IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.15572

Coder Buddy – An Agentic AI-Powered Python Development Assistant for Code Review, Code Generation, and Automated Mini Project Building

Sonika D[1], Harshitha L[2]

Department of MCA, BIT, K.R. Road, V.V. Puram, Bangalore, India[1]

Assistant Professor, Department of MCA, BIT, K.R. Road, V.V. Puram, Bangalore, India[2]

Abstract: The growing demand for intelligent programming assistants has led to the development of agentic AI systems that can autonomously plan and execute multi-step development tasks. This paper presents Coder Buddy, an agentic AI application designed to assist Python developers with three core capabilities: automated code review, prompt-based code generation, and complete mini project building. The system combines rule-based static analysis with Large Language Models (LLMs) to deliver accurate, context-aware feedback and generate production-ready Python code. A key feature of the system is its ability to auto-detect user intent and dynamically generate either CLI or GUI (Streamlit-based) projects with proper folder structure, dependency files, and runnable output. The application is built using Python and Streamlit, and maintains session-based history for all interactions. Evaluation across simulated development scenarios demonstrates the system's effectiveness in improving code quality, reducing development time, and supporting learning and rapid prototyping.

Keywords: Agentic AI, Code Review, Code Generation, Large Language Models, Python, Streamlit, AST Analysis, Mini Project Builder, Automated Software Development

I. INTRODUCTION

The rapid advancement of artificial intelligence has introduced new possibilities in software development, enabling intelligent systems that can assist developers throughout the coding process. Python, being one of the most widely used programming languages for education, data science, and rapid prototyping, demands robust tools that support developers at various skill levels. However, most existing solutions operate in isolation, offering either rule-based feedback or AI-generated suggestions, but rarely combining both within a unified platform. Traditional code review processes are time-consuming and dependent on human expertise, while generating well-structured Python projects from scratch requires significant effort and domain knowledge. There is therefore a growing need for an integrated system that can intelligently understand user intent, perform static code analysis, generate executable code, and automatically build complete project structures.

1.1 Project Description

Coder Buddy is an agentic AI-powered Python development assistant built using Python and Streamlit. The system assists developers across three core modes: **code review**, where it combines AST-based static analysis with LLM-powered feedback to evaluate and improve code quality; **code generation**, where users describe requirements in natural language and receive executable Python code; and **mini project building**, where the system auto-detects user intent and generates a complete CLI or GUI project structure with folder hierarchy, source files, and a requirements.txt, packaged as a downloadable ZIP. A session-based history tracker maintains records of all past interactions, making the platform ideal for learners, academic projects, and rapid prototyping.

1.2 Motivation

Beginners and students often struggle to get timely, meaningful feedback on their Python code and lack guidance to structure complete projects efficiently. Existing tools are either too limited or too complex to use effectively. This motivated the development of **Coder Buddy** — a unified agentic AI system that makes quality code development accessible and faster for learners and developers alike.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.15572

II. RELATED WORK

Paper [1] " *AI-Powered Code Review Systems Using Large Language Models"* This paper explores the use of LLMs for automated code review, demonstrating that AI-based systems significantly outperform traditional static analysis tools in detecting logical errors. However, the system lacks integration with code generation or project building capabilities.

Paper [2] *Static Code Analysis Using Abstract Syntax Trees for Python Applications"* This work presents an AST-based approach for detecting code quality issues such as unused variables and structural inefficiencies in Python programs. The study does not incorporate AI-driven feedback, leaving a gap for intelligent, context-aware suggestions

Paper [3] *"Prompt-Based Code Generation Using Transformer Models"* The authors investigate the effectiveness of transformer-based LLMs in generating syntactically correct and executable Python code from natural language prompts. The system, however, does not support automated project scaffolding or folder structure generation.

Paper [4] *"Agentic AI Systems for Automated Software Development Tasks"* This paper introduces agentic AI systems that autonomously plan and execute multi-step tasks to complete complex software development objectives. The proposed framework lacks a user-friendly interface suitable for beginners and students.

Paper [5] *"Streamlit-Based Interactive AI Applications for Developer Productivity"* This study examines the use of Streamlit as a rapid development framework for building interactive AI-powered developer tools. The work focuses primarily on data science use cases and does not address code review or automated project generation workflows.

III. METHODOLOGY

A. System Architecture

Coder Buddy follows a **three-tier architecture** consisting of a **presentation layer**, **application layer**, and **data layer**. The presentation layer provides an interactive web-based interface built using Streamlit, allowing users to select modes and interact with the system. The application layer hosts the core business logic including the agentic AI pipeline, static analysis engine, LLM integration, and project generation modules. The data layer manages session-based history records of all code reviews, generations, and project builds

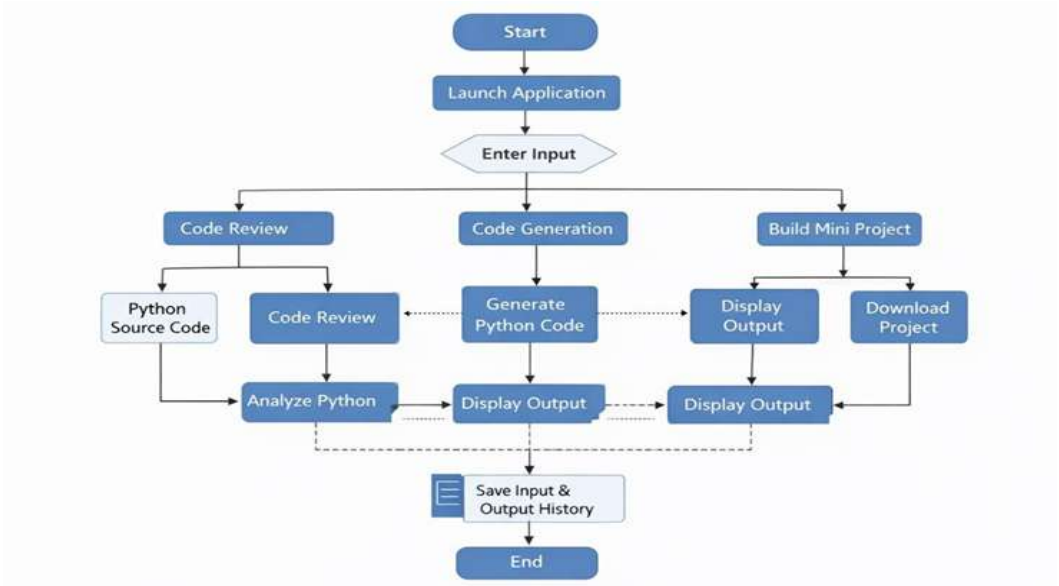

Fig. 1. Flowchart of methodology

B. Code Review Module

Donors (restaurants, events, households) register surplus food through the web interface by providing details such as food type, quantity, pickup location, and expiry time. The system stores this information securely in the database and forwards alerts to nearby NGOs. The module ensures that all donation records are complete, valid, and verified by administrators before being listed for pickup.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.15572

C. Code Generation Module

The code generation module accepts natural language prompts from the user and generates syntactically valid, executable Python code using LLMs. A Python-only validation check ensures that the generated output is strictly Python-based, maintaining safety and consistency. Users can optionally request a step-by-step explanation of the generated code alongside the output

D. Mini Project Builder Module

The mini project builder is the most distinctive component of Coder Buddy. The system analyzes the user's natural language description and automatically detects whether a CLI or GUI (Streamlit-based) application is required. Based on this intent detection, the system generates a complete project structure including folder hierarchy, source files, and a requirements.txt file, all packaged as a downloadable ZIP file ready for immediate use.

E. Agentic AI Pipeline

The agentic AI pipeline forms the intelligence core of the system. Upon receiving user input, the pipeline autonomously plans the required steps, selects appropriate tools and modules, executes the tasks in sequence, and delivers the final output. This multi-step autonomous planning distinguishes Coder Buddy from traditional single-step AI assistants and enables it to handle complex, multi-faceted development tasks effectively.

F. Implementation Flow

1. User selects a mode — Review, Generate, or Build Project..
2. System analyses user input and detects intent using the agentic pipeline
3. Appropriate module is triggered — static analysis, LLM generation, or project builder.
4. Appropriate module is triggered — static analysis, LLM generation, or project builder**.**
5. All interactions are logged into the session-based history tracker for future reference.

G. Hardware and Software Components

The system requires a standard computing platform with a minimum Intel Core i5 processor, 8 GB RAM, and stable internet connectivity for LLM API access. Software components include Python for backend development, Streamlit for the web interface, AST libraries for static code analysis, and LLM APIs for intelligent code feedback and generation.

.

IV. SYSTEM IMPLEMENTATION AND EVALUATION

A. System Architecture and Deployment

Coder Buddy is implemented as a web-based application using Python and Streamlit, following a three-tier client-server architecture. The frontend provides an interactive and responsive interface where users can select their desired mode — code review, code generation, or mini project building — and interact with the system in real time. The backend handles all core business logic including static code analysis, LLM API communication, intent detection, and project generation through secure and modular Python scripts.

B. Code Review Implementation

The code review module is implemented using Python's built-in AST (Abstract Syntax Tree) library, which parses user-submitted Python code without executing it. The parser scans for rule-based quality violations such as unused variables, missing function docstrings, improper naming conventions, and overly complex functions. The parsed code is then forwarded to the LLM API, which generates intelligent, context-aware feedback and provides rewritten code suggestions. The combined output is displayed to the user in a structured and readable format on the Streamlit interface.

C. Code Generation Implementation

The code generation module accepts natural language descriptions from the user through the Streamlit input field. The input is passed directly to the LLM with a structured prompt that enforces Python-only output and optionally requests a step-by-step explanation of the generated code. A validation layer checks the returned output to confirm it is valid Python before presenting it to the user, ensuring safety and correctness of all generated programs.

D. Mini Project Builder Implementation

The mini project builder is the most distinctive module of the system. Upon receiving a project description from the user, the agentic pipeline analyzes the input to detect whether the user requires a CLI or a GUI (Streamlit-based) application. Based on this detection, the system dynamically generates a complete project folder structure including all necessary source files such as main.py or app.py, supporting modules, and a requirements.txt file listing all dependencies. The

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.15572

complete project is packaged into a downloadable ZIP file, allowing users to immediately run the generated project on their local machines.

E. History Tracking Implementation

A session-based history tracking module is integrated across all three modes of the system. Every code review, generated program, and built project is automatically logged within the current session, allowing users to revisit, compare, and reuse past outputs without re-entering their queries. This feature significantly enhances usability for iterative development and learning workflows.

F. Performance Evaluation Metrics

System performance is evaluated using the following key metrics:

- **Code Review Accuracy** — Percentage of actual code quality issues correctly identified by the AST and LLM pipeline
- **Code Generation Success Rate** — Percentage of generated programs that are syntactically valid and directly executable
- **Project Build Accuracy** — Correctness of CLI vs GUI intent detection and completeness of the generated project structure
- **Response Time** — Time taken from user input submission to final output delivery
- **User Satisfaction** — Usability and output quality rated through simulated user testing scenarios

G. Experimental Setup and Results Analysis

The system is tested using simulated development scenarios involving multiple types of Python code submissions and project descriptions. Various code samples with intentional quality issues were submitted to the review module to evaluate detection accuracy. Diverse natural language prompts were used to test the generation module across different programming tasks. Project builder tests covered a range of CLI and GUI project descriptions to evaluate intent detection accuracy and structural completeness of the generated output.

G. Results and Observations

1.System Home / Dashboard
2.Code Review
3.Code Generator
4. Mini Project Builder

FIG. 2. SYSTEM HOME / DASHBOARD

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.15572

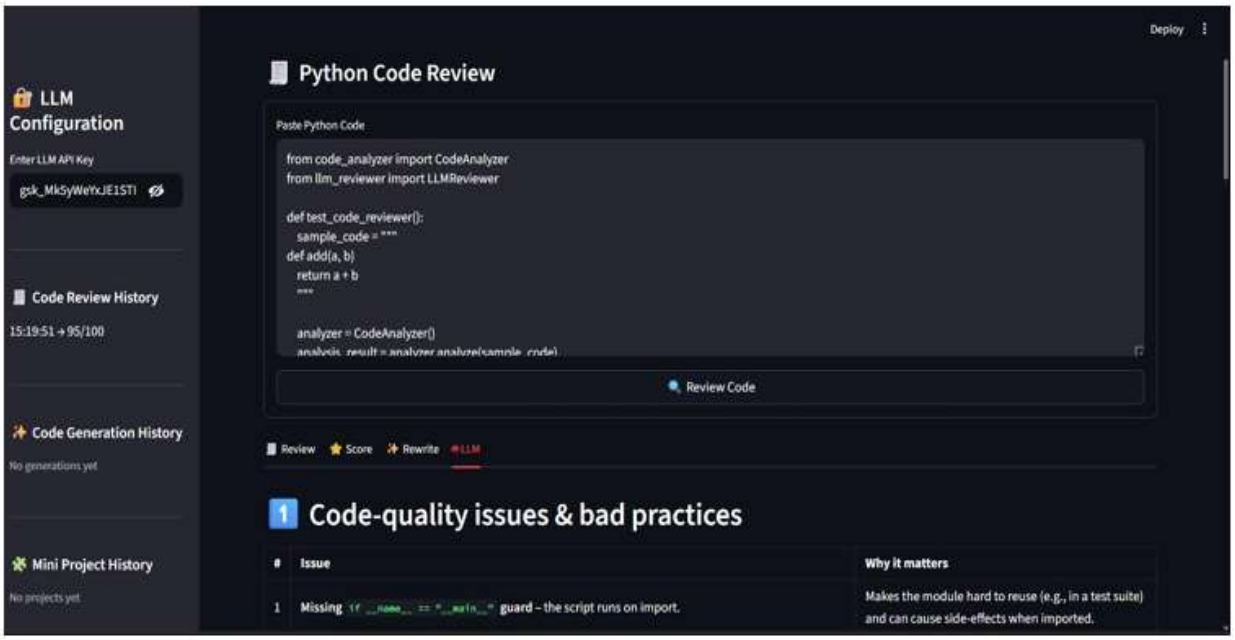

FIG. 3. SYSTEM HOME / DASHBOARD

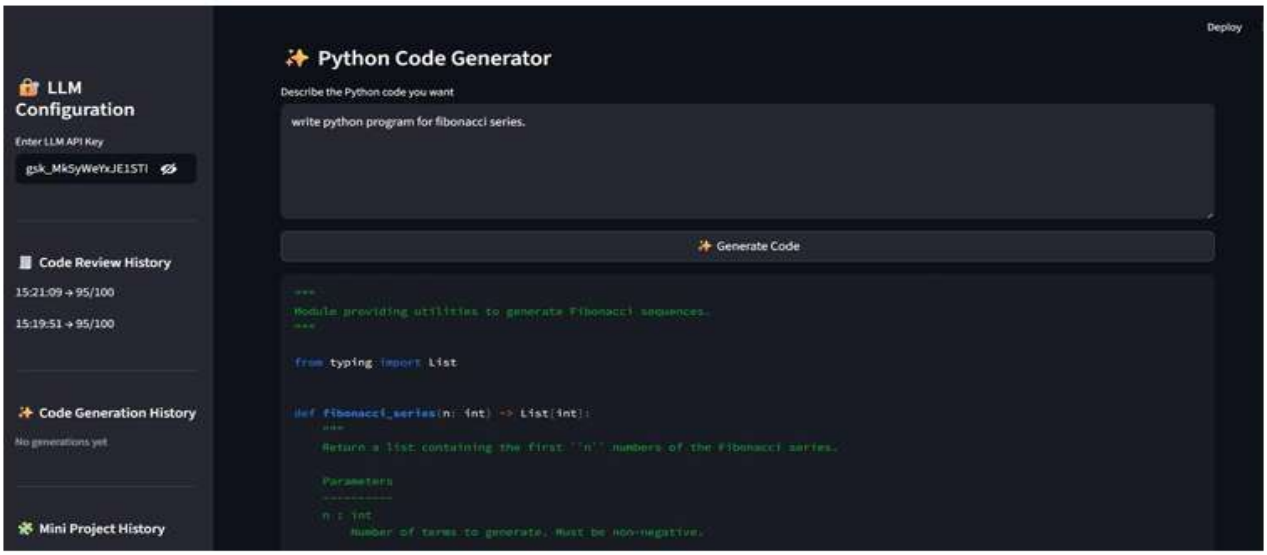

Fig. 4. Code Generator

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026

DOI: 10.17148/IJARCCE.2026.15572

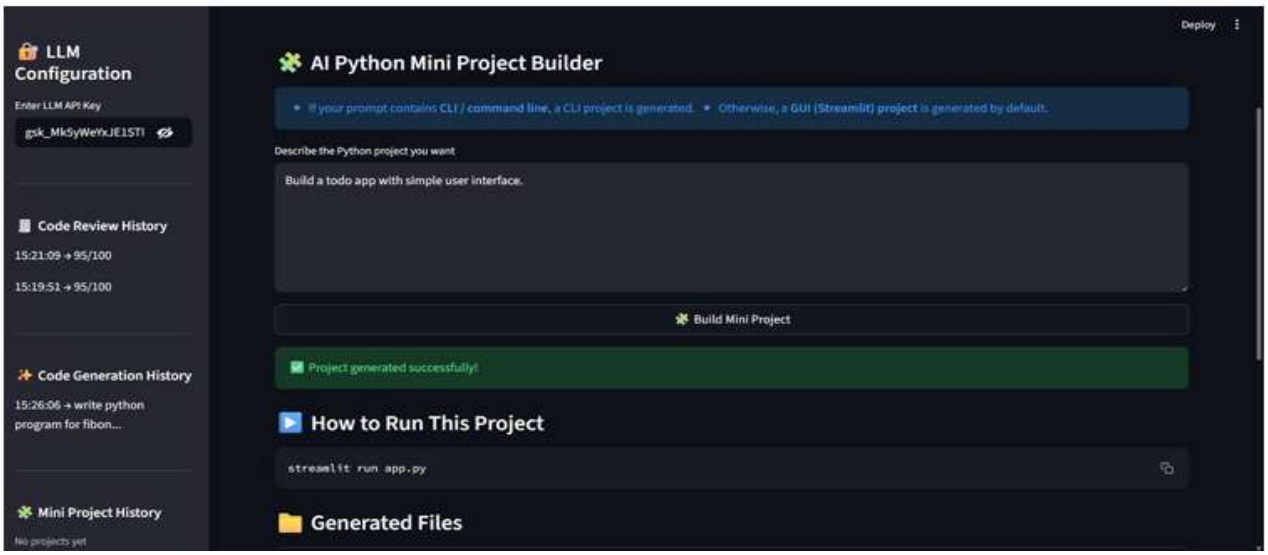

Fig. 5. Mini Project Builder

V. RESULTS AND DISCUSSION

The experimental evaluation demonstrates that the proposed Coder Buddy system effectively automates software development tasks using Agentic AI techniques. The multi-agent architecture improves task coordination, modular development, and project generation efficiency.

The Planner Agent successfully interprets user requirements and generates structured project plans. The Architect Agent improves software organization by creating modular engineering tasks, while the Coder Agent generates executable source code efficiently.

Compared to traditional coding approaches, the proposed system reduces development time, improves productivity, and simplifies project creation for developers and students. The integration of LangGraph enables efficient communication between agents and supports scalable workflow management.

VI. CONCLUSION

This paper presented Coder Buddy, an AI-powered multi-agent coding assistant developed using Agentic AI and LangGraph architecture to automate software development tasks. The system integrates Planner, Architect, and Coder agents that collaboratively analyze user requirements, generate project plans, create modular workflows, and produce executable source code automatically. By enabling intelligent task coordination and automated project generation through natural language interaction, the proposed system reduces manual coding effort, improves development efficiency, and enhances productivity for developers and students. Experimental evaluation demonstrates that the multi-agent workflow provides scalable, reliable, and efficient software automation, highlighting the potential of Agentic AI in transforming modern software engineering practices

VII. FUTURE WORK

Future enhancements to the Coder Buddy system can focus on integrating advanced Large Language Models to improve code quality, reasoning capabilities, and project accuracy. Additional AI agents can be introduced for automated debugging, testing, security analysis, and performance optimization to support a complete software development lifecycle. The platform can also be extended with voice-based interaction, real-time collaboration features, and cloud deployment support for generated applications. Integration with GitHub and DevOps tools can enable automated version control and continuous deployment workflows. Furthermore, support for mobile application development, multilingual project generation, and AI-powered documentation generation can make the system more scalable, intelligent, and accessible for a wider range of users and software engineering applications.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 5, May 2026
DOI: 10.17148/IJARCCE.2026.15572

REFERENCES

[1]. **Paper [1]:** *AI Coding Agents in Software Engineering* — **H. Li et al.**
Discusses the use of AI agents for automating software development and code generation tasks.
[2]. **Paper [2]:** *Where Do AI Coding Agents Fail?* — **R. Ehsani et al.**
Analyzes the limitations and challenges faced by autonomous AI coding systems.
[3]. **Paper [3]:** *Adoption of Coding Agents on GitHub* — **R. Robbes et al.**
Explains how AI coding agents are integrated into GitHub-based software engineering workflows.
[4]. **Paper [4]:** *Configuring Agentic AI Coding Tools* — **M. Galster et al.**
Highlights methods for improving workflow orchestration and configuration in Agentic AI systems.
[5]. **Paper [5]:** *LangGraph-Based Multi-Agent Workflow Systems* — **LangGraph Research Team**
Presents scalable multi-agent workflow architectures using LangGraph frameworks.
[6]. **Paper [6]:** *Artificial Intelligence in Software Automation* — **P. Singh and R. Kaur**
Discusses how AI improves software automation and reduces manual coding effort.
[7]. **Paper [7]:** *Conversational Interfaces for Intelligent Development Systems* — **S. B. Lee and J. Kim**
Explains the use of conversational AI interfaces in software engineering applications.
[8]. **Paper [8]:** *Cloud-Based AI Platforms for Automated Development* — **M. Hassan et al.**
Describes cloud-based platforms that support intelligent software project generation and deployment.
[9]. **Paper [9]:** *Agent-Based Workflow Systems for Intelligent Applications* — **A. Kumar and S. Iyer**
Focuses on task-planning and workflow coordination using autonomous AI agents.
[10]. **Paper [10]:** *Generative AI for Automated Debugging and Testing* — **T. Nguyen and H. Tran**
Highlights the application of Generative AI techniques in debugging, testing, and code optimization.
[11]. **Paper [11]:** *Large Language Models for Software Development*—**OpenAI Research Team**
Discusses how Large Language Models assist in intelligent code generation and software development
[12]. **Paper [12]:** *Multi-Agent Collaboration Frameworks Using AI* — **S. Verma and N. Rao**
Explains collaborative AI agent frameworks for managing complex software engineering workflows.

