Modular Multi-Stage Agent for Bug Fixing - Analysis of Potentials and Limitations

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Internationale Medieninformatik*

1. Gutachter_in: Prof. Dr. Gefei Zhang
2. Gutachter_in: Stephan Lindauer

Eingereicht von Justin Gebert [s0583511]

22.07.2025

Danksagung

[Text der Danksagung]

Abstract

Generative AI technologies are reshaping software engineering practices by automating critical tasks, including code generation, debugging and program repair. Despite these advancements, existing Automated Program Repair (APR) systems frequently suffer from complexity, high computational demands, and missing integration within practical software development lifecycles, particularly Continuous Integration and Continuous Deployment (CI/CD) workflows. Such shortcomings often lead to frequent context switching, which negatively impacts developer productivity.
In this thesis, we address these challenges by introducing a novel and lightweight APR system leveraging LLMs, explicitly designed for seamless integration into CI/CD pipelines deployed in budget constrained environments. Our containerized approach, developed with a strong emphasis on security and isolation, manages the complete bug-fixing lifecycle from issue creation on GitHub to the generation and validation of pull requests. By automating these processes end-to-end, the system significantly reduces manual intervention, streamlining developer workflows and enhancing overall productivity.
We evaluate our APR system using the QuixBugs benchmark, a recognized dataset for testing APR methodologies. The experimental results indicate that our streamlined and cost-effective solution effectively repairs small-scale software bugs, demonstrating practical applicability within typical software development environments.
The outcomes underscore the feasibility and advantages of integrating APR directly into real-world CI/CD pipelines. We also discuss limitations inherent in LLM-based solutions, such as accuracy and reliability issues and suggest future enhancement and research.

-add that this approach directly integrates into the development lifecycle reducing configuration ...

Contents

List of Figures

List of Tables

Listings

1. Introduction

1.1. Background and Motivation

Generative AI is changing the software industry and how software is developed. The emergence of Large Language Models (LLMs) has opened up new possibilities for automating various aspects of software engineering, including code generation, debugging, and program repair. These models have demonstrated remarkable capabilities in understanding and generating code snippets, making them valuable tools for developers. Since debugging and fixing bugs are critical tasks in software development, the integration of LLMs into Automated Program Repair (APR) systems has gained significant attention. Recent studies have shown that LLMs can effectively assist in identifying and fixing bugs, thereby improving the efficiency of the software development process. However, the integration of these practices into existing software development workflows remains understudied.

exsitng apporaches are often complex and require significant computational resources, making them less suitable for budget-constrained environments. Additionally, the lack of integration with Continuous Integration and Continuous Deployment (CI/CD) practices limits their practical applicability in real-world scenarios. The motivation behind this thesis is to explore the potential of LLMs in enhancing automated program repair (APR) systems, particularly in continuous integration and continuous deployment (CI/CD) environments. By leveraging the capabilities of LLMs, we aim to develop a more efficient and cost-effective approach to automated bug fixing that can seamlessly integrate into existing software development workflows.

while writing this paper lots of reasearch and tools have been published clearly showing the importance of the topic and the need for further research in this area. The rapid advancements in LLMs and their applications in software engineering highlight the potential for significant improvements in automated program repair systems. As the software industry continues to evolve, it is crucial to explore innovative solutions that can enhance the efficiency and effectiveness of software development processes.

1.2. Problem Statement

modern APR systems requiremt a lot of computational power budget and manual effort - stduies suggest emphasizing connections with DevOps Procedures [6]

1.3. Objectives and Research Questions

The primary objective of this thesis is to investigate the feasibility and effectiveness of LLMs in APR within the Software Development Lifecycle (CI/CD pipelines) in budget

restrained environment. The research questions guiding this investigation include:

2. Background and Related Work

2.1. Software Engineering

Software engineering is a complex dicipline consisting of constant

Continous Integration and Continous Deployment (CI/CD) is a software development practice that emphasizes frequent integration of code changes into a shared repository, followed by automated testing and deployment. This approach aims to enhance collaboration, reduce integration issues, and accelerate the delivery of high-quality software.

2.1.1. code hosting platforms

github is whre open soruces lives and development takes place. It is a platform that allows developers to host, share, and collaborate on code repositories. GitHub provides version control, issue tracking, and collaboration tools, making it a popular choice for open-source projects and software development teams.

allows for integration of systems liek rennovate

software development moves more and more towards lightlz coupled mullti microsversives which makes code repositories smaller but more numerous. This trend is driven by the need for flexibility, scalability, and faster development cycles. Smaller code repositories allow teams to work on specific components or services independently, reducing dependencies and enabling quicker iterations. This approach aligns with modern software development practices, such as microservices architecture and agile methodologies.

Bug fixing is a highly resource intensive task in software engineering, consiting of multiple stages .

APR:

this is where APR comes into play

2.1.2. Software Development Lifecycle

2.1.3. Continuous Integration and Continuous Deployment (CI/CD)

2.2. Automated Programm Repair

Automated Program Repair (APR) helps developers fix bugs

2.2.1. Evolution of Automated Program Repair

Earliest APR techniques were based on version control history, using the history to roll back to a previous version of the code part, where no issues were present. This

approach, while effective in some cases, often lacked the ability to perserve new features. (more like instant rollback) history based

Initial tempalte based repair, apply predefined transformtions to the code based on rules

The emerge of llm based techniques LLM based APR techniques have demonstrated siognificant uimrpovemetns over all other state of the art technqiues, benfitintting from theor coding knowledge [**hossainDeepDiveLarge2024**]

Agent Based agent based system improve fixing abilites by probiding llms the ability to interact with the code base and the environment, allowing them to plan their actions [8].

llms lay the groundwork of a new APR paradigm [1]

complex agent arcitectures produce good results espically paired with containerized environments. Emphasis on quality insureance and Devops practices [6]

modern aprs usally consist of multiple stages, including localization, repair, and validation. These stages are often implemented as separate modules, allowing for flexibility and modularity in the repair process [8].

2.3. LLMs in Software Engineering

modern large language models have billions of paramters, are pre-tained on massive codesbases which results in extraordinary capbilites in this area [1].

problems with llms are: Information leakage, hallucinations, and security issues

first LLms now research is looking into developing and improving workflows leveraging LLMs [6].

problems wi looking into Agents using tools, LLMs + RAG,

2.4. LLM-Based Tool Use and CI Context

CI allows seemless integration... this way there is no harmfull code executed on own machiene, its encapsulated mutliple times container in Ci runner

2.5. Related Work - Existing Systems

end to end without llms Sapfix from Facebook. Fixing bugs in production envrioments lowerring incidents mean time of recovery significantly [5]

FixAgegent [3]

swe agent [8]

Agentless minimal system [7] claims exsiting systems are too complex and compute/-costs intensive. lacks the ability to control the decision planning. they use a minimalist approach using localization, repair and validation.

this appraoch inspried the thesis to test how a simple approach will perform in a real world scendario on a code hosting platform.

–point out areas where current systems see limitation

– during the research for this thesis, full integrations where published by companies like OpenAI Codex and Github Copilot - but these are not open source

3. Requirements

- for this prototype we constructed Requirements which the system shall statisfy - we split into functional requirements: t.3.1 (EXPLAINATION), nonfucntional Requirements combined with security requriemnts

- these requiremtns allow for better planning and priotisation during development. - the satisfaction of all the requriemtns will allow for evaluation of the integration into the software development lifecycle

3.1. Functional Requirements

3.2. Non-Functional Requirements

ID	Title	Description	Verification
F0	Multi Trigger	The Pipeline can be triggered: manually, scheudled via cron, or by GitHub issue creation/labeling.	Runs can be found for these triggers
F1	Issue Gathering	Retrieve GitHub repository issues and filter them for correct state and configured labeles `BUG`.	`gate` logs list of fetched issues.
F2	Code Checkout	Fetch the repository code into a fresh workspace and branch (via Docker mount).	After F1, workspace/ contains the correct source files.
F3	Issue Localization	Use LLM to analyze the issue description and identify relevant files.	LLM output contains file paths with files that shall be edited.
F4	Fix Generation	Use LLM to edit the identified files.	LLM output contains adjusted content for the identified files.
F5	Change Validation	Run format, lint and relevant tests and capture pass/fail status.	Logs show build and test results.
F6	Iterative Patch Generation (retry logic)	If F4 reports failures, retry F4–F5 up to X times.	After retries, either F4 passes or fails with no further retries.
F7	Apply Patch	Commit LLM-generated edits and generate patch.	Git history shows a new commit which is referencing the Github issue
F8	Result Reporting	Open a PR or post a comment on GitHub with the diff and summary metrics.	A PR or comment appears for each issue, showing diff and summary.
F9	Logs and Metrics Collection	Provide log files and Metrics with fix-rate, attempt history, timings, token ussage.	A metrics file contains fields: issue, success, timings, stages.

Table 3.1.: *Functional requirements (F0–F8)*

ID	Title	Description	Verification
N1	Containerized Execution	All agent code runs in CI runner in a Docker container to isloate it.	Workflow shows Docker container usage
N2	Configurability	User can specify issue labels, branches, attempts, LLM models via YAML.	Changing the config file alters agent behavior accordingly.
N3	Portability	The system can be deployed on any repository on GitHub.	???
N4	Reproducibility	Runs are deterministic given identical repo state and config.	Multiple runs on the same issue report similar metrics.
N5	Oberability	The system provides logs and metrics for each run, including execution times, token usage and success rate.	Logs and metrics files are generated after each run, containing all relevant data.

Table 3.2.: *Non-Functional (N1–N5) requirements*

4. Methodology

The primary objective of this evaluation is to assess the potentials and limitations of our APR pipeline when integrated into a real-world software development lifecycle. We aim to answer the following research question (RQs) to evaluate the system's capabilities and impact on the software development process: RQ1: Which potentials and limitations does the APR pipeline have ???

4.1. Preparation

4.1.1. Dataset Selection

For the evaluation of the APR integration into the software development lifecycle, we selected the Quixbugs dataset [4] as our primary benchmark for testing APR integration. This dataset is well-suited for our purposes due to its focus on small-scale software bugs in Python. It consists of 40 algortithmic bugs each in one file consisting of a single erroinums line , each with a correspoding tests for repair validation. Because these bugs where developed as challending problems for developers [4], we can evaluate if our system can take over the complex fixing of small bugs without developer intervention to prevent context switching for developers.

Compared to other APR benchmarks like SWE-Bench [2] Quixbugs is relaivly small which accelerates setup and development.

if archieved I will ad swe bench lite later [2]

4.1.2. Environment Setup

To mirrow realistic software development environment, we prepared the Quixbugs dataset by creating a GitHub repository. This repository serves as the basis for the bug fixing process, allowing the system to interact with the codebase and perform repairs. The repository contains only relevant files and folders required for the bug fixing process, ensuring a clean environment for the system to operate in.

We automatically generated a GitHub issue for each bug, using a consistent template that captures just the Title of the Problem. These issues serve as the entry points to our APR pipeline.

4.1.3. Pipeline Arcitecture

4.2. Evaluation

In this section, we describe how we measure the effectiveness and performance of our APR pipeline when integrated into a real-world CI process, using our QuixBugs

repository as

For Evaluation we will focus on several key metrics to assess the system's performance and abilites in repairing software bugs. These metrics will provide insights into the system's efficiency, reliability, and overall impact on the software development lifecycle. The following metrics will be used:

The follwing metrics are automatically collected for each run of the APR pipeline:

- **Repair Success Rate**: Calculate the percentage of successfully repaired bugs out of the total number of bugs attempted by using test results. - **Number of Attempts**: Track the number of attempts made by the system to repair each bug. - **Overall Execution Time in CI/CD**: Evaluate the time taken for the system to execute within a CI/CD pipeline, providing insights into its performance in real-world development environments. - **Execution Time of Dockerized Agent**: Measure the time taken by the Containerized agent to execute the repair process, which helps in understanding the efficiency of the containerized environment and help evaluate the computer overhead of CICD. - ** Token Usage**: Monitor the number of tokens used by the LLM during the repair process, which can help in understanding the cost and efficiency of the model's usage. - **Cost per Issue**: Calculate the cost associated with repairing each bug, considering factors such as resource usage, execution time, and any additional overhead.

what I evaluate: script execution time + CICD overhead one issue vs mutliple issue times model vs model metrics costs attemps vs no attemps in all these categories

5. Implementation

```
object HelloWorld {
def main(args: Array[String]): Unit = {
  println("Hello, world!")
}
}
```

Listing 5.1: *Ein Beispiel: Hello World (Scala)*

5.1. Implementation

Here we break down the implementation of the system into its core components, following the design and methodology outlined in the previous sections. The full code is attached in the **??** appendix.

5.1.1. System Architecture

IMAGE of Figma diagram

5.1.2. System Components

ci pipeline agent core

5.1.3. System Configuration

secrets that need to be added: LLM provider API key, GITHUB TOKEN need to anable gituhb actions permission to create pull requests in repo settings

explain fields:

[0]During development we followed the paradigms introduced in clean code. [**martin2008**]).

6. Results

[Beschreibung der Ergebnisse aus allen voran gegangenen Kapiteln sowie der zuvor generierten Ergebnisartefakte mit Bewertung, wie diese einzuordnen sind]

6.1. APR System Results

6.2. Evaluation Results

- descibe and showcase execution times and repair success rates - show a log?
  with small models and attempt loop makes small models pass the whole benchmark?

7. Discussion

7.1. Validility

- quixbugs a small dataset, not representative of real world software development - only python, not representative of real world software development - but shows the potential of applying llm based agents in a real world cicd environment

7.2. Potentials

- can take over small tasks in encapuslated envrioment without intervention
  - small models can solve more problems with retrying with feedback

7.3. Limitations

- potential is highly dependand on model - github actions from github have a lot of computational noise - workflow runs on every issue and therefor has some ci minute overhead this could be solved by using a github app which replies on webhook events
  – SECURITY ISSUE: Prompt injection in issue: CICD makes this a bit safer?

7.4. Summary of Findings

7.5. Lessons Learned

- ai is a fast moving field with a lot of noise

7.6. Roadmap for Extensions

- Service Accoutns for better and mroe transparent integration - try out complex agent architectures and compare metrics and results - try out more complex bug fixing tasks - SWE bench

8. Conclusion

References

[1] Zhi Chen, Wei Ma, and Lingxiao Jiang. *Unveiling Pitfalls: Understanding Why AI-driven Code Agents Fail at GitHub Issue Resolution*. Mar. 2025. DOI: 10.48550/arXiv.2503.12374. arXiv: 2503.12374 [cs]. (Visited on 03/24/2025).

[2] Carlos E. Jimenez et al. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* Nov. 2024. DOI: 10.48550/arXiv.2310.06770. arXiv: 2310.06770 [cs]. (Visited on 03/06/2025).

[3] Cheryl Lee et al. *A Unified Debugging Approach via LLM-Based Multi-Agent Synergy*. Oct. 2024. DOI: 10.48550/arXiv.2404.17153. arXiv: 2404.17153 [cs]. (Visited on 03/06/2025).

[4] Derrick Lin et al. "QuixBugs: A Multi-Lingual Program Repair Benchmark Set Based on the Quixey Challenge". In: *Proceedings Companion of the 2017 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*. Vancouver BC Canada: ACM, Oct. 2017, pp. 55–56. ISBN: 978-1-4503-5514-8. DOI: 10.1145/3135932.3135941. (Visited on 04/17/2025).

[5] Alexandru Marginean et al. "SapFix: Automated End-to-End Repair at Scale". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. Montreal, QC, Canada: IEEE, May 2019, pp. 269–278. ISBN: 978-1-7281-1760-7. DOI: 10.1109/ICSE-SEIP.2019.00039. (Visited on 03/06/2025).

[6] Meghana Puvvadi et al. "Coding Agents: A Comprehensive Survey of Automated Bug Fixing Systems and Benchmarks". In: *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*. Mar. 2025, pp. 680–686. DOI: 10.1109/CSNT64827.2025.10968728. (Visited on 04/27/2025).

[7] Chunqiu Steven Xia et al. *Agentless: Demystifying LLM-based Software Engineering Agents*. Oct. 2024. DOI: 10.48550/arXiv.2407.01489. arXiv: 2407.01489 [cs]. (Visited on 04/24/2025).

[8] John Yang et al. *SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering*. Nov. 2024. DOI: 10.48550/arXiv.2405.15793. arXiv: 2405.15793 [cs]. (Visited on 04/20/2025).

A. Appendix

A.1. Quell-Code

A.2. Tipps zum Schreiben Ihrer Abschlussarbeit

- Achten Sie auf eine neutrale, fachliche Sprache. Keine „Ich“-Form.
- Zitieren Sie zitierfähige und -würdige Quellen (z.B. wissenschaftliche Artikel und Fachbücher; nach Möglichkeit keine Blogs und keinesfalls Wikipedia[1]).
- Zitieren Sie korrekt und homogen.
- Verwenden Sie keine Fußnoten für die Literaturangaben.
- Recherchieren Sie ausführlich den Stand der Wissenschaft und Technik.
- Achten Sie auf die Qualität der Ausarbeitung (z.B. auf Rechtschreibung).
- Informieren Sie sich ggf. vorab darüber, wie man wissenschaftlich arbeitet bzw. schreibt:
  - Mittels Fachliteratur[2], oder
  - Beim Lernzentrum[3].
- Nutzen Sie LaTeX[4].

[1] Wikipedia selbst empfiehlt, von der Zitation von Wikipedia-Inhalten im akademischen Umfeld Abstand zu nehmen [**wikipedia2019**].

[2] Z.B. [**balzert2011**], [**franck2013**]

[3] Weitere Informationen zum Schreibcoaching finden sich hier: `https://www.htw-berlin.de/studium/lernzentrum/studierende/schreibcoaching/`; letzter Zugriff: 13 VI 19.

[4] Kein Support bei Installation, Nutzung und Anpassung allfälliger LaTeX-Templates!

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

_____

Datum, Ort, Unterschrift