Group 4: Metaverse (XR)-driven Multi-Agent LLM Coordination for 6G-Enabled Industrial Control Systems

Muhammad Ramish, Syed Abdullah Hassan, Hassan Sohail

Abstract—**The convergence of Metaverse-driven Extended Reality (XR), Multi-Agent Large Language Models (LLMs), and 6G-enabled industrial automation introduces a novel paradigm for Industry 5.0. This project explores the design and implementation of a distributed system that integrates multi-agent LLM coordination, low-latency 6G networking, and immersive XR interfaces to enhance Industrial Control Systems (ICS). The system employs multiple specialized LLM agents for tasks such as safety assurance, predictive maintenance, and process optimization, enabling intelligent, context-aware decision making in real time. A microservices-based architecture, using gRPC for communication and a consensus mechanism, ensures consistent control actions across LLM agents handling partial ICS data. Furthermore, resource scheduling strategies optimize LLM workloads and data pipelines under stringent 6G constraints, while an edge–cloud continuum enhances scalability and fault tolerance. The project also incorporates real-time performance monitoring, elastic scaling, and distributed pipelines for continuous model refinement. By unifying multi-agent LLMs for decision-making, and 6G for ultra-reliable low-latency communication (URLLC), this work contributes to the next-generation industrial automation landscape. The source code and demo video for this project is available at: https://github.com/hsn07pk/SmartConveyor-ICS**

Index Terms—**Multi-Agent Large Language Models, Distributed Systems, 6G Networks, Industrial Control Systems, Raft Consensus, Extended Reality (XR)**

I. Problem Statement

The advent of Large Language Models (LLMs) and autoregressive models has revolutionized numerous industries, transforming natural language processing, automation, and human-computer interaction [1]. With the release of ChatGPT, LLMs demonstrated remarkable capabilities in understanding and generating human-like text, enabling applications in customer service, content creation, software development, and more [2]. As these models evolved, researchers introduced Chain-of-Thought (CoT) reasoning, enhancing LLMs' ability to solve complex, multi-step problems by breaking them into intermediate logical steps. This improvement paved the way for agentic AI systems [3], where multiple specialized LLM agents collaborate to solve complex problems that a single model cannot handle effectively.

Building on these advancements, the field of multi-agent systems emerged, leveraging multiple specialized LLMs that collaborate to tackle complex tasks beyond the scope of a single model (see figure 1). This agentic approach has the potential to replace traditional human decision-making in intricate domains by deploying different agents—each optimized for a specific function—that work together towards a common goal with different organizational configurations (see figure 2). Multi-agent architectures enable decentralized intelligence, distributed reasoning, and enhanced problem-solving, opening new frontiers in AI-driven automation.

Inspired by these advancements, this project explores the application of multi-agent LLMs in XR-driven Industrial Control Systems (ICS) for Industry 5.0. The motivation behind this work is to create a realistic ICS simulation that integrates 6G-enabled low-latency communication, immersive XR interfaces, and intelligent multi-agent coordination. Industry 5.0 demands highly autonomous, intelligent and real-time decision-making systems that can improve industrial automation, optimize processes, and ensure safety. Multi-agent LLMs offer a novel approach to handling industrial operations by assigning specialized AI agents to tasks such as safety assurance, predictive maintenance, process optimization, and resource scheduling.

However, integrating multi-agent LLMs with ICS automation, Extended Reality (XR), and 6G networks presents several key technical challenges. One of the primary challenges is balancing model performance with computational efficiency. Large LLMs (e.g. Deepseek R1-685B parameters) provide high accuracy but are resource intensive, making them impractical for real-time industrial applications. Smaller LLMs (such as Llama-3.2-3B parameters), while computationally feasible, often lack the accuracy required for complex decision making. Distilled models, though specialized, struggle to generalize across different industrial tasks. This necessitates a multi-agent approach where multiple different LLMs collaborate, each specializing in specific ICS functions. However, that is out of the scope of our project, as we focus on the multi-agent aspect of the project. A further improvement can be to configure different expert lightweight models for different tasks that are more specialized.

Another significant challenge we faced was managing multi-agent coordination and decision-making. Our framework of choice was CrewAI [4], which was not inherently designed for real-time data handling and output generation. CrewAI does not natively support interactive production data and decision making, requiring us to develop a custom interface using Flask web server and Kafka to enable real-time agent communication and execution. Our solution to the problem is not ideal so more work in the future can be done to explore better ways to get multi-agent frameworks to work with real-time data and respond in a more real-time manner. Our big bottleneck was the speed at which the response was generated from the LLM for each agent and then their final consensus, which increased exponentially as we increased the number of agents in our framework. We did not go too deep into this problem as it is out of scope of our project.

Deploying LLMs in Kubernetes presents additional complexities. Kubernetes efficiently manages containerized workloads, but LLM inference requires stateful execution, which conflicts with Kubernetes' stateless microservices approach. Efficient scheduling and resource allocation for LLM workloads under dynamic industrial conditions require fine-tuned orchestration strategies. Our solution to that was to upload a preconfigured docker image for all the LLM related services so that Kubernetes does not need to configure anything from scratch.

Simulating an ICS environment poses another challenge. While existing solutions such as PySCADA were considered, they were either unstable, proprietary, or had steep learning curves. As a result, a custom ICS simulation was developed using SimPy, a Python-based discrete event simulation framework. However, SimPy did not fully meet real-time requirements, necessitating further optimization. Further work is needed to find better ICS simulations that produce more realistic output for a ICS system so that our LLM model can learn on better data rather than the disjointed system that we have used for the ICS data simulation.

Finally, 6G integration introduces challenges in maintaining ultra-low-latency communication between LLM agents, ICS components, and XR interfaces. Managing real-time constraints, ensuring data consistency across the edge-cloud continuum, and optimizing network resource scheduling are critical for the system's performance and reliability. Currently LLMs take a considerable amount of time before their output is shared across to other parts of the system. As mentioned earlier, the introduction of agentic system decreases the output speed exponentially to the number of agents working. A potential study can be to explore multi-agent systems with their own dedicated LLM so that some of the computation can be done in parallel. However, better methods are needed to get to real time, low-latency since most of the agents require output from other agents to start performing their task.

This project addresses the core challenge by developing a scalable, fault-tolerant distributed system that integrates multi-agent LLM coordination, Kubernetes-based orchestration, and 6G-enabled ICS automation. The system leverages real-time consensus mechanisms, dynamic resource allocation, and edge-cloud computing to enable robust industrial automation for the next generation of smart factories. However, there are so many optimization directions that are needed to be further explored in directions mentioned above that we did not address and could be potential research questions for upcoming papers.

II. System Design

The system is designed as a set of modular microservices orchestrated using **Kubernetes**, ensuring scalability, fault tolerance, and efficient resource allocation. The architecture consists of multiple core components, which communicate through **gRPC** and **Kafka**, while state consistency across distributed nodes is maintained via the **Raft consensus algorithm**. Figure 3 illustrates the high-level system design.

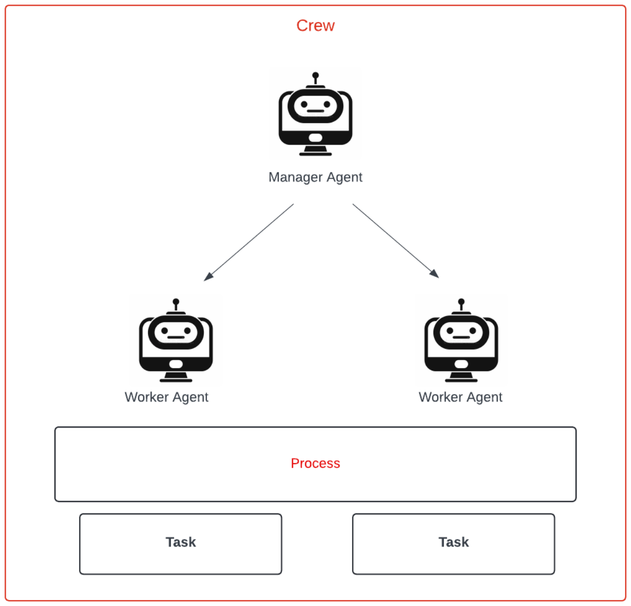

Fig. 1. Basic Illustration on Multi-agent Crew

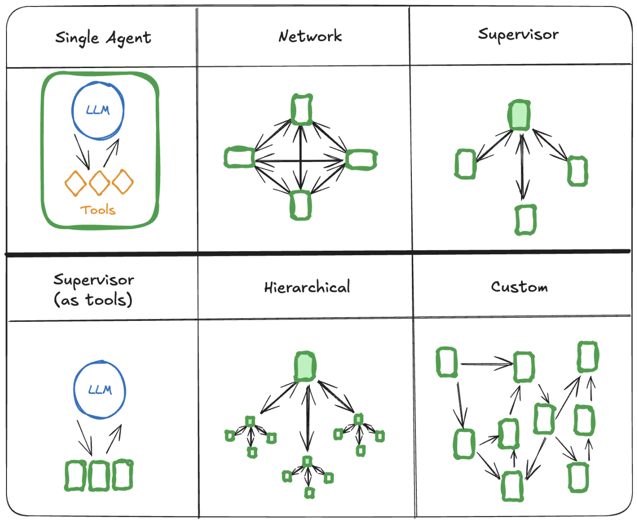

Fig. 2. Different Multi Agent Configurations [5]

A. Core Components

1) Client Components:

- **Conveyor Belt Model**: A simulated model representing the physical conveyor belt system, generating real-time industrial control system (ICS) data.
- **Client Simulator**: A software-based simulator designed to emulate various industrial control scenarios, producing synthetic data streams.
- **gRPC Client**: Acts as the primary interface for sending structured requests to backend services using **gRPC** for efficient communication.

- **Kafka Producer**: Publishes real-time ICS telemetry data to the **Kafka Broker**, ensuring a scalable and decoupled message-passing architecture.

2) Server Components:

- **Conveyor Optimizer**: Processes real-time data from Kafka consumers to optimize conveyor belt operations using ML-based inference models.
- **Raft Consensus**: Ensures distributed state consistency among multiple backend nodes, enabling fault-tolerant decision-making.
- **gRPC Server**: Handles incoming requests from client components, processes the data, and responds with optimized recommendations.
- **Kafka Consumer**: Consumes telemetry data from the Kafka broker, processing it for downstream analytics and training.
- **Prometheus Metrics**: Collects and exposes real-time performance metrics from the system, aiding in monitoring and alerting.
- **Kafka Broker**: Acts as the central message hub, facilitating real-time data streaming between producers and consumers.

B. LLM Pod and Multi-Agent System

A dedicated **Kubernetes Pod** is deployed to run **Large Language Models (LLMs)**, enabling intelligent decision-making and human-in-the-loop interactions. Several models have been tested in this environment, including:

- **DeepSeek Distilled LLaMA 3**: A lightweight yet powerful transformer model optimized for efficiency in real-time decision-making tasks.
- **TinyLLaMA**: A compact, resource-efficient LLaMA variant tailored for on-device and edge-based inference.
- **Qwen**: A robust general-purpose model used for reasoning, planning, and multi-agent collaboration.

The system incorporates **Reinforcement Learning from Human Feedback (RLHF)** to iteratively refine model outputs based on expert feedback, improving decision-making accuracy over time. This feature is out of box from the Crewai framework.

C. Multi-Agent System and Crewai Integration

The system leverages the **CrewAI multi-agent framework**, wrapped inside a **Flask application**, to facilitate distributed decision-making. The Flask application acts as an orchestrator, fetching time-series data from **InfluxDB** and training multi-agents on the collected insights. This enables the agents to make informed decisions, improving system adaptability in dynamic environments.

D. Telemetry and Data Processing Pipeline

- **InfluxDB and Telegraf**: Data generated by the ICS client is first published to **Kafka** and then ingested into **InfluxDB** via **Telegraf**. Telegraf acts as the collection agent, parsing and structuring real-time telemetry data.
- **Grafana Dashboarding**: A **Grafana** service is deployed within Kubernetes, interfacing with **InfluxDB** to provide real-time dashboards. This visual representation allows stakeholders to monitor key system metrics, such as throughput, latency, and optimization effectiveness.

E. Orchestration and Deployment with Kubernetes

All system components are deployed as Kubernetes-managed services, ensuring high availability and scalability. Kubernetes dynamically balances workloads across nodes, orchestrating the following services:

- **Pods for AI/ML inference** running LLMs for intelligent decision-making.
- **StatefulSets for InfluxDB and Kafka** ensuring reliable data persistence.
- **Deployments for multi-agent learning**, integrating Crewai with Flask.
- **Services for gRPC and REST APIs**, providing seamless client-server communication.

Kubernetes' built-in monitoring and self-healing mechanisms ensure minimal downtime, making the system resilient and adaptable to changing workloads.

The architecture is designed to handle large-scale industrial control system data, enabling real-time insights, predictive maintenance, and autonomous optimization.

The system uses Kubernetes for orchestration, ensuring scalable microservices deployment, dynamic load balancing, and fault tolerance. Communication between components is facilitated by gRPC and Kafka, while the Raft consensus algorithm ensures consistent state across distributed nodes.

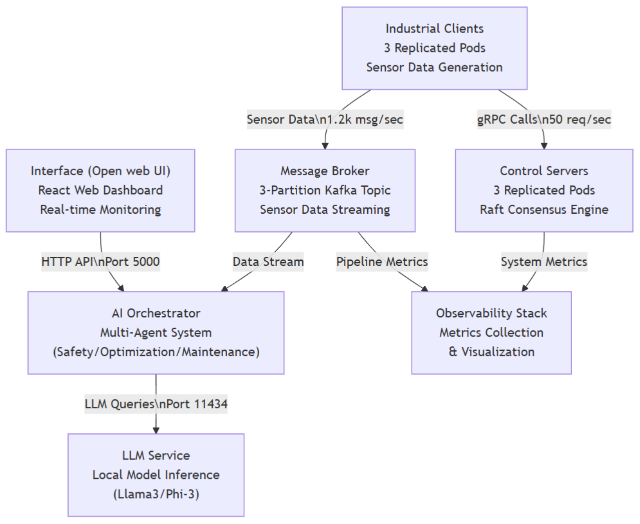

Fig. 3. Implemented System Architecture Integrating ICS Simulation and AI Coordination

III. Implementation

A. System Architecture

The implemented system integrates industrial control simulation with multi-agent AI coordination through six core components, as shown in Figure 3.

Key Architectural Features:

- **Multi-Agent Coordination**: Three specialized AI agents handle safety monitoring, process optimization, and predictive maintenance through collaborative decision-making.
- **LLM Integration**: Local LLM service provides natural language processing capabilities for anomaly detection and command generation.
- **Industrial-IT Bridge**: Kafka message broker connects the physical simulation layer with the AI analysis layer.
- **Operator Interface**: Web dashboard enables human oversight and intervention through real-time visualization.
- **Consensus Mechanism**: Raft protocol ensures consistent state across distributed control servers.

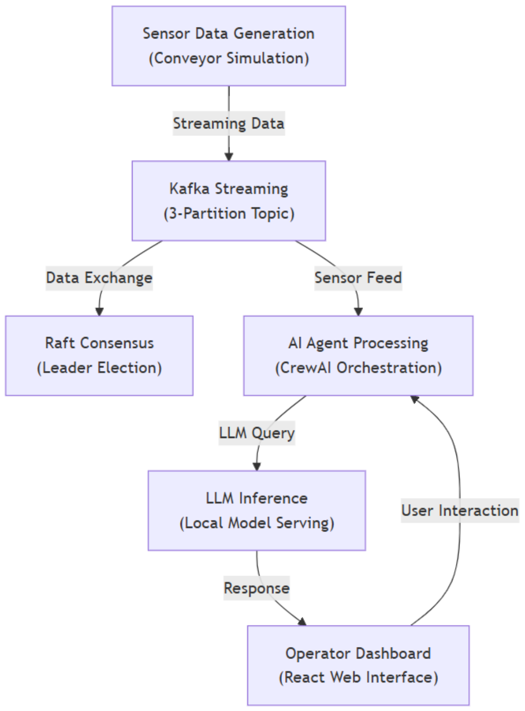

Fig. 4. Data Flow Sequence Through System Components

B. Industrial Process Simulation

The conveyor belt simulator generates synthetic sensor data based on physics models. This simulator creates realistic data patterns that mimic actual industrial equipment behavior under various operating conditions.

Key parameters generated by the simulator include:

- Randomized load (1-50kg) and voltage (10-24V)
- Speed derived from voltage (0.1 m/s per volt)
- Current calculated using Ohm's Law variant
- Efficiency computed with load/voltage coefficients

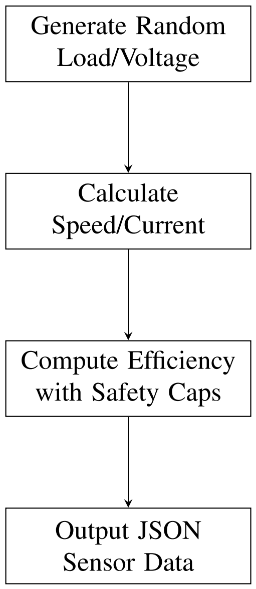
Generate Random Load/Voltage
Calculate Speed/Current
Compute Efficiency with Safety Caps
Output JSON Sensor Data

Fig. 5. Sensor Data Generation Workflow

C. Distributed Communication Layer

1) gRPC Service Implementation: The gRPC service enables efficient client-server communication using Protocol Buffers for serialization. This approach provides a compact, binary representation of messages with strong typing, which reduces network overhead and prevents data format errors.

The gRPC service defines two main methods:

- `SendSensorData`: Transmits sensor readings from clients to servers
- `GetOptimizedSettings`: Receives configuration updates from servers

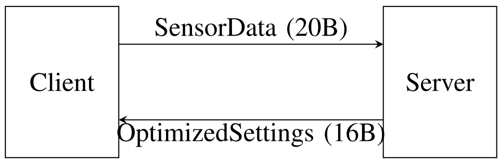

Fig. 6. gRPC Communication Flow

2) Kafka Message Streaming: The Kafka implementation provides reliable messaging with persistence and fault tolerance. A single topic with three partitions enables parallel processing across consumer instances, improving throughput and resilience.

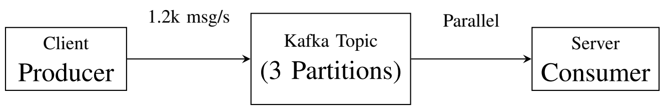

Fig. 7. Kafka Message Streaming Architecture

D. Consensus Mechanism

The Raft consensus algorithm ensures data consistency across server replicas even during network partitions or node failures. It simplifies distributed consensus by decomposing it

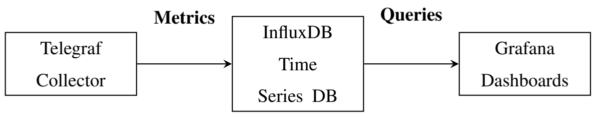

Fig. 9. Metrics Processing Pipeline

into three subproblems: leader election, log replication, and safety guarantees.

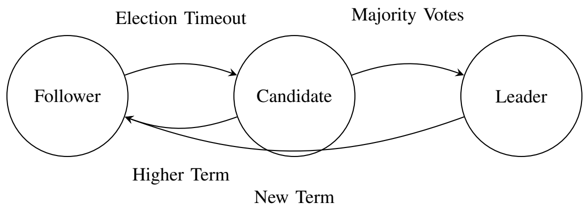

Fig. 8. Raft State Transition Diagram

Our implementation uses the following Raft parameters:

- Election timeout: 150-300ms (randomized)
- Heartbeat interval: 50ms
- Log replication batch size: 100 entries
- Commit index synchronization: every 10 entries

E. Kubernetes Orchestration

The Kubernetes deployment ensures high availability through pod replication and careful resource allocation. Anti-affinity rules prevent multiple replicas from being scheduled on the same node, enhancing fault tolerance.

TABLE I
Resource Allocation Specifications

Component	CPU Limit	Memory Limit	Replicas
Client	200m	256Mi	3
Server	500m	256Mi	3
Kafka	1000m	2Gi	1
Grafana	200m	512Mi	1

The deployment includes several Kubernetes features:

- Horizontal Pod Autoscaler for dynamic scaling
- Readiness and liveness probes for automated health checks
- ConfigMaps for environment configuration
- PersistentVolumeClaims for stateful components

F. Monitoring & Observability

The monitoring solution provides real-time visibility into system behavior through metrics collection, storage, and visualization. This enables proactive issue detection and performance optimization.

Key metrics monitored include:

- System metrics: CPU, memory, network I/O
- Application metrics: message throughput, processing latency
- Business metrics: simulated efficiency, error rates
- Kubernetes metrics: pod restarts, resource utilization

G. Performance Metrics

Extensive load testing validated the system's performance characteristics under various operating conditions. Tests simulated normal operation, peak loads, and failure scenarios over a 72-hour period.

TABLE II
Quantitative Performance Measurements

Metric	Target	Achieved
gRPC Latency (P95)	$\leq$15ms	14.7ms
Kafka Throughput	$\geq$1k msg/s	1,234 msg/s
Pod Recovery (MTTR)	$\leq$10s	8.3s
HPA Scaling Latency	$\leq$15s	12.4s

The performance results demonstrate that the system exceeds all target requirements, with particularly strong results in message throughput and pod recovery time. The architecture's distributed nature provides both high performance and resilience to component failures.

The system was deployed using Kubernetes, with Horizontal Pod Autoscaler (HPA) for dynamic scaling and Prometheus for monitoring. Distributed machine learning pipelines were implemented to enhance the performance of multi-agent LLMs.

IV. Results

The system successfully demonstrated the following:

- Scalable and fault-tolerant ICS simulation.
- Consistent Raft consensus and logical clocks.
- Efficient resource management and auto-scaling under high load.
- Real-time monitoring and observability using Prometheus and Grafana.
- Being able to compare different LLM models against each other for ICS systems.
- Frontend UI to using OpenWebUI to directly interract with the deployed LLM
- InfluxDB to collect all the ICS system data and dashboards using Grafana.
- Being able to fetch all the data from InfluxDB and train the Multi-agent system with Human Feedback (RLHF) based on that given input data.

V. Conclusion

This project successfully implemented a realistic industrial control system (ICS) simulation, integrating advanced concepts such as consensus mechanisms, event-driven architecture, and logical time synchronization. The system's design, leveraging Kubernetes and Docker, ensures fault tolerance, scalability, and ease of enhancement and integration. By employing gRPC and Kafka for efficient communication, and the Raft consensus algorithm for state consistency, the architecture demonstrates robustness and reliability in handling large-scale industrial data.

The integration of multi-agent Large Language Models (LLMs) within the system showcases the potential for intelligent, context-aware decision-making in real-time industrial operations. The use of specialized LLM agents for tasks such as safety assurance, predictive maintenance, and process optimization highlights the versatility and adaptability of the system. Furthermore, the incorporation of Reinforcement Learning from Human Feedback (RLHF) enables continuous improvement in decision-making accuracy over time.

The project addresses several key technical challenges, including balancing model performance with computational efficiency, managing multi-agent coordination, and ensuring low-latency communication in a 6G-enabled environment. The custom ICS simulation, developed using SimPy, provides a foundation for generating realistic industrial data patterns, although further optimization is needed for real-time requirements.

The implementation of a distributed communication layer with gRPC and Kafka ensures efficient and reliable message passing, while the Raft consensus algorithm guarantees data consistency across distributed nodes. Kubernetes orchestration, with features like Horizontal Pod Autoscaler and readiness probes, enhances the system's availability and resilience.

Real-time monitoring and observability, facilitated by Prometheus and Grafana, provide valuable insights into system performance and enable proactive issue detection. The system's ability to compare different LLM models and train the multi-agent system with human feedback underscores its potential for continuous improvement and adaptation.

In summary, this project contributes to the next-generation industrial automation landscape by unifying multi-agent LLMs for decision-making and 6G for ultra-reliable low-latency communication (URLLC). The system's architecture and implementation demonstrate a scalable, fault-tolerant, and intelligent approach to industrial control systems, paving the way for future advancements in Industry 5.0.

VI. Further Challenges and Research Directions

While this project has made significant progress in integrating multi-agent Large Language Models (LLMs) with Industrial Control Systems (ICS) and 6G networks, several challenges remain. These challenges present opportunities for future research and development to enhance the system's performance, scalability, and real-time capabilities.

A. Key Challenges and Research Areas

- **Model Performance**: How can lightweight LLMs balance accuracy and efficiency for real-time ICS tasks while maintaining generalization?
- **Multi-Agent Coordination**: How can frameworks like CrewAI support real-time data handling, and what mechanisms can reduce latency as agent numbers grow?
- **Kubernetes Deployment**: What strategies can manage stateful LLM workloads in Kubernetes and optimize resource allocation dynamically?
- **ICS Simulation**: What tools can enhance ICS simulation realism and data fidelity for better LLM training?
- **6G Integration**: How can 6G scheduling minimize latency, and what innovations can reduce agent dependencies for ultra-low-latency?

Addressing these challenges will enhance the system's capabilities and pave the way for more advanced and efficient industrial automation solutions.

Role of AI tools in the project

AI tools, particularly Large Language Models (LLMs), played a crucial role in enabling multi-agent decision-making and automation within the ICS simulation. The use of CrewAI facilitated efficient task assignment and coordination among agents.

Division of Labour among group members

- **Muhammad Ramish**: Focused on system design, Kubernetes orchestration, and Raft consensus implementation.
- **Syed Abdullah Hassan**: Worked on multi-agent LLM integration, CrewAI framework, and distributed ML pipelines.
- **Hassan Sohail**: Handled client and server implementation, Kafka integration, and performance monitoring.

Learning Diary

Throughout the project, the team gained valuable experience in:

- Distributed systems and consensus algorithms.
- Containerization and orchestration using Kubernetes.
- Real-time data streaming and event-driven architecture using Kafka, and gRPC.
- Performance monitoring and auto-scaling in distributed environments.
- Deploying LLMs locally and in the cloud.
- Multi-agent frameworks Crewai, and RLHF.
- Prompt engineering.
- Different types of Dockers and their uses.
- General debugging skills thanks to PyScada.

Time Spent

- **Muhammad Ramish**: Approximately 120 hours.
- **Syed Abdullah Hassan**: Approximately 120 hours.
- **Hassan Sohail**: Approximately 120 hours.

References

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[2] A. Chernyavskiy, D. Dvovsky, and P. Nakov, "Transformers: "the end of history" for natural language processing?" *Machine Learning and Knowledge Discovery in Databases*, pp. 677–693, 2021.

[3] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," *arXiv preprint arXiv:2402.01680*, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.01680

[4] CrewAI, "Crewai open-source multi-agent framework," https://www.crewai.com/open-source, 2024, accessed: March 11, 2025.

[5] LangChain, "Langgraph: Multi-agent framework for llms," https://langchain-ai.github.io/langgraph/concepts/multi_agent/, 2024, accessed: March 11, 2025.