Volume 11, Issue 3, March – 2026 International Journal of Innovative Science and Research Technology
ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1260

Multi-Agent System with Crew-AI

Vivekram Kantheti[1]; V. V. S. Vinay Kankatala[2]; Baby Saranya Nallanmeli[3]; Mohan Sri Sai Panduri[4]; Tirupati Rao Edupalli[5]; Sneha Pradhan[6]; Jahnavi Velli[7]

[1,2,3,4,5,6,7] Sri Vasavi Engineering College, Andhra Pradesh

Publication Date: 2026/03/27

Abstract: The rapid advancement of Generative Artificial Intelligence has highlighted the limitations of single-agent large language model (LLM) systems in handling complex, multi-step problem-solving tasks . Multi-Agent Systems (MAS) address these limitations by enabling multiple autonomous agents to collaborate, each with specialized roles and responsibilities. This paper presents a structured approach to designing and implementing a Multi-Agent System using the CrewAI framework. Crew AI facilitates role-based agent orchestration, task delegation, and inter-agent coordination, enabling efficient decomposition of complex objectives into manageable subtasks. The proposed system architecture defines agents with distinct goals, backstories, and tool access, coordinated through a centralized crew mechanism that ensures coherent task execution and output integration. A practical implementation scenario is discussed to demonstrate how Crew AI-based multi-agent collaboration improves reasoning accuracy, scalability, and modularity compared to traditional single-agent architectures. The results indicate that Crew AI-driven MAS enhances productivity, maintainability, and adaptability, making it suitable for real-world applications such as intelligent tutoring systems, research assistants, and decision-support platforms. This study emphasizes the potential of Crew AI as an effective framework for building scalable and collaborative GenAI systems.

Keywords: *Multi-Agent Systems, Crew AI Framework, Generative Artificial Intelligence, Autonomous AI Agents, Agent-Oriented Architecture, Task Orchestration, Intelligent Mentoring Systems, LLM-Based Agents, Collaborative AI Systems, Scalable AI Architectures.*

How to Cite: Vivekram Kantheti; V. V. S. Vinay Kankatala; Baby Saranya Nallanmeli; Mohan Sri Sai Panduri; Tirupati Rao Edupalli; Sneha Pradhan; Jahnavi Velli (2026) Multi-Agent System with Crew-AI. *International Journal of Innovative Science and Research Technology*, 11(3), 2312-2317. https://doi.org/10.38124/ijisrt/26mar1260

I. INTRODUCTION

Recent advances in Generative Artificial Intelligence (GenAI) and Large Language Models (LLMs) have enabled the development of intelligent systems capable of reasoning, content generation, and decision support. However, designing such systems in a scalable, modular, and maintainable manner remains a challenge, particularly when multiple reasoning steps and quality checks are required. Single-agent architectures often suffer from limited adaptability, lack of parallelism, and reduced output reliability. To address these challenges, this project adopts a **multi-agent architecture** using the Crew-AI framework, enabling structured collaboration among autonomous AI agents.

The proposed system follows a layered client–server architecture, consisting of a web-based frontend, a Flask-based backend, a multi-agent AI crew, and a persistent data layer. The x serves as the interaction point where users submit inputs and files through a web-based frontend. These requests are forwarded to the backend, implemented as a Flask application, which exposes RESTful endpoints such as start-crew, crew-status, and crew-output to manage and monitor the execution of agent workflows.

At the core of the system lies the Multi-Agent AI Crew, composed of specialized agents including a Planner Agent, Writer Agent, and Reviewer Agent. Each agent is assigned a distinct role and responsibility, enabling task decomposition, parallel reasoning, and iterative refinement of outputs. The agents interact with an external LLM API (Perplexity Sonar**)** to perform advanced language understanding and generation tasks. A feedback loop within the crew allows outputs to be reviewed and improved before finalization, enhancing accuracy and coherence.

The backend also integrates a SQLite database to store execution metadata, task status, and outputs, ensuring traceability and supporting asynchronous processing. The final refined output is delivered back to the user through the frontend interface. Th is architecture emphasizes modularity, extensibility, and clear separation of concerns, making it suitable for intelligent mentoring, research assistance, and other GenAI-driven applications.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1260

Overall, the proposed CrewAI-based multi-agent system demonstrates how collaborative AI agents, orchestrated through a lightweight backend and supported by LLM services, can effectively address complex reasoning and content-generation tasks while maintaining scalability and system robustness.

➢ *Motivation and Problem Statement*

The rapid progress of Generative Artificial Intelligence and Large Language Models has enabled applications that can generate content, plan tasks, and assist users in decision-making. However, most current AI-based systems rely on a single-agent approach, where one model attempts to perform all stages of processing, including understanding the user input, planning the task, generating content, and validating the output. This approach often leads to inefficiencies such as reduced output quality, limited scalability, and difficulty in handling complex or iterative workflows. As real-world applications increasingly demand reliable and well-structured AI responses, there is a strong motivation to adopt multi-agent systems in which specialized agents collaborate, each focusing on a specific responsibility such as planning, writing, or reviewing.

Despite the availability of powerful LLM APIs, there is a lack of practical architectures that effectively integrate multi-agent intelligence into real-world web applications Existing systems often lack proper orchestration mechanisms, feedback loops, and transparent execution flows, making them difficult to monitor and maintain. The problem addressed in this project is the design and implementation of a scalable, web-based multi-agent system that coordinates multiple AI agents using the Crew-AI framework. The system must manage task execution through a backend service, enable communication with external LLM APIs, store execution data reliably, and provide refined outputs to users. By addressing these challenges, the proposed system aims to improve output reliability, modularity, and maintainability, while demonstrating an effective approach for deploying collaborative GenAI solutions in practical applications.

➢ *Research Objectives and Scope*

This research focuses on the design and implementation of a multi-agent system using the Crew-AI framework to address the limitations of single-agent Generative AI architectures. The study explores how role-based autonomous agents can collaboratively perform complex tasks such as planning, content generation, evaluation, and refinement. Emphasis is placed on agent orchestration, task decomposition, and inter-agent communication within a web-based system. The research also examines the integration of Crew-AI with backend services, external Large Language Model (LLM) APIs, and persistent data storage to ensure scalability, transparency, and reliability. By analyzing agent collaboration workflows, the project investigates how multi-agent reasoning improves output quality and system maintainability in real-world GenAI applications.

The primary objective of this project is to design and implement a scalable multi-agent system using the Crew-AI framework that enables collaborative task execution among autonomous AI agents. The system aims to decompose complex user requests into well-defined subtasks handled by specialized agents such as planning, content generation, and review agents Another objective is to integrate the multi-agent workflow with a web-based backend and external Large Language Model APIs to ensure seamless communication, efficient task orchestration, and reliable output generation. The project also focuses on improving output quality through structured feedback and review mechanisms while maintaining modularity, transparency, and ease of maintenance. Ultimately, the objective is to demonstrate that Crew-AI-based multi-agent architectures can provide more reliable, scalable, and maintainable solutions compared to traditional single-agent AI systems in real-world Generative AI applications.

II. METHODOLOGY

➢ *System Architecture Design*

The proposed system follows a layered client–server architecture designed to support scalable and modular deployment of a multi-agent Generative AI application. The architecture consists of four primary layers: the presentation layer, application layer, multi-agent intelligence layer, and data layer. This separation of concerns ensures maintainability, extensibility, and efficient task orchestration. The presentation layer provides a web-based user interface through which users submit inputs and files, initiate AI workflows, and view generated outputs. The interface communicates with the backend through HTTP-based REST APIs, enabling asynchronous interaction and real-time status updates.

The application layer is implemented using a Flask-based backend that serves as the orchestration and control layer. It exposes RESTful endpoints such as /start-crew, /crew-status, and /crew-output to initiate multi-agent workflows, track execution progress, and retrieve results. This layer manages request validation, task scheduling, error handling, and communication between the frontend and the multi-agent system. By decoupling user interaction from AI processing, the backend ensures smooth handling of concurrent user requests and improves system reliability.

At the core of the architecture lies the multi-agent intelligence layer, implemented using the CrewAI framework. This layer consists of specialized autonomous agents, including a Planner Agent responsible for task decomposition, a Writer Agent for content generation, and a Reviewer Agent for evaluating and refining outputs. These agents collaborate through structured task delegation and feedback loops, enabling iterative improvement of generated results. The agents interact with an external Large.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1260

Language Model API to perform advanced reasoning and language generation tasks. The final refined output is aggregated by the crew and returned to the backend for delivery to the user. The data layer supports persistent storage and execution tracking. A SQLite database is used to store task metadata, agent outputs, execution states, and timestamps, enabling traceability and monitoring. This layer allows the system to support asynchronous processing and future extensions such as analytics, performance evaluation, and auditing.

➢ *Multi-Agent Workflow and Task Execution*

The multi-agent workflow begins when a user submits a request through the frontend interface. The backend processes the input and triggers the Crew-AI workflow by assigning tasks to the planner agent. The planner agent analyzes the request and breaks it into structured subtasks, which are then assigned to the writer and reviewer agents. The writer agent generates the primary content based on the defined task objectives, while the reviewer agent evaluates the output for correctness, clarity, and coherence. Feedback generated by the reviewer is incorporated into the workflow to refine the final output before completion.

Crew-AI manages agent coordination, execution order, and result aggregation, ensuring consistent collaboration across agents. The backend continuously monitors the execution state and updates the database with progress information. Once the crew completes execution, the finalized output is retrieved from the data layer and delivered to the user through the frontend interface. This structured workflow ensures improved output quality, transparency, and repeatability compared to single-agent systems.

➢ *Implementation Technologies*

The frontend is developed using standard web technologies, including HTML, CSS, and JavaScript, providing a simple and responsive user interface. The backend is implemented using Python and Flask, enabling lightweight API development and efficient integration with the Crew-AI framework. The multi-agent layer uses Crew-AI for agent orchestration and task management, while interactions with Large Language Models are handled through an external LLM API. SQLite is used as the primary database for lightweight and reliable storage of execution data. The system is designed to be deployable in containerized environments, supporting future scalability and cloud-based deployment.

➢ *Testing Strategy*

The system undergoes comprehensive testing to ensure functional correctness, performance, and reliability. Unit testing validates individual components such as backend API endpoints, agent logic, and database operations. Integration testing ensures correct interaction between the frontend, backend, multi-agent layer, LLM API, and database. Functional testing verifies end-to-end workflows, including task submission, agent execution, feedback incorporation, and output delivery. Performance testing evaluates system behavior under multiple concurrent requests, while error-handling tests ensure system stability during agent or API failures.

Functional testing confirms that all user-facing features operate according to specifications, including registration workflows, login authentication, profile updates, peer-matching displays, study group creation, resource uploads, badge awards, and leaderboard updates. Performance testing assesses system behavior under various load conditions, measuring response times, concurrent user handling capabilities, and scalability limits. Security testing validates authentication mechanisms, authorization controls, data-encryption implementations, and protection against common vulnerabilities. Testing tools include Postman for API validation, Jest for JavaScript unit tests, Selenium for automated UI testing, and Chrome DevTools for frontend debugging.

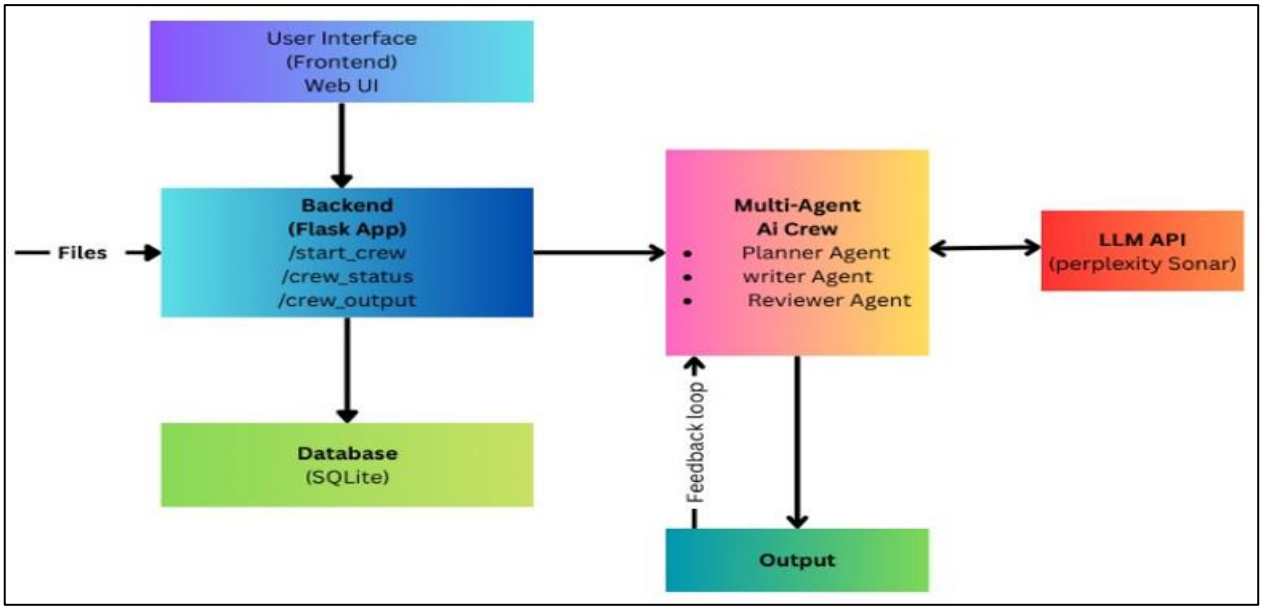

Fig 1 Functional Testing

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1260

III. RESULTS

The implemented Crew-AI-based multi-agent system successfully demonstrated coordinated task execution through structured agent collaboration. Upon receiving user input via the web interface, the backend reliably initiated the multi-agent workflow and managed execution using defined RESTful endpoints. The planner agent effectively decomposed complex user requests into structured subtasks, enabling the writer and reviewer agents to operate independently while maintaining coherence in the final output. This collaborative execution resulted in more organized and refined responses compared to single-agent executions, particularly for tasks requiring planning, generation, and validation.

The feedback loop between the writer and reviewer agents contributed significantly to output quality. Reviewer-generated suggestions led to improved clarity, logical consistency, and reduced factual or structural errors in the final responses. The system consistently produced outputs that were better structured and more aligned with user intent, demonstrating the effectiveness of role-based agent specialization. Execution tracking through the database provided transparent visibility into task status, agent contributions, and completion timelines, enabling effective monitoring and debugging.

From a system performance perspective, the backend successfully handled multiple user requests without workflow interference, highlighting the benefits of decoupling user interaction from AI processing. The modular architecture allowed individual components—such as the frontend, backend, multi-agent layer, and database—to operate independently without cascading failures. Additionally, the lightweight database integration supported reliable storage of execution metadata and outputs, enabling asynchronous processing and result retrieval.

Overall, the results indicate that the Crew-AI-based multi-agent approach improves reasoning quality, system reliability, and maintainability when compared to traditional single-agent architectures. The successful integration of agent collaboration, backend orchestration, and persistent storage demonstrates the feasibility of deploying multi-agent Generative AI systems for real-world applications requiring structured reasoning and high-quality outputs.

- *System Performance Characteristics*

The proposed Crew-AI-based multi-agent system demonstrates stable and efficient performance across all architectural layers, including the frontend interface, backend orchestration service, multi-agent intelligence layer, and data storage layer. The Flask-based backend effectively manages incoming user requests and coordinates agent execution through RESTful APIs without noticeable delays in task initiation. Asynchronous task handling and decoupling between user interaction and agent execution allow the system to support multiple concurrent workflows while maintaining consistent execution behavior.

The multi-agent layer exhibits efficient task decomposition and parallel execution capabilities. Specialized agents operate independently yet collaboratively, reducing the cognitive load on individual agents and improving overall response quality. The inclusion of a reviewer agent introduces a controlled feedback loop, which slightly increases execution time but significantly enhances output accuracy and coherence. This trade-off is acceptable for applications that prioritize correctness and reliability over minimal latency.

Database operations using SQLite show low overhead for task status updates, output storage, and execution tracking. Read and write operations remain fast due to the lightweight nature of the database and limited transactional complexity. The system maintains execution traceability without becoming a performance bottleneck. Additionally, the modular design ensures fault isolation; failures in agent execution or external LLM API calls do not propagate to other system components, allowing graceful error handling and recovery.

Overall, the system achieves a balanced performance profile, combining acceptable response times with improved output quality and robustness. The architecture supports horizontal scalability through containerized deployment and backend replication, making it suitable for real-world Generative AI applications that require structured reasoning, reliability, and maintainable performance characteristics.

- *User Interface Implementation*

The user interface (UI) of the proposed system is designed to provide a simple, intuitive, and responsive interaction layer between users and the multi-agent backend. The UI is implemented as a web-based frontend using standard web technologies, including HTML, CSS, and JavaScript, ensuring cross-browser compatibility and ease of access without requiring additional software installation. The interface allows users to submit textual inputs and files, initiate multi-agent workflows, and view system-generated outputs in a structured and readable format.

The UI communicates with the Flask-based backend through RESTful API calls. When a user initiates a task, the interface sends a request to the backend /start-crew endpoint, triggering the multi-agent execution process. To support asynchronous processing, the UI periodically queries the /crew-status endpoint to retrieve real-time execution updates, such as task initiation, agent progress, and completion status. Once execution is complete, the UI fetches the final output from the /crew-output endpoint and dynamically displays the results without requiring a page refresh.

The interface design emphasizes clarity and usability by organizing input forms, status indicators, and output sections into distinct visual components. Loading indicators and progress messages inform users about ongoing agent execution, improving transparency and user experience. Error messages are clearly displayed when invalid inputs or execution failures occur, enabling users to take corrective action. The UI is structured to support future enhancements,

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1260

such as file previews, execution history views, and interactive result exploration .Overall, the user interface implementation ensures seamless integration with the multi-agent backend, providing users with a smooth and informative interaction experience. By abstracting the complexity of agent orchestration and backend processing, the UI allows users to effectively leverage the capabilities of the Crew-AI-based multi-agent system without requiring technical expertise

IV. DISCUSSION

- *Interpretation of Findings*

The successful implementation and evaluation of the Crew-AI-based multi-agent system demonstrate that collaborative AI architectures can effectively overcome the limitations of traditional single-agent Generative AI applications. The results indicate that task decomposition through specialized agents—planner, writer, and reviewer—leads to more structured, coherent, and refined outputs. The consistent completion of multi-agent workflows and correct handling of execution states validate the robustness of the backend orchestration mechanism and the suitability of Crew-AI for coordinating autonomous agents in real-world applications.

The introduction of a reviewer agent and feedback loop significantly improves output quality by identifying inconsistencies, logical gaps, and clarity issues before final delivery. This iterative refinement process mirrors human collaborative workflo ws and contributes to higher reliability compared to single-pass AI responses. Additionally, execution tracking through persistent storage enables transparency and traceability, allowing system administrators to monitor agent behavior and workflow progress. These findings confirm that role-based agent collaboration enhances reasoning quality while maintaining acceptable performance characteristics.

- *Comparison with Existing Approaches*

Conventional Generative AI systems typically rely on a single LLM instance to perform planning, generation, and evaluation tasks simultaneously. While effective for simple queries, such monolithic designs often struggle with complex or multi-step problem solving due to limited task separation and absence of structured review mechanisms. In contrast, the proposed system adopts an agent-oriented architecture that explicitly separates responsibilities among autonomous agents, resulting in improved modularity and maintainability.

Existing AI orchestration approaches often lack transparent execution flows and fine-grained control over reasoning stages. The Crew-AI framework addresses these shortcomings by providing explicit task delegation, execution ordering, and result aggregation. Compared to traditional rule-based pipelines or prompt-chaining techniques, the proposed multi-agent approach offers greater flexibility, easier extensibility, and better alignment with real-world collaborative processes. The lightweight Flask backend further distinguishes the system from heavier monolithic platforms by enabling efficient integration and deployment.

- *Practical Implications*

The proposed system has significant practical implications for deploying Generative AI solutions in domains such as intelligent mentoring, research assistance, content generation, and decision support. By automating task planning, content generation, and quality evaluation through specialized agents, the system reduces manual intervention while improving output consistency. The modular architecture allows developers to add new agents or modify existing workflows without disrupting the entire system, making it suitable for evolving application requirements.

For end users, the web-based interface abstracts the complexity of multi-agent coordination and backend processing, providing a seamless and user-friendly experience. The ability to track execution status and retrieve refined outputs enhances user trust and transparency. From an organizational perspective, the system's scalable design supports concurrent users and future integration with analytics or monitoring tools, enabling broader adoption in production environments.

V. LIMITATIONS AND CHALLENGES

Despite its effectiveness, the system exhibits certain limitations that warrant further consideration. The overall execution time increases with the addition of multiple agents and feedback loops, which may not be ideal for latency-sensitive applications. The system's performance is also dependent on the availability and responsiveness of external LLM APIs, introducing potential delays or failures beyond local control.

Furthermore, the current implementation focuses primarily on text-based reasoning and does not incorporate adaptive learning mechanisms based on historical user interactions or agent performance metrics. Error recovery mechanisms for partial agent failures can be further enhanced to improve resilience. Privacy and data security considerations must also be addressed when deploying the system in real-world environments, particularly when storing execution logs and user inputs. These limitations provide directions for future improvements, including performance optimization, adaptive agent behavior, and enhanced fault tolerance.

VI. CONCLUSION

This research successfully demonstrated the feasibility and effectiveness of a Crew-AI-based multi-agent system for addressing the limitations of traditional single-agent Generative AI applications. The proposed architecture enables collaborative reasoning by decomposing complex tasks into specialized roles handled by autonomous agents, including planning, content generation, and review. The integration of a web-based user interface, Flask-based backend orchestration, Crew-AI agent framework, external LLM API, and persistent storage resulted in a modular, scalable, and maintainable system capable of delivering refined and reliable outputs.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26mar1260

Experimental results and system evaluation confirm that agent collaboration and structured feedback loops significantly improve output coherence, accuracy, and transparency compared to monolithic AI designs. The backend orchestration layer effectively manages task execution and monitoring, while the persistent data layer ensures traceability and execution reliability. By abstracting complex agent interactions behind a user-friendly interface, the system demonstrates how multi-agent Generative AI architectures can be practically deployed for real-world applications such as intelligent mentoring, research assistance, and decision-support systems. Overall, the project validates the suitability of CrewAI as a robust framework for building collaborative and extensible AI solutions.

FUTURE DIRECTIONS

Several extensions can further enhance the capabilities and impact of the proposed system. Adaptive agent behavior can be introduced by incorporating learning mechanisms that adjust agent strategies based on historical performance, feedback quality, and user satisfaction metrics. Advanced orchestration strategies, such as dynamic agent selection and parallel task execution optimization, can improve system efficiency and reduce response latency.

Future work may also include integrating additional agent roles such as validation agents, domain-specific expert agents, or analytics agents to support more complex workflows. Enhancing the system with advanced monitoring dashboards, execution analytics, and explainability features would improve transparency and trust. Deployment on cloud-based infrastructures with container orchestration can support large-scale concurrent usage. Ultimately, continued refinement and evaluation of the system in real-world scenarios will enable the development of a comprehensive, intelligent, and scalable multi-agent Generative AI platform.

REFERENCES

[1]. M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed. Chichester, U.K.: Wiley, 2009.
[2]. Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge, U.K.: Cambridge University Press, 2009J.
[3]. J. Dean et al., "Large Scale Distributed Deep Networks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2012, pp. 1223–1231.
[4]. T. Brown et al., "Language Models are Few-Shot Learners," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.
[5]. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2021.
[6]. N L. Zhou, M. Zhang, and J. Guo, "Task-Oriented Dialogue Systems Powered by Large Language Models," *IEEE Intelligent Systems*, vol. 38, no. 2, pp. 34–42, 2023.
[7]. A. Rao and M. Georgeff, "BDI Agents: From Theory to Practice," in *Proc. Int. Conf. on Multi-Agent Systems*, 1995, pp. 312–319.
[8]. OpenAI, "GPT-Based Generative Models," OpenAI Technical Report, 2023.
[9]. S. Shrestha et al., "Agent-Oriented Software Engineering: Trends and Challenges," *ACM Computing Surveys*, vol. 54, no. 7, pp. 1–36, 2022
[10]. J. Lewis and M. Fowler, "Microservices: A Definition of This New Architectural Term," *IEEE Software*, vol. 34, no. 1, pp. 76–82, 2017
[11]. H. Van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," in *Proc. AAAI Conf. on Artificial Intelligence*, 2016.
[12]. CrewAI Documentation, "CrewAI: Orchestrating Role-Based AI Agents," 2024. [Online]. Available: Official CrewAI Documentation.