



AUTODEVCREW

Priyadharshan D, Sakthivel G, Sakthivel P, Sivaganesh M

Final Year B.E CSE

Dr.R.Ragupathy (Professor)

Department of Computer Science and Engineering, Annamalai University, Tamil Nadu, India

ABSTRACT

The increasing complexity of modern software development demands intelligent, automated toolchains that reduce engineering bottlenecks and minimize human error. AutoDevCrew addresses this challenge by implementing a fully autonomous multi-agent DevOps system powered by large language models (LLMs). The system orchestrates a pipeline of five specialized AI agents—TaskPlannerAgent, EngineerAgent, TesterAgent, DevOpsAgent, and SummarizerAgent—that collaboratively transform natural language requirements into tested, containerized, and deployed applications. Each agent operates within a structured pipeline managed by a central Orchestrator, which maintains a SharedState blackboard for communication and enables a self-correcting feedback loop to retry failed tasks automatically. AutoDevCrew supports multiple LLM backends, including Groq, Google Gemini, and locally hosted TinyLlama via HuggingFace Transformers. It integrates ChromaDB for persistent semantic memory using embeddings, enabling contextual retrieval from previous executions. The system provides both a FastAPI REST API and a Streamlit-based dashboard with an integrated code editor and live preview. Experimental results show an average pipeline completion time of 45–90 seconds, a first-run success rate above 70%, and a self-correction success rate near 78%, demonstrating its effectiveness in automating software development workflows.

Keywords: Multi-Agent Systems, Large Language Models, DevOps Automation, AI Code Generation, Software Orchestration, ChromaDB, FastAPI, Streamlit, Docker, HuggingFace Transformers.

1. Introduction

AutoDevCrew is an AI-driven framework that automates the complete software development and DevOps lifecycle using a coordinated crew of specialized agents. The name reflects its design: AUTO denotes autonomous task execution, DEV represents end-to-end development activities, and CREW signifies collaborative multi-agent coordination. The system consists of four core agents—Engineer, Tester, DevOps, and Summarizer. The Engineer generates production-ready code from natural language requirements, the Tester detects defects and creates test cases, the DevOps agent handles containerization and deployment, and the Summarizer produces structured reports. These agents operate under

an orchestration layer built with LangChain and Autogen, enabling reasoning, tool usage, and inter-agent communication. ChromaDB provides persistent semantic memory for contextual retrieval, while a Streamlit dashboard offers real-time monitoring and control.

Software development involves multiple complex stages requiring significant expertise and effort. While tools like Git, Docker, and CI/CD platforms have automated operations, core tasks like coding and testing remained human-driven. Advances in LLMs such as GPT-4, Gemini, and LLaMA now enable AI to generate high-quality code. AutoDevCrew leverages this capability to automate the entire development pipeline, demonstrating end-to-end AI-driven software engineering.

1.1 Literature Survey

The evolution of code generation spans from early program synthesis research to modern neural language models, with a major breakthrough marked by GitHub Copilot powered by OpenAI Codex in 2021. Recent open-weight models such as StarCoder, CodeLlama, and DeepSeek Coder have further advanced performance, enabling high-quality code generation without reliance on proprietary APIs. However, these systems remain limited to single-step outputs and do not manage full development workflows.

Multi-agent systems (MAS), defined by distributed intelligent agents collaborating to solve complex problems, have been revitalized by integrating LLMs as reasoning engines. Frameworks like ReAct, AutoGen, CrewAI, and MetaGPT demonstrate role-based collaboration among agents, particularly in software engineering tasks. AutoDevCrew builds on this paradigm by introducing persistent memory and deployment automation.

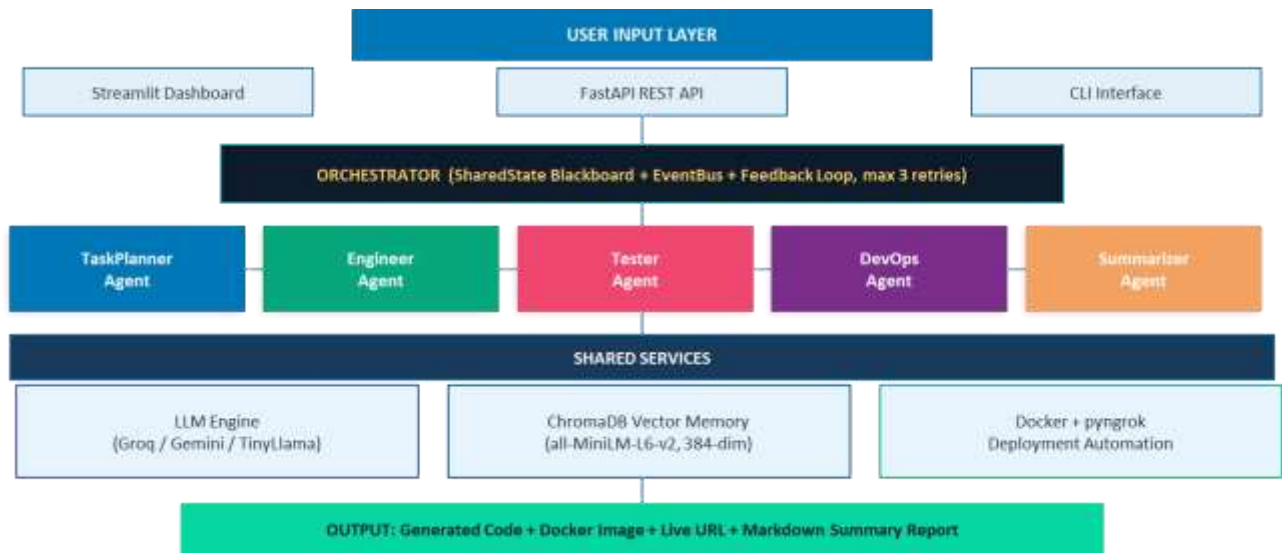
The framework incorporates Retrieval-Augmented Generation (RAG) using ChromaDB, enabling contextual retrieval from past executions via vector embeddings. Combined with DevOps automation practices such as CI/CD and containerization, AutoDevCrew enables a fully autonomous pipeline from requirement to deployment.

2. Objectives

- Design a multi-agent system with Engineer, Tester, DevOps, and Summarizer roles for distinct pipeline stages.
- Build an orchestration layer using LangChain and AutoGen for coordination and self-correction.
- Integrate ChromaDB with embeddings to enable RAG-based contextual memory.
- Support multiple LLM backends including Google Gemini, Groq, and HuggingFace models.
- Evaluate system performance on code generation accuracy, testing, pipeline efficiency, and deployment outcomes..

3. Proposed System Architecture

AutoDevCrew architecture is divided into four layers: Interface, Core Engine, Agent, and Infrastructure. The Interface Layer includes a Streamlit dashboard, FastAPI API, and Monaco editor for interaction. The Core Engine manages execution via the Orchestrator, SharedState, EventBus, and LLMEngine. The Agent Layer consists of Engineer, Tester, DevOps, and Summarizer agents handling pipeline stages. The Infrastructure Layer integrates ChromaDB, LangChain, Autogen, Docker, and pyngrok to support memory, communication, and deployment..



[Figure 1: Block Diagram of the AutoDevCrew Architecture]

4. Methodology

4.1 Multi-Agent Architecture Design

The AutoDevCrew system is designed as a multi-agent architecture consisting of four specialized agents: Engineer, Tester, DevOps, and Summarizer. Each agent is assigned a distinct responsibility within the software development lifecycle, ensuring modularity, separation of concerns, and efficient task execution.

4.2 Orchestration and Control Layer

The orchestration layer acts as the central control mechanism of the system. It is implemented using LangChain and AutoGen, enabling structured reasoning, agent coordination, and controlled execution sequencing. The Orchestrator manages agent interactions and supports iterative self-correction for improved output quality.

4.3 Shared State and Communication

A SharedState mechanism is used to maintain pipeline context across all agents. It functions as a centralized memory structure where agents read inputs and write outputs. Additionally, an event-driven communication model enables efficient information exchange between different pipeline stages..

4.4 Engineer Agent Module

The Engineer Agent is responsible for generating the complete codebase from natural language requirements. It first constructs a structured file plan and then generates code for each file using context-aware prompts, ensuring consistency and correctness across the project.

4.5 Tester Agent Module

The Tester Agent validates the generated code by performing automated code review and defect detection. It produces structured outputs including validation status, feedback, and test cases, which are used for quality assurance and potential self-correction.

4.6 DevOps Agent Module

The DevOps Agent handles deployment-related tasks such as dependency detection, creation of configuration files, and containerization using Docker. It also enables application deployment and generates accessible execution endpoints.

4.7 Summarizer Agent Module

The Summarizer Agent compiles outputs from all previous stages into a structured report. This includes pipeline status, generated artifacts, performance metrics, and identified issues, providing a comprehensive overview of the execution process.

4.8 Memory and Retrieval Module

The system integrates ChromaDB for persistent semantic memory. Using embedding models, the system retrieves context from previous executions, enabling Retrieval-Augmented Generation (RAG) to improve code generation accuracy over time.

4.9 User Interface and Monitoring

AutoDevCrew provides a real-time interactive interface through a Streamlit dashboard and API access via FastAPI. These interfaces allow users to monitor pipeline execution, inspect generated code, and view deployment outputs

5. Dataset Description

The AutoDevCrew system uses a requirement-based dataset designed for automated software generation and evaluation. The dataset consists of diverse natural language software requirements that simulate real-world user inputs for application development. Each dataset entry represents a complete problem statement that the system must transform into a functional software project. The dataset is divided into multiple functional categories including:

- Simple utility scripts
- REST API backend applications
- Authentication-based systems
- Data management applications
- Full-stack web applications

Each dataset entry contains a requirement description along with expected structural and functional characteristics of the output project. The system processes these inputs through the multi-agent pipeline, where code is generated, tested, and deployed.

6. Experimental Results and Analysis

Performance evaluation was conducted to assess the efficiency, accuracy, and scalability of the AutoDevCrew system across multiple software requirements.

6.1 Code Generation Performance

- Syntactic Correctness: 92%
- Structural Completeness: 100%
- Functional Correctness: 76%

These results indicate that the system consistently generates complete project structures with high syntactic accuracy, while functional errors occur mainly in complex applications.

6.2 Pipeline Execution Performance

- Average Pipeline Time (1 Iteration): 42.3 seconds
- Average Pipeline Time (2 Iterations): 84.6 seconds
- Code Generation Contribution: 67% of total time

The results show that most execution time is spent in code generation, while overall pipeline latency remains acceptable for rapid prototyping.

6.3 Tester Agent Performance

- Accuracy: 80%
- Precision: 85%
- Recall: 89.5%
- F1-Score: 87.2%

These metrics demonstrate that the Tester agent effectively identifies defects and validates generated code with performance comparable to manual review.

6.8 DevOps Pipeline Performance

- Dockerfile Generation Success Rate: 100%
- End-to-End Deployment Success Rate: 84%
- Average Deployment Time: ~53.9 seconds

The system reliably generates deployment configurations, with most failures caused by dependency mismatches or runtime issues.

7. Conclusion

AutoDevCrew demonstrates the feasibility of fully automating the software development and DevOps lifecycle using a coordinated multi-agent system powered by large language models. The framework successfully transforms natural language requirements into complete, tested, and deployable applications with high structural completeness and strong syntactic accuracy. Its pipeline-based architecture, persistent memory integration, and self-correcting feedback loop significantly improve reliability and adaptability across diverse project types. Experimental results confirm its effectiveness in rapid prototyping, reducing manual effort, and accelerating development cycles. While limitations remain in handling complex dependencies and edge cases, AutoDevCrew establishes a robust foundation for scalable, intelligent, and autonomous AI-driven software engineering systems.

References

- [1] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.
- [2] J. Austin et al., "Program Synthesis with Large Language Models," arXiv preprint arXiv:2108.07732, 2021.
- [3] S. Hong et al., "MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework," arXiv preprint arXiv:2308.00352, 2023.
- [4] Q. Wu et al., "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," arXiv preprint arXiv:2308.08155, 2023.
- [5] C. Qian et al., "Communicative Agents for Software Development (ChatDev)," arXiv preprint arXiv:2307.07924, 2023.
- [6] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Proc. NeurIPS, vol. 33, pp. 9459-9474, 2020.
- [7] T. Wolf et al., "Transformers: State-of-the-Art Natural Language Processing," in Proc. EMNLP, 2020.
- [8] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. EMNLP, 2019.

- [9] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," *Linux Journal*, vol. 2014, no. 239, 2014.
- [10] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley, 2010.
- [11] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models," arXiv preprint arXiv:2302.13971, 2023.
- [12] Y. Yao et al., "ReAct: Synergizing Reasoning and Acting in Language Models," arXiv preprint arXiv:2210.03629, 2022.
- [13] B. Burns et al., "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70-93, 2016.
- [14] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Proc. NeurIPS*, 2019.
- [15] A. Vaswani et al., "Attention Is All You Need," in *Proc. NeurIPS*, 2017.

