

Fix CLI: A Multi-Agent Command-Line Interface System for Autonomous Bug Fixing in Web Development

Prof. Rajashree Salokhe¹, Harshal Patil², Akshay Chaudhari³, Dipak Shinde⁴, Mayur Wayzade⁵

¹Prof. Department of Computer Engineering, MGM College Of Engineering and Technology, Navi Mumbai, Maharashtra, India.

¹²³⁴Student, Department of Computer Engineering, MGM College Of Engineering and Technology, Navi Mumbai, Maharashtra, India.

Abstract - In today's world of software development, software developers face multiple challenges and errors or bugs in their projects. Debugging manually takes time and if developers help from today's Large Language Models (LLMs) models like Chatgpt then Gemini or Claude this takes time and less accuracy. Developers need one system that solves their errors and help them in their development and speed of developing the software must be fast.

To solve this problem, we introduced this paper which includes the AI powered CLI based Debugging Agent that solves the bugs and errors autonomously with help of Large Language Models (LLMs). Developers have tools like GitHub Copilot that suggest code snippets, or LLMs that suggest the codes, but they don't have the tool which solves the error autonomously without any interference of developers. This system has memory for agents to store the previously solved errors; this will reduce the API calling again and again for LLMs.

This system is based on reinforcement model like agents learns the processes overtime because they have their memory and due to that these agents get smarter overtime.

This agent system has debate feature which allows agents to take multiple suggestions from different LLMs with the error is complex and debate among the agents for one good solution and then fix it autonomously.

Key Words: Multi-Agent Systems, Large Language Models, Self-Healing Software, Automated Program Repair, MERN Stack, Command-Line Interface

1. INTRODUCTION

Software related bugs lead to system failures, badly impact the user experience, frustrate the developers, and this task becomes more complicated and takes a lot of time [1]. Usually, software developers take the help of Google or coding models that suggest code solutions, but it requires manual searching and takes lots of developers' time that they can use in other things. Studies show that developers can spend up to 35%–50% of their total development time on debugging and verification tasks.

In earlier times, Automated Program Repair (APR) methods used rule-based techniques, predefined fix patterns, or constraint solving approaches [5][8]. Later,

machine learning methods were introduced to learn bug-fixing patterns from existing code [6]. Recently, Large Language Models (LLMs) have become the most powerful tools for automated bug fixing because they understand code and generate source code [2][3][5].

Now in recent times, LLMs give suggestions for errors and bugs but only when the prompt given by the user is correct and appropriate [14]. This is a limitation of LLMs — the prompt needs to be good enough and the developer must also have knowledge about prompting, so this becomes more difficult when solving bugs [5]. Big organizations want projects to be completed on time, and bugs are becoming one of the main reasons for delays in project development [1].

Now in our system, multiple agents are working together to solve bugs effectively [10][12][16]. They have access to multiple LLMs at a time, and they take bugs from the user side, send an appropriate prompt to an LLM, take responses and suggestions, convert them into code, and automatically fix it [2][4].

If the code is not fixed, then a multi-agent system is activated. Multiple prompts are sent to multiple agents, they generate multiple suggestions and code solutions, then they start debating to find the best solution and apply it to the code [10][16]. This process increases the accuracy of the code and helps developers achieve their goal of solving bugs efficiently [4].

This system has a memory feature that stores previously solved code, which can be read by agents. This helps reduce the cost of calling multiple LLMs together because calling them takes time and has high cost, so memory gives an advantage here. Due to this memory feature, agents can learn over time how previous bugs were solved, and instead of calling LLMs again, agents can check memory — similar to how humans learn from past experiences and apply them in the present and future [11][1].

2. LITERATURE SURVERY

Increasing Complexity and heavy load of Developer to Create a Software, which is required human effort, proper technical knowledge, debugging code, lot of test cases To

Build A software user friendly So that's This is Not Proper Solution Even if Static work, but it is error Prone.

By Observing this limitation CLI Based Agent is Effective Solution in Software Development. CLI Allow Faster Execution which is not same for GUI Based application because GUI Based required Lot of time to Creating Effective Solution. CLI Agent give better Accuracy Because CLI Based Agent Work On different platform like PyCharm, Vscod etc and smoothly Connected with different programming language like JavaScript, python ,CSS, HTML that's why CLI agent Better for autonomous agent

CLI Bug Agent is a program That can observe the system, Generate Decision and fix error without human effort. multiple agent work inside Cli they read User code and perform task like fix bug, solve it and improve accuracy. Cli agent read User output and decide which agent used to Complete Task.

Recent growth of Artificial industry Multiple agent integration is easy and All Agent Work Together Like Shares information, Assign Task and solve it Fastly. This is useful for large software development also AI understand error message and learn from previous error which is make agent more accurate. Hence CLI-BASED AUTONOMOUS AGENTS provide Faster solution for software development by comparing other AI Tools like ChatGPT Because Other tool is not easy to handle Are required technical knowledge and prompt knowledge wrong prompt lead wrong output

3. METHODOLOGY

This project is trying something to create and test a system that helps fix problems in MERN stack web applications. The main goal of this system is to save time and work for developers when they are fixing bugs. Normally developers have to find and fix errors by themselves. This system uses artificial intelligence to do the fixing for them. The system is controlled from a command line interface. It works on its own to find and fix problems, in MERN stack web applications.

This work uses a way of doing research that tries things. The system we are talking about is tested with kinds of mistakes that people usually make when they are building web applications. These mistakes include ones that we know about already and ones that we find when we are actually working on a project. We use a design that is based on the system to put together a bunch of parts, such, as the Analyzer, the Fixer and the Verifier into one simple workflow that you can use from the command line. The system is made up of the Analyzer, the Fixer and the Verifier, which all work together in the system.

Data Source:

The information we use to test the system is from a MERN stack web project. These MERN stack web projects have mistaken that people make when they are programming like syntax errors and runtime errors and logical errors.

We also have life test cases so that the system works like it is, in a real work place. This way we can see how the MERN stack web projects and the system really work when we use them.

The tools and technologies that were used include the following:

* The tools

* The technologies: The tools and technologies that were used are the tools and technologies. The tools and technologies were good tools and technologies.

These are the tools and technologies that are used to make the system work:

* The system uses tools: the system also uses certain technologies to make the system work properly the system relies on these tools and technologies. Python is what we use to write the parts of the system. We also use Python to manage the agents. The main logic of the system is written in Python. This helps us to manage the agents with Python. LangChain is used to handle communication between agents. It helps these agents talk to each other. LangChain makes it possible for different agents to share information and work together. The main job of LangChain is to handle communication, between agents.

The OpenAI API is used to look at errors and come up with fixes for the OpenAI API. We use the OpenAI API to find mistakes. Then the OpenAI API helps us figure out how to fix them.

I use Node.js and MongoDB when I am testing applications that are built with the stack. This means I am working with Node.js and MongoDB to see how they work together in a stack application. The MERN stack is what I am really testing and Node.js and MongoDB are the tools that help me do that.

People use Command Line Interface tools to do things on the computer. They show what happens when you give a command. These Command Line Interface tools are really good for running commands. Then showing you what the results are. You can think of Command Line Interface tools like helpers that make it easy to give commands to the computer and see what the computer does with those commands. Command Line Interface tools are very useful for doing lots of things, on the computer.

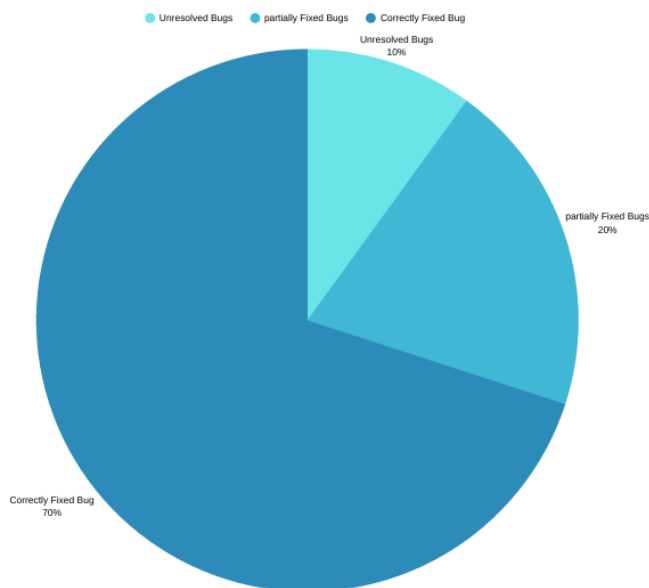
Performance Evaluation:

The system is checked to see how long it takes to find and fix errors, how good the fixes are and how time it saves the people who develop the system. We compare the results from this automated system with the results from debugging to see how well the automated system really works. The automated system is evaluated on the time

taken to debug errors the accuracy of the generated fixes and the reduction in developer effort. We look at the results from the automated system and the results, from manual debugging techniques to measure the effectiveness of the automated system.

Table -1: System Performance Metrics

Parameters	Values	Units
Debugging Time	10-15sec	Time
Fix Accuracy	70%	Percentage
Error Detection Rate	95%	Percentage
Developer Effort	Low	Level
Time Saved	70%	Percentage



Accuracy Of Cli Bug Debugging Agent

Fig -1: SYSTEM ARCHITECTURE

The pie chart represents the Accuracy Of CLI BUG AGENT. Approx 70 % bug are correctly fixed including MERN Stack Error.30 % Error are Partially Resolve and 10 % Error remains unsolved. This Demonstrate That System Almost Reduced Manual Effort and improve System Performance.

Working of the Proposed System:

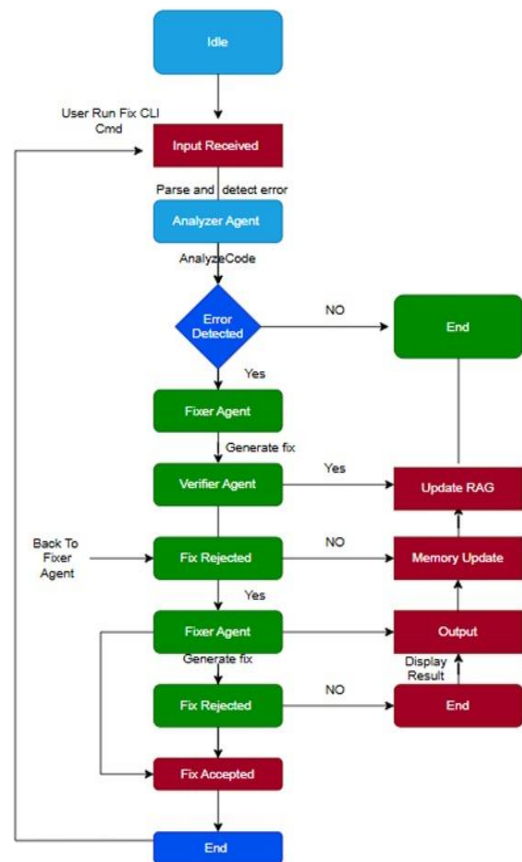


Fig -2: SYSTEM ARCHITECTURE

The debugging process is done by itself using a kind of pipeline that has many agents working together. First the Analyzer Agent looks at the code or error message that we get and figures out what kind of errors there. The Analyzer Agent does this to understand the problem.

Once the Analyzer Agent finds the error the Fixer Agent comes up with a solution or a patch for the code. Then the Verifier Agent checks the fix that the Fixer Agent generated to see if it really works or not. The Verifier Agent makes sure the fix is correct, for the code or error message that the Analyzer Agent looked at. If the fix does not work the Fixer Agent gets it back to make it better.

The Fixer Agent and the system keep doing this until they find a fix that works. When they finally find a fix, it is saved using a Retrieval-Augmented Generation mechanism. This helps the system remember the fix so it can solve problems with the fix more easily next time. The system uses this stored knowledge to solve errors with the fix more efficiently in the future.

Advantages of the Proposed Methodology

The system gives us a way to automatically find and fix problems using a command line interface. This is really helpful for debugging. We can use the command line to make things work properly. The system does all the work, for us so we do not have to do it. The command line interface is what makes this possible. This thing helps cut down the time it takes to fix mistakes. It really saves time when you have to fix errors. The time required to fix errors is a lot less now. The manual debugging effort that developers have to put in is really small. This means that developers do not have to spend a lot of time and effort on debugging. The manual debugging effort by developers is minimized, which is a thing, for developers. The system gets better over time because it uses the solutions that it has stored. It keeps getting better and better as time goes on. The system uses these stored solutions to improve itself. It can be scaled for large MERN stack projects

This picture shows the process of how the command line-based system works on its own to fix problems. The command line-based system is what we are talking about here and this command line-based system is able to find and fix issues by itself.

The system just sits there until the user decides to run a debugging command using the command line interface. When this command is given the system looks at the code or error details that the user has provided. The Analyzer Agent then goes through this information. Checks for things like syntax errors, problems with the logic or errors that happen when the code is running. The Analyzer Agent is really good at finding these kinds of errors in the input code or error details.

If everything is okay the process stops. If the Fixer Agent finds an error, it comes up with a solution. The Fixer Agent solution is then checked by the Verifier Agent. If the Fixer Agent solution does not work right it goes back, to the Fixer Agent for work. The Fixer Agent and Verifier Agent keep doing this until the Fixer Agent solution is checked and it works. When we check everything the fix that works is stored in the RAG memory. Then the correct output is shown to the user, on the command line. The RAG memory and command-line interface help us with this. After that the process is finished.

4. RESULT AND DISCUSSION

The system successfully detected and fixed syntax, runtime, and logic errors in multiple test scenarios. Debugging time was significantly reduced compared to the manual approach. Memory reuse improves correction accuracy with repeated use. Results are displayed using tables and graphs that show time savings, repair success rates, and system efficiency.

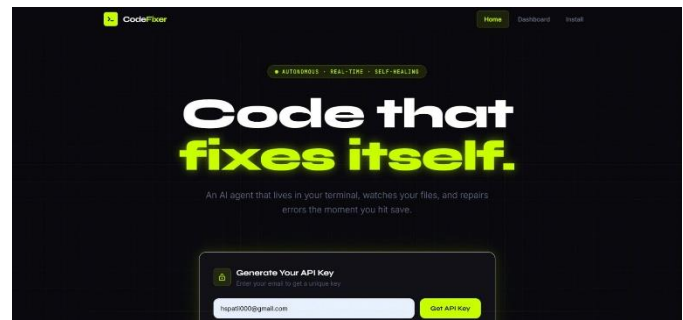


Fig -3: Generate API Key

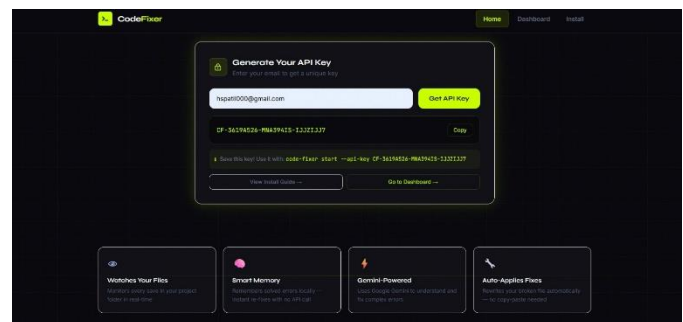


Fig -4: Copy API Key

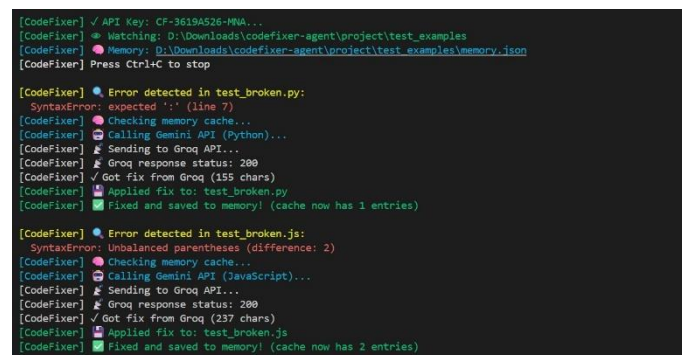


Fig -5: Paste API To Install System

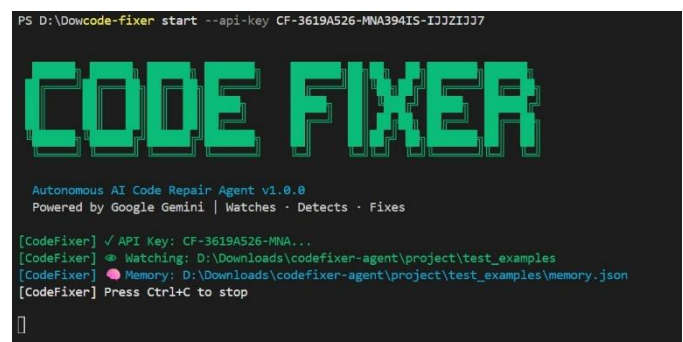


Fig -6: Start Agent System By Clicking "Ctrl+s"

During Development We Observed Following Point Discuss With our Project Guide and group member

1) Understanding Bug: Our Agent Does not see only error it collects information on many agents. Error messages shown Why the program crashes. The part of the code where the error happens Some time We Solved before. We introduced Memory based system which is store similar error. The System remembers everything and how it fixed This method help to save time and minimize repeated LLM call.

2) How The Multiagent Connectivity Help: Instead of one agent Give Ans, multiple agents suggest correct code then they check each other Fix code and Mistakes and Suggest better Code which is helping to remove incomplete error and improve accuracy specially where complicated error not reduced For One Quickly Ans.

3) Problem With Fix code: Some-time System Suggests Big and Complicated changes which is not required This happens because sometime LLM Overthink, but we not required this type of Ans. we decided in the future we try to solve this error. first Give simple Changes If it does not work then Give advance Suggestion. overall, the system needs to better understand and Knowledge Between Files.

4) Result can change sometime agent not Give Fix and if developer run the system again also agent use different methods so result may be partially changes.

5) High Cost: Multiagent system Improve accuracy but slightly costly because more agent required to more LLM calls and more time need to decide the final fix. Even only authorized user can use this agent due to privacy issue. in the future we trying to make the system smart like Direct Fixing For simpler bug and used multiagent only for hard to solve bug.

6) Need Good Test Cases

In real world Projects Some Test cases doesn't cover all bug so system think the bug is fix even if a problem still Exists

5. CONCLUSIONS

This Research Paper Demonstrate that, We successfully Developed but CLI bug agent which is Work on Multiagent System which is helping to reduce debugging time, minimizes human effort and Quickly identify error and auto fix it without required extra resources like ChatGPT and other powerful ai tools.

Our project Limitation include internet connectivity's must require When developer Used agent, and we will Focus on our agent work on offline which is beneficial because Developer can work anytime.

REFERENCES

- [1] M. Baqar, R. Khanda, and S. Naqvi, "Self-Healing Software Systems: Lessons from Nature, Powered by AI," 2025.
- [2] I. Bouzenia, P. Devaux, and M. Pradel, "RepairAgent: An Autonomous, LLM-Based Agent for Program Repair," 2024.
- [3] D. Hidvégi, K. Etemadi, S. Bobadilla, and M. Monperrus, "CigaR: Cost-Efficient Program Repair with Large Language Models," 2024.
- [4] X. Meng, Z. Ma, P. Gao, and C. Peng, "An Empirical Study on LLM-Based Agents for Automated Bug Fixing," 2024.
- [5] S. MacNeil et al., "The Use of Large Language Models for Program Repair," 2024.
- [6] Multiple Authors, "A Deep Dive into Large Language Models for Automated Bug Repair," 2024.
- [7] M. Lewis et al., "LangChain Documentation: Building Applications with Large Language Models," 2023.
- [8] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy, "InferFix: End-to-End Program Repair with Large Language Models," 2023.
- [9] Y. Zhang, Z. Jin, Y. Xing, and G. Li, "STEAM: Simulating the Interactive Behaviour of Programmers for Automatic Bug Fixing," 2023.
- [10] P. K. Rajput and G. Sikka, "Multi-Agent Architecture Approach for Self-Healing Systems: Run-Time Recovery with Case-Based Reasoning," 2022.
- [11] M. Chen et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," 2021.
- [12] F. C. Sampaio et al., "A Multi-Agent-Based Integrated Self-Healing and Adaptive Protection System," 2020.
- [13] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed., Pearson, 2020.
- [14] T. Brown et al., "Language Models Are Few-Shot Learners," 2020.
- [15] A. Sweigart, Automate the Boring Stuff with Python, 2nd ed., 2019.