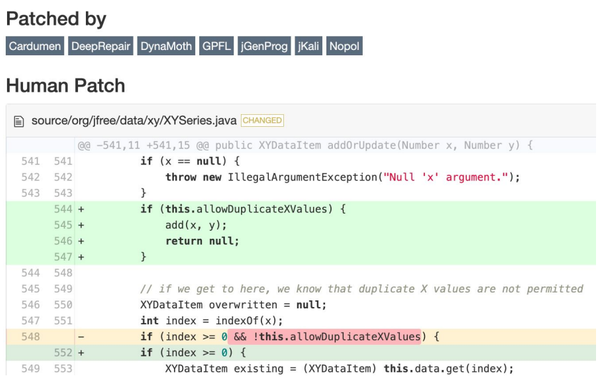
Patching Multi-Location Software Bugs: A Multi-Agent Large Language Model Framework for Automated Program Repair

Background

Automated program repair is the process of automatically fixing software bugs with little to no human intervention. A **patch** is a piece of code that modifies a program to fix a bug.

Multi-location bugs span multiple non-contiguous locations in source code. These bugs are widespread, but most existing repair tools either do not support or perform poorly on multi-location repair.



Example of a multi-location bug (Bug ID: Chart 5), as shown in the Defects4J web user interface. Image from program-repair.org/defects4j-dissection.

Defects4J (v1.2) is the most popular bug repair dataset. It spans 395 bugs across six real-world open-source projects. Of the 395 bugs, 190 are multi-location. In this research, all analysis and testing is performed on these multi-location bugs.

Introduction

Repairing bugs is crucial in code development because unfixed bugs can cause a wide range of consequences. Every year, over $100 billion and vast amounts of time are spent on debugging.

This research proposes a framework to automatically repair multi-location bugs. When implemented, the system first takes the source code and bug locations as input. Then, it retrieves candidate APIs and repair patterns from the databases. Finally, the task is distributed to large language model (LLM) agents which generate, verify, and return the best patch.

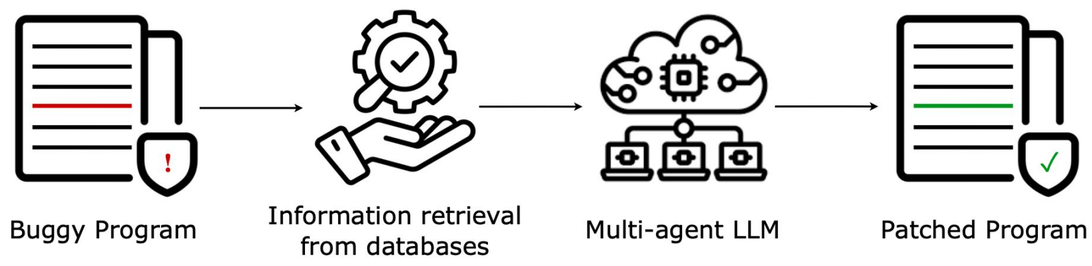


Figure 1: Overall system architecture. Figure created by researcher using draw.io (2024).

Methodology and Implementation

Challenge #1: Identifying and applying prior knowledge. While LLMs have been trained on extensive open-source code bases, they struggle to apply their general programming knowledge to specific bug repair scenarios.

API database: Constructed to help the LLM agents rapidly pinpoint candidate APIs to call. It contains general and project-specific APIs.

Repair pattern database: Summarizes the most common repair patterns. The step-by-step repair instructions serve as a guide for the LLM agents as they fill in the framework with program-specific details.

Challenge #2: Length of LLM prompt. Due to their limited long-term memory, LLMs often lose context and fail to follow all instructions when prompts get excessively long.

A **collaborative multi-agent design** is employed, where each of the six LLM agents performs a different role in the repair process. This improves the accuracy and efficiency of patch generation. Each patch is verified via a self-check approach with bidirectional reasoning. This reveals logical contradictions in incorrect patches.

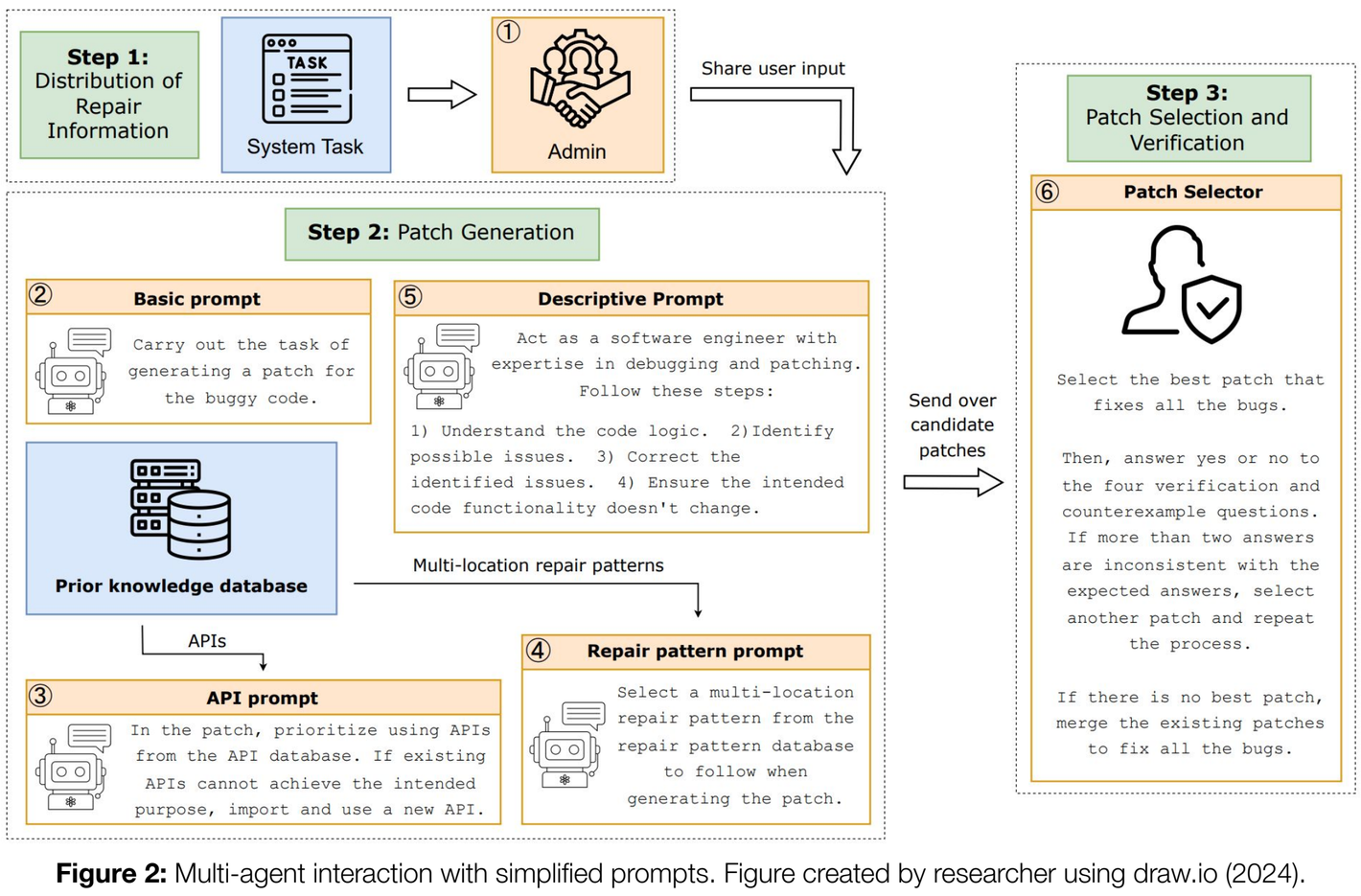


Figure 2: Multi-agent interaction with simplified prompts. Figure created by researcher using draw.io (2024).

The system is implemented with AutoGen, an open-source framework for building LLM applications. Databases were imported as separate files. GPT-4 was set as the model for each agent. This system operates under the assumption of perfect fault localization (i.e., the bug location has been identified), which is the preferred setting for recent works.

Results

Project	Count	This research	GPT-4	Nopol	DynaMoth	DeepRepair
Chart	10	7	3	3	3	5
Closure	54	20	5	20	19	0
Lang	31	13	4	0	1	4
Math	59	23	12	10	9	18
Mockito	22	5	1	0	0	0
Time	14	2	0	1	0	0
Total	190	70 (37%)	25 (13%)	34 (18%)	32 (17%)	27 (14%)

Table 1: Repair results on all multi-location bugs compared to base GPT-4 and traditional repair tools, categorized by project.

This research yields a significant increase in repair accuracy compared to existing repair tools. In addition to the displayed results, this research also outperforms existing multi-location repair tools (i.e., ITER, Hercules, VarFix, DEAR) when compared to their 30 publicly available results. This research **incurs an approximately equal monetary cost and is especially efficient in terms of time consumption** when compared to current LLM-based repair tools (i.e., RepairAgent, ITER, ChatRepair).

	Total	Average
API Requests	2,054	10.81
Token consumption	5,842,074	30,747.76
Time (seconds)	18,663	98.23
Cost (dollars)	80.25	0.42

Table 2: Cost analysis (total vs. per patch).

Impact

- **This research is among the first to utilize a multi-agent framework to specifically target multi-location bug repair.** The automated program repair community can easily extend and refine this framework going forward.
- **This repair system can be incorporated into continuous integration and continuous deployment (CI/CD) pipelines.** When used with a fault localization tool, it can streamline the CI process, making it easier to merge bug-free code into a centralized code base for deployment.

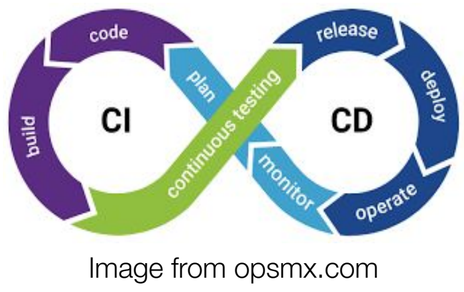


Image from opsmx.com

- **The generated patches can be used as training data.** The buggy source code files and labeled patches can be used to train code generation models.