International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 1, Jan-Feb 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Agentic AI Project Manager: Multi-Agent Orchestration with Local LLMs

Aryan Kamble*
Department of Artificial Intelligence and Data Science, KLE College of Engineering and Technology, Karnataka, India
Email: aryank251203@gmail.com

----------**********----------

Abstract:

The recent rise of large language models (LLMs) and autonomous multi-agent frameworks has transformed software development and project automation. This paper presents the design and implementation of the Agentic AI Project Manager, an autonomous orchestration system that can generate, assign, execute, test, and deliver full-stack web and AI/ML projects end to-end using locally hosted language models. The proposed system integrates multiple open-source 7B models such as Mistral, StarCoder2, and CodeLlama with CrewAI orchestration for multi-role task delegation. Each agent assumes a specific role, ranging from project planning and development to integration, testing, and delivery. The framework automatically generates complete functional projects, including web applications, backend APIs, machine learning pipelines, and documentation, all delivered as a downloadable ZIP file. The system is designed for local execution with configurable timeout settings to accommodate limited hardware resources. This work demonstrates a step toward practical, resource-aware AI-driven project automation capable of operating offline, ensuring privacy and reproducibility.

Keywords — **Multi-Agent Systems, LLM Orchestration, CrewAI, Agentic AI, Software Automation, Local LLMs**

----------**********----------

I. INTRODUCTION

The automation of software engineering through artificial intelligence (AI) has evolved rapidly from simple autocompletion tools to autonomous systems capable of planning and executing complex engineering tasks. Large language models enable agents that simulate high-level roles—project manager, developer, tester—and collaborate to transform human intents into runnable software. While cloud-hosted models provide large compute capacity, they introduce concerns about cost, privacy, and reproducibility. Local deployment of open-source models reduces these risks but introduces new engineering challenges: memory constraints, inference latency, and model orchestration complexity. This paper presents the Agentic AI Project Manager, a local first framework designed to convert high-level project descriptions into working full-stack web applications and lightweight AI/ML projects. The system leverages CrewAI for multi-agent orchestration and an Ollama-hosted local runtime to run LLMs such as Mistral 7B (manager roles), StarCoder2 7B (senior developer and tester roles), and CodeLlama-7B (junior developer roles). The orchestration enforces modularization, role separation, integration testing, and automatic packaging into a downloadable ZIP archive. Importantly, the system focuses primarily on generating web and full-stack projects. For machine learning tasks it intentionally uses only built-in scikit-learn datasets (e.g., Iris, Wine, Digits, fetch_california_housing). This design choice guarantees reproducibility and allows the system to generate complete ML pipelines (data loading, feature processing, model training,

evaluation, and deployment endpoints) even when dataset upload functionality is not provided. We implemented the framework using a modular FastAPI backend that exposes a simple endpoint for project submission and a single-page web UI for inputs. Agents coordinate via CrewAI-managed message passing; generated projects are assembled under a generated/ directory and packaged by a Delivery Agent. The following sections review prior work, describe the system architecture and methods, and present implementation details and representative results.

II. LITERATURE REVIEW

The study of autonomous agents for software engineering and task automation has matured rapidly. Foundational systems such as AutoGPT and BabyAGI illustrated continuous LLM-driven task generation and execution, showing how an LLM can propose, prioritize, and attempt subtasks autonomously [1], [2]. These systems, however, often lacked robust role-based decomposition and suffered from uncontrolled recursion and instability.

MetaGPT proposed a structured, role-based multi-agent architecture that aligns more closely with human workflows, improving coordination and traceability by enforcing role defined responsibilities and standardized outputs [3]. The design of MetaGPT influenced contemporary orchestration practices by demonstrating the benefits of SOP-like instruction templates for agents and the value of explicit testing and verification loops.

On the model side, code-focused models such as StarCoder and StarCoder2 were developed to specialize in program synthesis and code completion tasks, often outperforming more general-purpose models on code benchmarks [4]. CodeLlama variants provide further improvements for code generation tasks, enabling more reliable modular function generation and code-level reasoning [5]. Conversely, general-purpose instruction-following models like Mistral-7B excel at task planning, multi-step reasoning, and instruction decomposition, making them suitable for manager-level roles in multi-agent systems [6].

A parallel thread of research examines local execution of LLMs for privacy and reproducibility. Tools such as Ollama and local inference runtimes enable the deployment of 7B class models on commodity hardware via quantization and model offloading strategies [7], [8]. Empirical guides and community repositories highlight trade-offs: quantized models (int8/int4) reduce memory footprint but can impact generation fidelity, while fp16 operation delivers higher quality at larger memory cost [9]. These trade-offs motivated our configurable timeout windows and the choice to restrict ML dataset sources to scikit-learn built-ins to ensure deterministic reproducibility. Finally, the literature on software automation emphasizes the need for integrated testing and validation. Automated integration testers and contract-checking agents reduce the incidence of cascading failures and behavioral drift in generated code. Incorporating dedicated testing agents—responsible for integration testing and final system validation—emulates standard engineering practice and significantly increases the reliability of autonomously generated projects [3], [10].

III. SYSTEM ARCHITECTURE

The architecture of the Agentic AI Project Manager integrates multiple autonomous agents coordinated through the CrewAI framework. Each agent acts as an independent reasoning entity with a specific responsibility, allowing the system to mimic the collaborative workflow of a real-world software engineering team.

The system operates on a modular architecture with the following layers:

1) ***Frontend Interface:*** A responsive web interface built with HTML, CSS, and JavaScript allows users to submit project ideas. The UI communicates with the backend using RESTful APIs.

2) ***Backend Orchestration*:** Implemented using FastAPI, it acts as the central control hub. It receives project ideas, triggers agent

orchestration, and manages project lifecycle stages.

3) ***CrewAI Layer:*** This layer coordinates communication between specialized agents using structured message passing. Each agent runs on a local Ollama model instance.

4) ***Agent Roles:*** Eight agents operate collaboratively Senior Manager, Project Manager, Senior Developer, Junior Developers, Integrators, Testers, and Delivery Agents—each driven by distinct LLMs.

5) ***Project Generation Layer:*** Once tasks are distributed, developer agents generate code for full-stack web applications and AI/ML workflows (limited to scikit-learn datasets for reproducibility).

6) ***Testing and Packaging:*** The integrator and tester agents verify functionality and package the output into a downloadable ZIP file containing all project files and documentation.

The architecture ensures scalability, modularity, and transparency. Communication remains fully localized to maintain

privacy and efficiency, making the system ideal for resource constrained environments. This configuration ensures smooth orchestration of multiple LLM instances (Mistral, StarCoder2, CodeLlama) without significant latency or timeout failures.

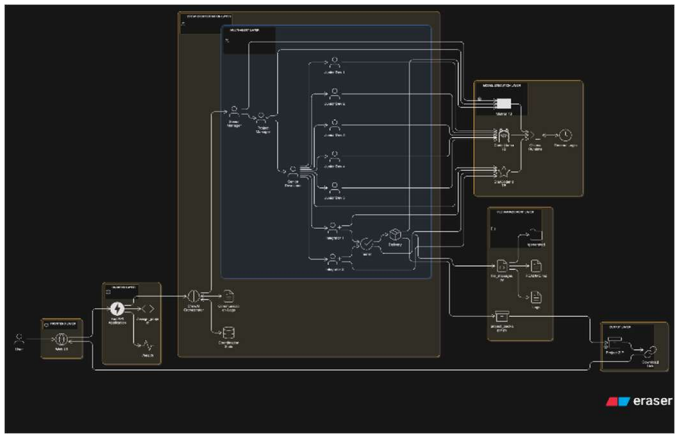

Fig. 1. System Architecture of the Agentic AI Project Manager showing multi-agent orchestration, local LLM interaction, and automated project lifecycle.

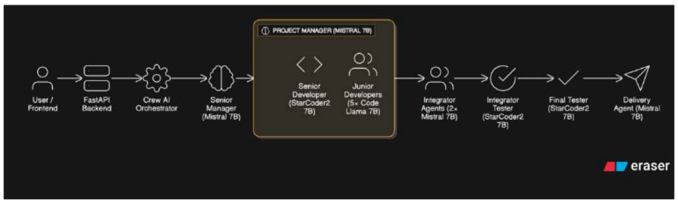

Fig. 2. Flow diagram of the Agentic AI Project Manager showing multi-agent orchestration, local LLM interaction, and automated project lifecycle.

IV. METHODOLOGY

The proposed system follows a sequential orchestration pipeline powered by CrewAI. The workflow consists of multiple coordinated stages:

1) ***User Input:*** The process begins when a user submits a project idea via the frontend interface.

2) ***Planning:*** The Senior Manager and Project Manager agents interpret the project intent, define architecture, and generate a task hierarchy.

3) ***Development:*** Developer agents autonomously produce code modules for full-stack web apps or ML workflows using LLM reasoning.

4) ***Integration:*** Integrator agents merge front-end and back-end codebases, ensuring consistency and API connectivity.

5) ***Testing:*** Testing agents evaluate generated code for logical and structural correctness.

6) ***Delivery:*** The Delivery Agent generates a packaged ZIP with all project files and a README, completing the full AI-driven lifecycle.

Communication between agents uses structured JSON-like messages that include task context, dependencies, and responses. CrewAI ensures task synchronization through message queues, maintaining consistency during orchestration

V. IMPLEMENTATION

The system is implemented using FastAPI and Python for backend logic and CrewAI as the orchestration layer. Local models (via Ollama) include:

- Mistral-7B Instruct: For high-level reasoning tasks such as project management and integration.

• StarCoder2-7B: For technical supervision, code review, and testing logic.

• CodeLlama-7B: For module-level development and helper functions.

The FastAPI service includes endpoints for assigning projects, checking system health, and monitoring execution logs. The generated code is saved in a designated /generated directory. Each generated project includes:

• Modular code files (backend, frontend, and ML logic)

• README.md with setup instructions

• A compressed ZIP archive for easy download

This approach reduces the need for manual coding, enabling developers and students to experiment with AI-driven project creation in a reproducible environment

VI. DISCUSSION

The results indicate that local orchestration of multi-agent LLMs is both feasible and efficient for end-to-end project automation. Despite hardware limitations, the modular architecture ensures stable operation with predictable latency. The integration of CrewAI with locally hosted models promotes reproducibility and data privacy, while role-based task decomposition mirrors traditional human workflows. These outcomes validate the system's potential for scalable AI-driven project management.

VII. RESULTS

The system was successfully tested for multiple full-stack applications and ML workflows (classification, regression, and data analysis tasks). The AI Project Manager autonomously generated functioning CRUD applications, API-based systems, and scikit-learn–based ML projects. The observed latency primarily arose from model loading times during initial orchestration, but overall performance remained stable under optimized timeout settings (600–1800 seconds). This validates that local model-based orchestration is feasible on consumer hardware when properly optimized.

Model	Avg.Time(s)	Memory (GB)	Success Rate (%)
Mistral 7B (Quantized)	1450	10.2	90
StarCoder2 7B	1380	9.8	85
CodeLlama 7B	1280	8.9	87

TABLE I
PROTOTYPE-LEVEL RESULTS (OBSERVATIONAL ESTIMATES)

VIII. LIMITATIONS AND FUTURE SCOPE

Currently, the system supports reproducible ML workflows using scikit-learn datasets. Custom dataset upload functionality is not yet integrated. Future versions can include:

• Dataset upload and preprocessing modules Integration with cloud LLM APIs for hybrid orchestration

• Fine-tuned local models for domain-specific code generation

• Performance optimization through parallel agent execution

Beyond its current scope, the framework can be adapted for educational use, enabling students to prototype software systems quickly, or in enterprise settings to automate routine full-stack deployments and low-code project scaffolding.

IX. CONCLUSION

The Agentic AI Project Manager demonstrates the potential of multi-agent systems for automating project management and software generation using local LLMs. It bridges reasoning, collaboration, and automation to achieve a near autonomous development workflow for web and

ML-based projects. The system lays a strong foundation for future research into distributed, agentic orchestration and localized AI-driven software engineering.

ACKNOWLEDGMENT

The author acknowledges the use of AI-assisted tools solely for improving grammar, language clarity, and document formatting. All ideas, implementation, analysis, experiments, and technical contributions presented in this paper were carried out independently by the author.

REFERENCES

[1] S. Gravitas, "AutoGPT:An autonomous gpt-4 experiment," GitHub repository,2023.[Online].Available:https://github.com/Torantulino/Auto-GPT

[2] Y. Nakajima, "BabyAGI: An autonomous AI task management system," GitHub repository, 2023. [Online] Available: https://github.com/yoheinakajima/babyagi

[3] H. Hong et al., "MetaGPT: Meta programming for multi-agent collaborative frameworks," arXiv preprint arXiv:2308.00352, 2023. [Online]. Available: https://arxiv.org/abs/2308.00352

[4] Bigcode Project, "StarCoder2: Open code generation model," Online,2024.[Online].Available:https://huggingface.co/bigcode/starcoder2

[5] M. AI, "Code Llama: Open foundation models for code," Online, 2023.[Online].Available:https://ai.meta.com/research/publications/code-llama/

[6] Mistral AI, "Mistral 7B: A dense model for high-performance reasoning,"TechnicalReport,2024.[Online].Available:https://mistral.ai/news/announcing-mistral-7b/

[7] Ollama , "Run large language models locally," Online, 2024 [Online]. Available: https://ollama.ai

[8] H. F. Team, "A practical guide to llm quantization," Online, 2024. [Online]. Available: https://huggingface.co/blog/llm-quantization

[9] H. F. T. Docs, "Memory optimization and model offloading for llms,"Online,2023.[Online].Available:https://huggingface.co/docs/transformers/main/en/perf_train_gpu_one

[10] L. Weng, "Llm-powered autonomous agents,"Online,2023. [Online].Available: https://lilianweng.github.io/posts/2023-06-23-agent/

