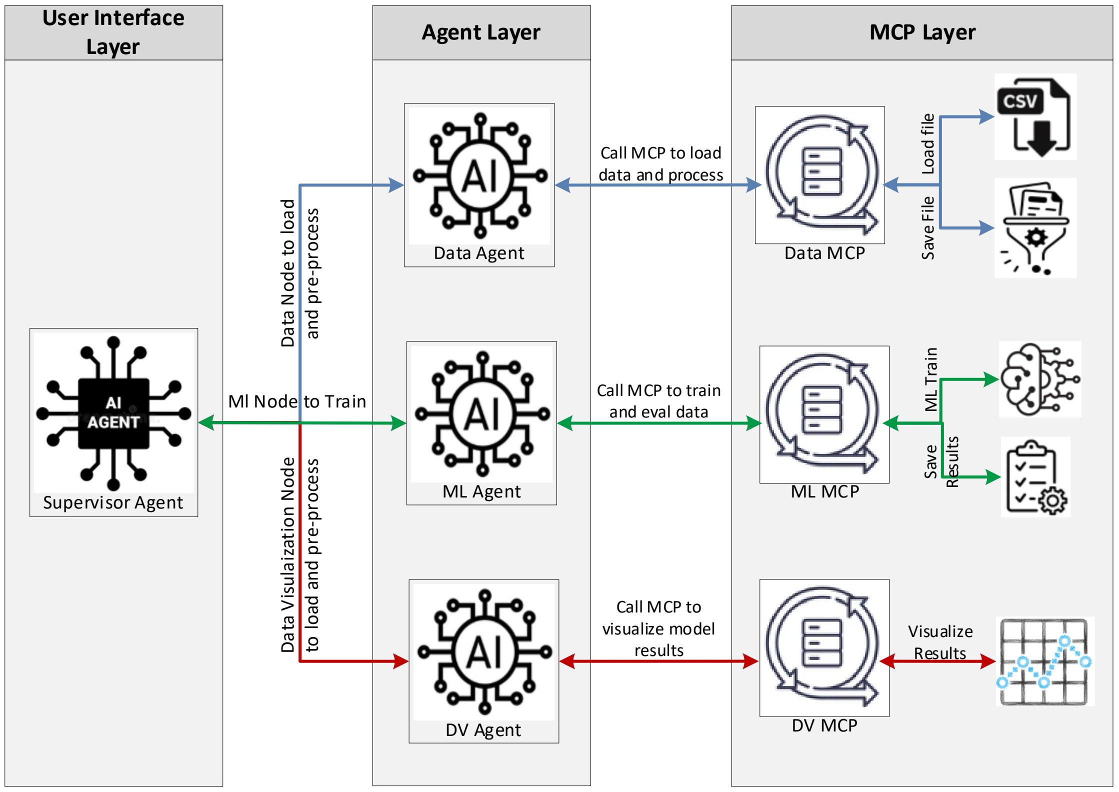
This **Multi-Agent System (MAS)** is a **layered, modular, and distributed architecture** designed to automate the end-to-end pipeline of **data ingestion, pre- processing, machine learning (ML) and visualization workflows.**

The system is composed of multiple **specialized agents** — each responsible for a distinct functional domain such as **data management, ML modeling, and data visualization**. These agents operate **independently** but are **orchestrated by a central Supervisor Agent**, which ensures workflow sequencing, communication, and health monitoring across all agents.

Structured Communication and co-ordination between agents are done through LangGraph defined in supervisor agent. Each agent can run on its own MCP port or node, enabling **horizontal scaling** and **easy extensibility** — new agents or capabilities can be plugged into the ecosystem without disrupting existing operations.

2. Layered Architecture

The **Multi-Agent System** is organized into a set of **logical layers**, each serving a specific responsibility while maintaining loose coupling and clear boundaries. This design ensures flexibility, maintainability, and scalable deployment across environments.



2.1 User Interface Layer

The **User Interface Layer** serves as the **entry point** of the system — enabling users or external systems to trigger and interact with the multi-agent workflow.

2.1.1 Purpose

To coordinate and execute multiple agents following a **defined set of rules and sequences** using **LangGraph**. This layer abstracts the internal agent communication and provides a unified interface to initiate and monitor process execution.

2.1.2 Components

Below is the list of components in User Interface Layer:

- **Entry Point**
  Acts as the exposed interface (e.g., command-line entry, API endpoint, or UI trigger) for external users or systems to start the entire process execution.

- **Agent Cards**
  Define metadata and operational details for each agent — including its purpose, capabilities, and configuration parameters — used to dynamically instantiate agents during runtime.

- **Agent URI Registry**
  A centralized list of all agent endpoints (URIs) mapped to their corresponding Agent Cards. This enables the system to discover and route tasks to the appropriate agent nodes.

- **Execution Graph (LangGraph)**
  Represents the workflow structure as a directed graph, where **nodes** correspond to agents and **edges** define execution conditions or dependencies.
  Once triggered, LangGraph orchestrates agent execution based on this graph logic, ensuring a controlled and rule-based flow.

2.2 Agent Layer

The **Agent Layer** contains the functional building blocks of the system — the specialized agents that perform domain-specific or task-specific operations such as data processing, machine learning, or visualization.

2.3.1 Purpose

To execute independent, modular tasks defined by the Supervisor Layer. Each agent is responsible for a specific function and can operate autonomously, exchanging results or intermediate outputs with other agents as required.

2.3.2 Components

Below is the list of components in the Agent Layer:

- **Data Agent**
  Responsible for data ingestion, preprocessing, and transformation. It connects to data sources; cleanses input and prepares datasets for downstream tasks.

- **ML Agent (Modeling Agent)**
  Handles model training, inference, and evaluation. It utilizes machine learning frameworks to perform predictive or analytical computations and returns structured results to the Supervisor.

- **DV Agent (Data Visualization Agent)**
  Generates analytical reports, charts, or dashboards using processed data or model outputs. It enables insight generation for end-users through automated visualization.

- **Communication Interface**
  Defines standardized methods (e.g., REST API, message queues, MCP calls) for communication between agents and the Supervisor Layer. This ensures consistent, reliable, and asynchronous data exchange

2.3 MCP Layer

The **MCP (Model Context Protocol) Layer** serves as the communication and context management backbone of the multi-agent system. It enables seamless interaction between agents, allowing them to exchange data, models, and metadata during different stages of execution — from data ingestion to visualization.

2.4.1 Purpose

To facilitate standardized communication, resource sharing, and context synchronization between agents involved in the end-to-end workflow. The MCP Layer ensures that each agent receives the right input, maintains contextual consistency, and produces outputs that can be consumed by the next agent in the defined sequence.

2.4.2 Components

Below is the list of components in the MCP Layer:

- **MCP Server**
  Act as point of connection for all communications to external resources. For data-agent it connected to external dataset and preprocess. For ml_agent it connects to external model and train the dataset. For data visualization layer it plots graphs for the ML outputs.

- **MCP Client Connectors**
  Lightweight interfaces embedded in each agent to connect with the MCP Server. They handle authentication, message exchange, and data transfer, ensuring agents remain loosely coupled but context-aware.

3. Directory Structure

3.1 Supervisor

```
assignment_level1/
├── code/
│   ├── supervisor_agent/           # Supervisor Layer (Central Orchestration)
│   │   ├── agent_main.py           # Entry point for workflow coordination
│   │   └── supervisor_agent.py     # Defines workflow orchestration logic
```

3.2 Agents

```
assignment_level1/
├── code/
│   ├── agents/                     # Agent Layer (Core Functional Agents)
│   │   ├── data_agent/             # Agent 1: Data Agent (Load & Preprocess)
│   │   │   ├── agent_executor.py
│   │   │   ├── agent_main.py
│   │   │   └── data_agent.py
│   │   ├── ml_agent/               # Agent 2: ML Agent (Train & Evaluate)
│   │   │   ├── agent_executor.py
│   │   │   ├── agent_main.py
│   │   │   └── ml_agent.py
│   │   └── dv_agent/               # Agent 3: DV Agent (Visualize)
│   │       ├── agent_executor.py
│   │       ├── agent_main.py
│   │       └── dv_agent.py
```

3.3 Clients

```
assignment_level1/
├── code/
│   ├── client/                     # User Interface Layer (Entry Points)
│   │   ├── data_client.py          # Triggers Data Agent workflow
│   │   ├── ml_client.py            # Triggers ML Agent workflow
│   │   └── dv_client.py            # Triggers DV Agent workflow
```

3.4 MCP Servers

```
assignment_level1/
├── code/
│   ├── mcp_servers/                # MCP Layer (Context & Communication)
│   │   ├── mcp_data.py             # MCP server for Data Agent context
│   │   ├── mcp_ml.py               # MCP server for ML Agent context
│   │   └── mcp_dv.py               # MCP server for DV Agent context
```

3.5 Artifacts

```
│   ├── models/                     # Saved Models
│   │   └── random_forest.pkl
│   │
│   ├── artifacts/                  # Output Artifacts (Results & Visuals)
│   │   ├── data_results/           # Output from Data Agent
│   │   │   └── processed_data.csv
│   │   ├── ml_results/             # Output from ML Agent
│   │   │   ├── metrics.json
│   │   │   └── predictions.csv
│   │   └── dv_results/             # Output from DV Agent
│   │       ├── pred_vs_actual_pred_vs_actual.png
│   │       ├── pred_vs_actual_residuals.png
│   │       └── pred_vs_actual_summary.json
│   │
│   ├── scripts/                    # Infrastructure Layer (Execution Utilities)
│   │   ├── start_all.bat           # Starts all agents and MCP servers
│   │   ├── stop_all.bat            # Stops all running agents and servers
│   │   └── logs/                   # Execution Logs
│   │       ├── data_agent.log
│   │       ├── ml_agent.log
│   │       ├── dv_agent.log
│   │       ├── mcp_data.log
│   │       ├── mcp_ml.log
│   │       └── mcp_dv.log
│   │
│   └── README.md                   # Project overview and usage instructions
│
└── documents/                      # (Optional) Reports, diagrams, and documentation
    └── architecture_diagram.png
```

4. Execution and Results Interpretation

4.1 Overview

The end-to-end system is orchestrated through a hierarchical control flow consisting of a **Supervisor Agent**, three domain-specific **sub-agents** (Data, DV, and ML Agents), and supporting **Model Control Point (MCP)** servers. The Supervisor acts as the command center, routing structured tasks to the agents via the **A2A (Agent-to-Agent)** communication protocol. Each agent, in turn, interacts with its respective MCP for data retrieval, computation, and persistence.

This section details the exact procedure to **start, execute, and monitor** the system, followed by instructions to interpret outputs and verify successful execution.

4.2 Starting the System

All agents and MCP servers are initiated through a single automation script placed under the scripts directory.

Step 1: Launch all background processes

Navigate to the scripts folder inside the main project directory and execute:

start_all.bat

This script sequentially starts:

- The **Data MCP Server** (handles data I/O and serialization)
- The **DV MCP Server** (handles visualization computations)
- The **ML MCP Server** (handles model training, prediction, and metrics)
- The **Data Agent**
- The **DV Agent**
- The **ML Agent**
- The **Supervisor Agent (Controller)**

Each component runs as an independent Python process bound to a predefined local port (e.g., 10100–10300). Logs for every service are automatically written to the scripts/logs/ folder.

Step 2: Running the Supervisor (Main Application)

Once the environment is initialized, open a new terminal in the project's root code directory and execute:

```
python -m supervisor_agent.agent_main
```

This command launches the **Supervisor Agent**, which performs the following sequence:

1. **Establish communication** with each active agent via the A2A interface.
2. **Dispatches the data preparation request** to the Data Agent.
3. Waits for completion and validation of processed dataset artifacts.
4. **Triggers the DV Agent** to perform visual diagnostics and create summary plots.
5. **Invokes the ML Agent** to train a model, generate predictions, and compute metrics.
6. Aggregates all structured results and stores them in the respective artifacts subfolders.

At runtime, the terminal and log outputs display detailed status messages, including execution checkpoints, intermediate data summaries, and structured JSON messages returned by each agent.

Step 3: Stopping the System

To safely terminate all background services (agents and MCP servers), execute the following command from the same scripts directory:

```
stop_all.bat
```

This ensures graceful shutdown of each running process and prevents port locking or zombie processes.

Step 4: Monitoring Execution Logs

Each service maintains its own log file for traceability and debugging. The log files are automatically created and stored under:

```
scripts/logs/
```

Key log files include:

data_agent.log	records data loading, preprocessing, and validation stages
dv_agent.log	records visualization and plots processing and exceptions
ml_agent.log	records model training details, processing and exceptions
supervisor_agent.log	Records supervisor execution and exceptions

Logs are timestamped, formatted, and display success or failure messages for each operation. Errors (if any) can be traced using stack traces within these files.

4.6 Outputs and Results

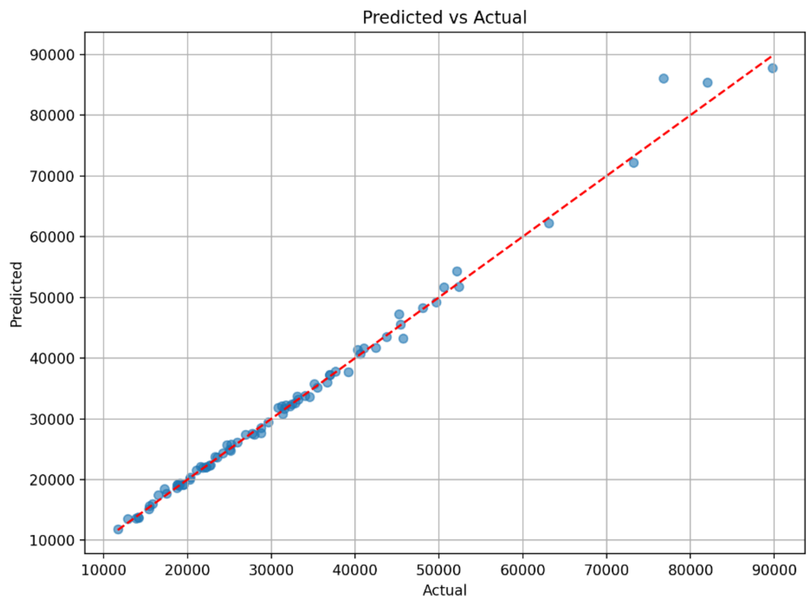
Upon successful execution, the pipeline generates multiple artifacts across the following locations:

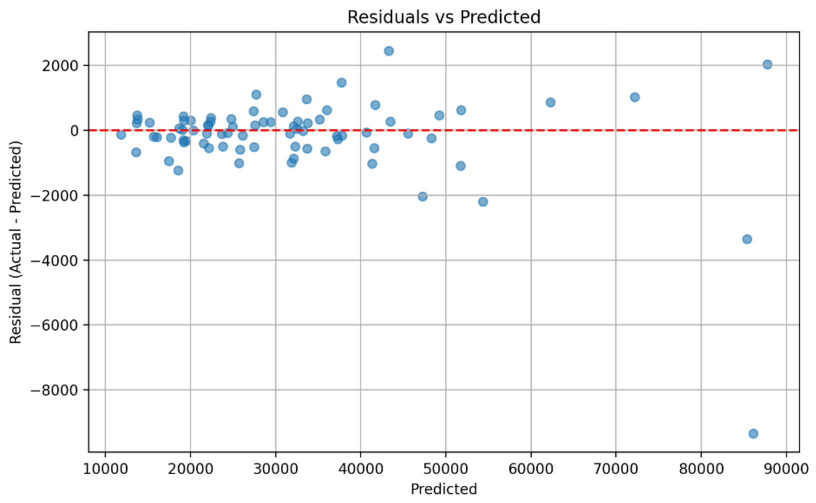
Component	Directory	Description
Data Agent	artifacts/data_results	Cleaned, feature-engineered dataset ready for modeling.
DV Agent	artifacts/dv_results/	Visualization outputs such as pred_vs_actual.png, residuals.png, and summary JSONs.
ML Agent	artifacts/ml_results/	Model predictions, evaluation metrics, and serialized model objects.
Models	models/random_forest.pkl	Persisted machine learning model for reuse or retraining.
Logs	scripts/logs/	Detailed execution logs for every agent and MCP component.

Each output file is timestamped and traceable through the respective log entries. For instance:

- The **Data Agent** output confirms data ingestion and preprocessing.
- The **ML Agent** output provides final model metrics such as RMSE, R^2, or accuracy, depending on the task type.
- The **DV Agent** output contains diagnostic plots illustrating prediction accuracy and residuals.

Here are final results :





4.7 Interpreting a Successful Run

A successful end-to-end execution is confirmed when the **Supervisor log** concludes with a statement similar to:

DataAgent, DVAgent, and MLAgent execution completed successfully.

Results stored in artifacts/ directory.

Additionally, all agent logs should contain entries marked [INFO] ... execution completed successfully without subsequent [ERROR] traces.

5. Summary

The execution workflow can thus be summarized as follows:

1. **Initialize environment** → start_all.bat
2. **Run Supervisor** → python -m supervisor_agent.agent_main
3. **Monitor progress** → via terminal or scripts/logs/
4. **Verify outputs** → in artifacts/ and models/ directories
5. **Shut down** → stop_all.bat

This ensures full reproducibility, controlled orchestration, and transparent traceability across the distributed agentic ecosystem.