International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Multi-Agent Autonomous Coding System

Muhammad Saadullah[1], Abdul Hai[2], Irshad Ahmad Kumar[3], Dr. Ch Ramesh Babu[4], Ms. N Musrat Sultana[5], Mr. G Nagi Reddy[6]

[1,2,3]Student, Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad, Telangana, India.
[4,5,6]Associate Professor, Department of Computer Science and Engineering, Mahatma Gandhi Institute of Technology, Gandipet, Hyderabad, Telangana, India.

Abstract
AutoDev is a multi-agent autonomous coding system designed to transform natural language prompts into executable software projects through a structured workflow. Instead of stopping at code generation, the system continues through prompt refinement, specification extraction, planning, execution, error handling, and review. A browser-based interface is paired with a FastAPI backend and a LangGraph-based workflow engine so that the complete development lifecycle can be observed and controlled in one place. The design also includes session management, persistent state, and Retrieval-Augmented Generation support so that previous context can be reused during refinement.
This arrangement is intended to improve reliability in multi-step programming tasks such as concurrency, data processing, networking, and backend development. Although related autonomous coding systems already exist, AutoDev is presented as an integrated workflow implementation that emphasizes execution feedback, review, and continuity across sessions. Evaluation on benchmark tasks shows that the approach improves functional correctness, robustness, and error recovery, while accepting a higher execution time than a single-pass baseline. The system is therefore positioned as both a practical coding assistant and a research platform for workflow-based autonomous software engineering.

Keywords: Multi-agent systems, Code generation, Large language models, Autonomous software engineering, Workflow systems, Program synthesis

1. Introduction

Software development still depends heavily on repetitive cycles of coding, testing, debugging, and revision. Large Language Models have made code generation easier, but in practice they are still used mainly as assistive tools rather than fully autonomous systems. Many current workflows require a human to manually coordinate planning, execution, validation, and correction, which limits their usefulness for longer tasks that extend beyond a single prompt-response interaction.
The problem is not only code quality, but also process continuity. In the absence of a structured workflow, generated code may remain untested, runtime errors may be missed, and important context from earlier attempts may be lost. This makes it difficult to support end-to-end development tasks in a consistent way. In addition, different systems often store state, outputs, and user feedback in disconnected ways, which weakens traceability and reduces the ability to learn from prior attempts.

AutoDev addresses this gap by organizing software generation as a multi-stage pipeline. The workflow decomposes a task into prompt refinement, specification extraction, planning, code generation, execution, error handling, and review. Rather than claiming that autonomous coding itself is new, the contribution is the integration of these parts into a single browser-based environment with persistent memory, execution-aware feedback, and session continuity.

1.1 Problem Definition

The software development process remains repetitive and time-consuming, requiring developers to manually write, test, and debug code. While Large Language Models assist with code generation, they still act as passive tools without the ability to execute or validate their output. This limits their effectiveness in handling complex, multi-step development tasks. Existing systems also lack full autonomy and long-term learning capabilities. They are unable to correct their own mistakes, retain knowledge from previous attempts, or reuse past solutions for new problems. As a result, development remains dependent on continuous human intervention. Therefore, there is a need for an intelligent system that can autonomously generate, evaluate, and refine code while improving over time through accumulated experience.

1.2 Existing System

In the current software development environment, developers manually perform repetitive cycles of coding, testing, and debugging to complete tasks. Large Language Models provide useful suggestions but function mainly as passive co-pilots without the ability to execute or validate code. Error handling in existing systems remains manual, requiring developers to identify and fix issues themselves. Additionally, most tools lack persistent memory, preventing them from learning from previous mistakes or reusing past solutions. As a result, the development process continues to be time-consuming and heavily dependent on human effort. These limitations highlight the need for more autonomous and self-improving systems.

1.3 Proposed System

The proposed system, AutoDev, is a browser-based autonomous coding system designed to automate code generation, execution, and refinement through a structured multi-stage workflow. Instead of relying on isolated agents, the system follows a pipeline consisting of stages such as prompt refinement, specification extraction, planning, code generation, execution, error handling, and review. These stages operate in a continuous generate-execute-debug loop, enabling automatic error detection and correction. The system also incorporates session memory and structured state management, allowing it to retain previous context and improve over time. A Retrieval-Augmented Generation mechanism is used to support better reasoning and decision-making by leveraging stored knowledge and past outputs. This reduces ambiguity and improves the accuracy of generated solutions. By minimizing manual intervention and providing a complete development environment with execution and visualization, AutoDev improves efficiency and enables a more autonomous and self-improving approach to software development.

1.4 Technical Specifications

1.4.1 Hardware Requirements

Component	Requirement
Processor	Quad-core CPU, 2.5 GHz or higher
RAM	8 GB, 16 GB recommended for local LLM

Component	Requirement
	hosting
Storage	10 GB free disk space, SSD recommended
Network	Stable broadband connection, 10 Mbps upload/download minimum

1.4.2 Software Requirements

Component	Requirement
Operating System	Microsoft Windows 10 or later
Programming Language	Python 3.9 or later
IDE / Text Editor	VS Code
Frameworks / Tools	LangGraph, FastAPI
Database	Qdrant Vector Database
Python Libraries	pandas, numpy, scikit-learn, qdrant-client, LLM SDKs such as Gemini, Groq, and Ollama
Browser	Google Chrome or Mozilla Firefox

2. Literature Review

Research on autonomous coding systems and LLM-based software engineering agents has accelerated in recent years. The works reviewed here focus on multi-agent coordination, structured workflows, code generation, execution feedback, memory, and iterative refinement. Together they show that agentic systems are most useful when reasoning, execution, and validation are not treated as separate manual steps.

The reviewed literature also indicates a recurring trade-off. Systems that are fast and simple may produce useful code quickly, but they often struggle with runtime validation, error recovery, and state continuity. Systems that add planning, retry loops, memory, and execution feedback are usually slower, but they are more reliable when the task requires several dependent steps.

2.1 Literature Survey Table of AutoDev

S. no	Study	Year	Methodology	Relevance to AutoDev
1	Cai et al.	2025	Analyzed design patterns in LLM-based multi-agent systems across 94 papers.	Supports structured workflows and role-based coordination.
2	Du et al.	2025	Classified optimization methods for LLM agents into parameter-driven and parameter-free approaches.	Informs planning and interaction efficiency.
3	Aratchige and Ilmini	2025	Surveyed architecture, memory, and planning in effective multi-agent systems.	Supports the layered orchestration design.
4	Li et al.	2025	Evaluated autonomous coding agents with the AIDev dataset.	Shows that speed alone does not guarantee quality.
5	Yang et al.	2025	Proposed SWE-smith for scaling data used to train software engineering agents.	Highlights the role of data quality and scale.

S. no	Study	Year	Methodology	Relevance to AutoDev
6	Dong et al.	2025	Reviewed code generation with LLM-based agents across the software lifecycle.	Aligns with end-to-end automation.
7	He et al.	2025	Studied plan-then-execute workflows and their effect on trust and performance.	Supports the planning-first pipeline.
8	Wang et al.	2024	Presented OpenHands as an open platform for AI software developers.	Shows the value of integrated development environments.
9	Yang et al.	2024	Introduced SWE-agent and agent-computer interfaces for automated software engineering.	Supports code navigation and execution.
10	Madaan and Shinn	2023	Presented Self-Refine and Reflexion for iterative improvement.	Directly aligns with feedback-driven correction.

The survey indicates that AutoDev is best viewed as a workflow integration effort, where the emphasis is placed on executable output, refinement, and continuity rather than on a single-pass code suggestion.

3. Design Methodology

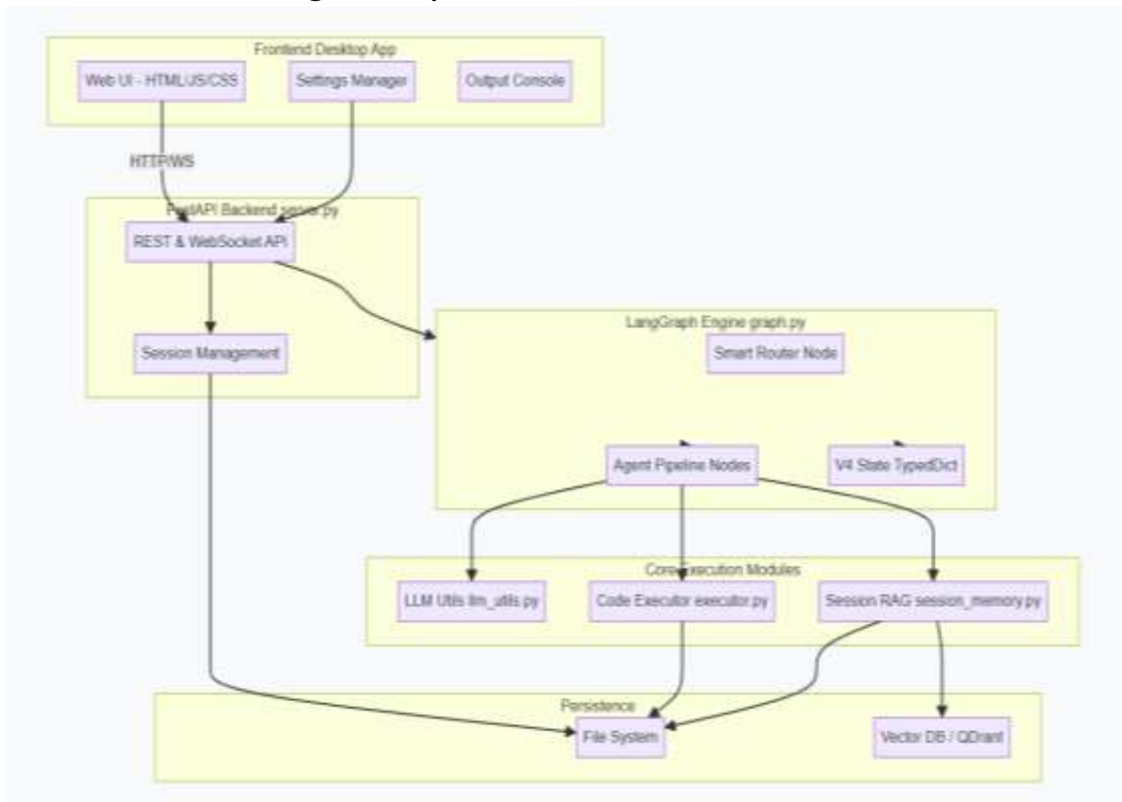
The design of AutoDev is centered on a modular client-server architecture with persistent state, execution feedback, and graph-based workflow orchestration. The objective is to move a user prompt through a sequence of well-defined stages until a valid and reviewed solution is obtained.

3.1 System Architecture of AutoDev

The architecture follows a layered approach. The presentation layer contains the browser-based user interface, where tasks are entered and results are viewed. The application layer contains the workflow logic, authentication, task routing, test execution, review handling, and session management. The data layer stores user records, generated files, execution results, retrieval context, and analytics information. Together, these layers support organized communication and make it easier to extend individual parts of the system without changing the entire application.

The frontend is implemented with HTML, CSS, and JavaScript. It provides a web interface with a prompt area, settings controls, session controls, and output panels. The backend uses FastAPI to expose REST and WebSocket endpoints. LangGraph is used to define the workflow nodes and transitions, while Qdrant stores retrievable context for sessions and repeated tasks. This separation allows AutoDev to remain modular while still supporting real-time updates.

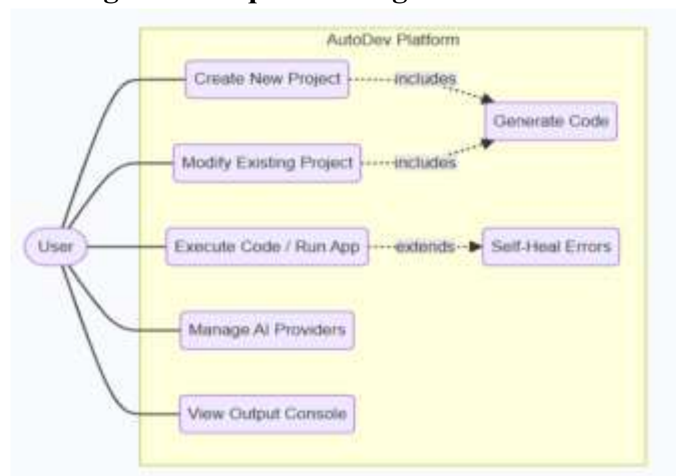
Figure 1: System Architecture of AutoDev



3.2 Use Case and Interaction Overview

The user submits a task prompt through the interface and selects the execution mode and related settings. The system then refines the prompt, builds a specification, generates a plan, installs any required dependencies, produces code, runs the code, and reviews the outcome. If an error occurs, the workflow is redirected through a correction cycle. This interaction pattern is consistent with the report's emphasis on real-time monitoring and follow-up handling.

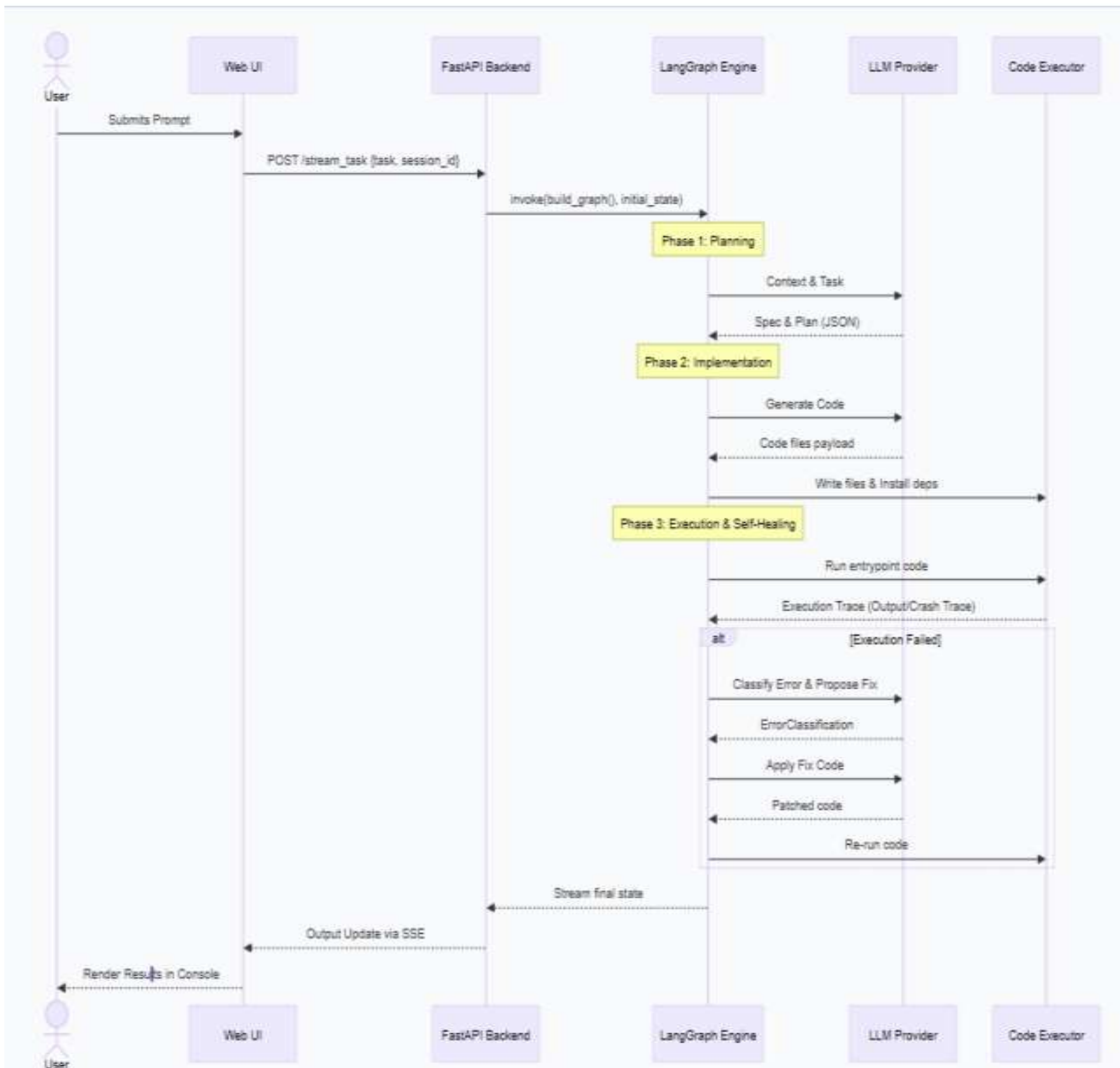
Figure 2: Sequence Diagram of AutoDev



3.3 Sequence Diagram of AutoDev

The sequence view captures the order of interaction between the user interface, the backend API, the workflow engine, the execution module, and the persistence layer. A request is received first, then the task is placed into the workflow, the code is generated, execution results are collected, and feedback is returned to the interface. The sequence is repeated when refinement is required.

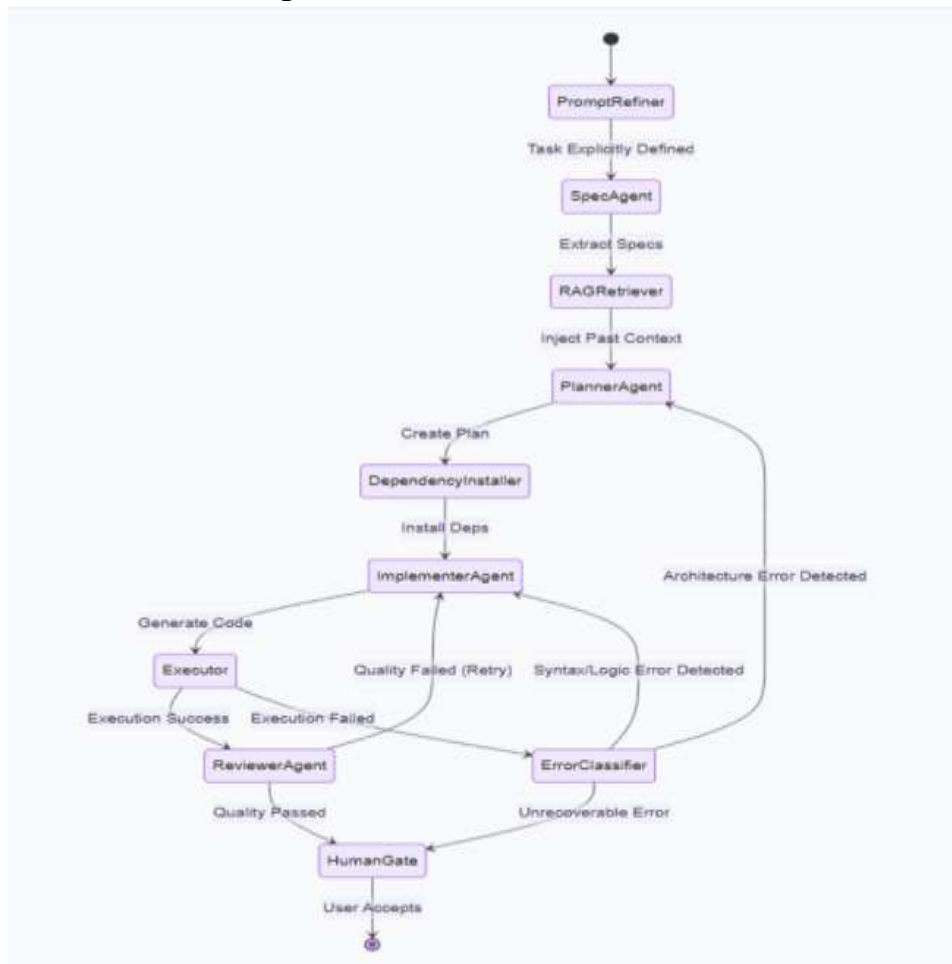
Figure 3: Sequence Diagram of AutoDev



3.4 Process Flow

The process flow begins with prompt submission and continues through refinement, specification extraction, planning, generation, execution, error analysis, and review. When the output fails validation, the workflow loops back to earlier stages so that a corrected solution can be produced. The process therefore behaves as a generate-execute-debug cycle rather than a one-step generator.

Figure 4: Process Flow of AutoDev



4. Implementation

The implementation of AutoDev combines a browser-based frontend, a Python backend, a LangGraph workflow engine, execution tooling, and persistent storage. The system is designed so that each stage can be inspected and repeated without losing context.

4.1 Frontend Components

The frontend is responsible for prompt entry, task selection, settings configuration, session viewing, and output inspection. It presents workflow stages in a visible sequence so that the user can follow how the request is transformed. A floating console and multiple result panels are used to show execution logs, generated files, project specifications, and validation output.

4.2 Backend Components

The backend exposes request handlers for task execution, session loading, session retrieval, and streaming updates. REST endpoints are used for standard requests, while Server-Sent Events or WebSocket style streaming is used for real-time progress reporting. The backend also stores session identifiers and routes the workflow state between stages.

4.3 Workflow Engine

The workflow engine is implemented as a state-driven graph. A shared state object stores task details, refined prompts, specifications, plans, generated files, dependency information, execution errors, retry status, and history. Each node in the graph represents one stage such as prompt refinement, planning,

code generation, execution, review, or recovery. The graph is then invoked with the initial state so that the pipeline runs in a controlled order.

4.4 Code Execution Module

Generated code is executed in a controlled subprocess so that output and errors can be captured safely. This execution layer is important because the system is not limited to code synthesis. It checks whether the generated program actually runs and whether its output satisfies the expected conditions. Syntax errors, runtime errors, and logic errors are handled differently, which allows the system to choose a better corrective action.

4.5 Session Memory and RAG

Session memory is used to preserve the current task state and to allow users to resume previous work without starting over. Retrieval-Augmented Generation is used to search stored embeddings and fetch relevant past context. This improves continuity across retries and helps the system reuse information from earlier attempts, which is particularly useful when a task spans several iterations.

4.6 Runtime Settings and Provider Management

Runtime settings allow the user to choose the execution mode, provider, retry depth, and workflow behavior. The system can invoke external LLM providers such as Gemini, Groq, or Ollama depending on the configured setup. This makes it possible to compare providers while retaining the same workflow structure.

4.7 Output and File Handling

Generated files are saved in the workspace and returned to the user for inspection and download. The output layer is responsible for writing files to disk, preserving file structure, and showing previews in the interface. This keeps the execution cycle connected to the visible results.

5. Testing and Results

The system was benchmarked against eight programming tasks covering concurrency, algorithms, databases, networking, data processing, software architecture, full stack development, and error handling. Each task used ten test cases, including three edge cases, so that both ordinary and boundary behavior could be evaluated. The same model configuration and execution environment were used for both the baseline and AutoDev systems so that the comparison remained controlled.

5.1 Test Setup

The experiments were run using Anthropic Claude Opus 4.7 through the API, with a fixed system prompt and consistent runtime configuration. The local environment used Python 3.10, isolated subprocess execution, and dedicated workspace directories for each task. Evaluation was based on the outputs generated by the test harness, which reported pass or fail status for each case.

5.2 Benchmark Tasks and Aggregate Metrics

Table 1: Comparison of Baseline LLM and AutoDev Across Benchmark Tasks

Task	Domain	Baseline (Tests / Edge)	AutoDev (Tests / Edge)
t1	Concurrency	7/10, 2/3	10/10, 3/3
t2	Algorithms	10/10, 3/3	10/10, 3/3
t3	Architecture	10/10, 3/3	10/10, 3/3
t4	Data Processing	10/10, 3/3	10/10, 3/3
t5	Full Stack	10/10, 3/3	10/10, 3/3

Task	Domain	Baseline (Tests / Edge)	AutoDev (Tests / Edge)
t6	Networking	0/10, 0/3	9/10, 3/3
t7	Data + Error Handling	10/10, 3/3	10/10, 3/3
t8	Advanced OOP	10/10, 3/3	10/10, 3/3

Table 2: Aggregate Performance Comparison

Metric	Baseline LLM	AutoDev
Functional correctness	83.8%	98.8%
Structural quality	8.8 / 10	9.4 / 10
First-attempt pass rate	87.5%	87.5%
Error recovery	Not supported	Consistent recovery
Robustness (edge cases)	83.3%	100%
Autonomy rate	Not supported	Fully autonomous
Average completion time	41.9 s	73.8 s
Average iterations	1.0	1.12

The aggregate results show that AutoDev improves correctness, robustness, and recovery behavior. The trade-off is higher completion time, which is expected because the workflow includes planning, execution, validation, and retry steps.

5.3 Illustrative Example

A thread-safe banking system with concurrent transactions was used as a realistic example of the workflow. The task required synchronization, state consistency, and multi-threaded execution, which are difficult for single-pass code generation. The initial output executed successfully but failed validation because the total balance was not preserved under concurrent operations. The failure was identified as a logical error. After a few refinement attempts, a brief human hint about the imbalance in account value was introduced, after which the locking strategy was revised and the corrected version passed validation. The example demonstrates how execution feedback and iterative correction can recover from non-trivial logical errors.

Figure 5: Initial execution failure showing imbalance during concurrent execution.

Figure 6: Human-in-the-loop guidance after automatic retries.

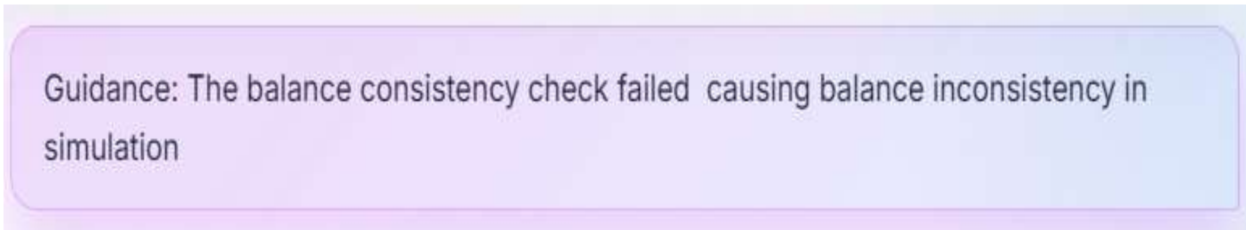
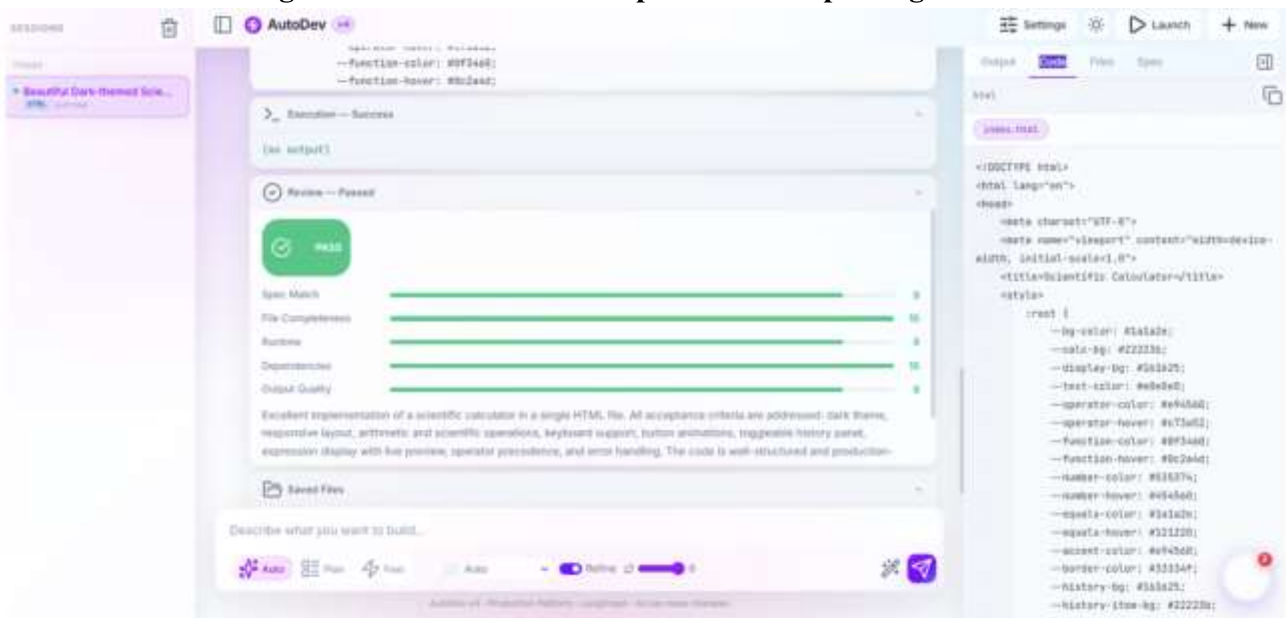


Figure 7: Final corrected implementation passing all tests.



5.4 Observations

The user interface makes the workflow visible, which improves trust and debugging. The session system reduces repetition by preserving previous context. RAG support improves continuity across attempts. The biggest limitation remains the extra execution time introduced by the multi-stage pipeline, but that overhead is balanced by a measurable gain in reliability.

6. Conclusion and Future Scope

6.1 Conclusion

AutoDev presents a workflow-based approach to generating and executing code from natural language prompts. By combining planning, execution, review, and iterative correction within a single system, it is able to produce working solutions across a range of programming tasks. The results show that incorporating execution feedback into the generation process improves reliability, particularly in tasks where single-pass models tend to fail.

6.2 Future Scope

The system can be extended with stronger provider support, finer-grained model selection, better output inspection, and richer session analytics. Further improvements may include more accurate dependency prediction, deeper local and remote model integration, better code preview tools, and stronger benchmarking against additional software engineering tasks. With these changes, AutoDev could be expanded into a broader autonomous development platform.

References

1. Cai Y, et al. Designing LLM-based multi-agent systems for software engineering tasks: Quality attributes, design patterns and rationale. arXiv preprint arXiv:2511.08475; 2025.
2. M. Monperrus, Automatic software repair: A bibliography. ACM Computing Surveys, 51(1), 2018.
3. M. Harman, The current state and future of search based software engineering. Proceedings of ICSE, 2007.
4. Li H, et al. The rise of AI teammates in software engineering (SE 3.0): How autonomous coding agents are reshaping software engineering. arXiv preprint arXiv:2507.15003; 2025.
5. Yang J, et al. SWE-smith: Scaling data for software engineering agents. arXiv preprint arXiv:2504.21798; 2025.
6. Dong Y, et al. A survey on code generation with LLM-based agents. arXiv preprint arXiv:2508.00083; 2025.
7. He G, et al. Plan-then-execute: An empirical study of user trust and team performance when using LLM agents as a daily assistant. arXiv preprint arXiv:2502.01390; 2025.
8. Amershi S, et al. Software engineering for machine learning: A case study. Proceedings of ICSE, 2019.
9. Zhang L, et al. SWE-bench goes live! arXiv preprint arXiv:2505.23419; 2025.
10. He J, et al. LLM-based multi-agent systems for software engineering: Literature review, vision and the road ahead. arXiv preprint arXiv:2404.04834; 2024.
11. Wang X, et al. OpenHands: An open platform for AI software developers as generalist agents. arXiv preprint arXiv:2407.16741; 2024.
12. Yang J, et al. SWE-agent: Agent-computer interfaces enable automated software engineering. arXiv preprint arXiv:2405.15793; 2024.
13. Xia CS, et al. Agentless: Demystifying LLM-based software engineering agents. arXiv preprint arXiv:2407.01489; 2024.
14. Yang J, et al. SWE-bench multimodal: Do AI systems generalize to visual software domains? arXiv preprint arXiv:2410.03859; 2024.
15. Zhuo TY, et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. arXiv preprint arXiv:2406.15877; 2024.
16. Lei C, et al. Planning-driven programming: A large language model programming workflow. arXiv preprint arXiv:2411.14503; 2024.
17. Ridnik T, et al. Code generation with AlphaCodium: From prompt engineering to flow engineering. arXiv preprint arXiv:2401.08500; 2024.
18. [18] Madaan A, et al. Self-refine: Iterative refinement with self-feedback. arXiv preprint arXiv:2303.17651; 2023.
19. Shinn N, et al. Reflexion: Language agents with verbal reinforcement learning. arXiv preprint arXiv:2303.11366; 2023.
20. Yao S, et al. ReAct: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629; 2022.
21. M Saadullah. AutoDev: Multi-agent autonomous coding system. GitHub repository, 2026. Available: https://github.com/MDSD0/AutoDev