Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

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 poor 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.

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

2.2. Automated Programm Repair

Automated Program Repair (APR) helps developers fix bugs

2.3. 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].

2.4. 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,

2.5. LM-Based Tool Use and CI Context

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

2.6. 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.

3. Requirements

3.1. Functional Requirements

ID	Title	Description	Verification
F0	Issue Gathering	Query GitHub for open issues labeled `BUG` in the repository.	`runAgent` logs list of fetched issue numbers and URLs.
F1	Fetch Code	Fetch the code referenced by the issue into a fresh workspace (via Docker mount).	After F1, workspace/issue/ contains the correct source files.
F2	Apply Patch	Apply an LLM-generated diff patch to the workspace files.	After patch, `git diff` output matches the patch payload.
F3	Build & Test	Run the project's test suite inside a Docker container and capture pass/fail status.	Docker exit code = 0 for pass and a JSON report file exists.
F4	Report Results	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.
F6	Metrics Collection	Log fix-rate, CI-cycle time, and sandbox events in CSV or JSON format.	A metrics file contains fields: issue, pass/fail, time, incidents.
F7	Per-Issue Trigger	Execute F1–F4 for each fetched issue in sequence and aggregate results.	For N issues, produce N PRs (or "no-fix" comments) and N metric entries.
F8	GitHub Integration	Use the GitHub API (with `GITHUB_TOKEN`) to list issues, create branches, open PRs, and post comments.	All GitHub API calls succeed without manual intervention.
F9	Cost Accounting	The system shall record prompt tokens, completion-tokens and total cost per issue aggregated into a metrics file.	The metrics file contains fields: issue, prompt-tokens, completion-tokens, cost.

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

3.2. Non-Functional and Safety Requirements

3.3. Benchmark Setup

- QuixBugs problems (Python).
- Prepare a base Docker image (e.g., `python:3.10`) with pytest and JSON-report support.
- Ensure each workspace clone is clean (no leftover artifacts).
- Record benchmark IDs and Docker image tags in a configuration file for reproducibility.

ID	Title	Description	Verification
N1	Performance Budget	Total wall-clock time per issue run $\leq$ X minutes (including Docker startup).	Average CI-cycle time across issues <= X minutes.
N2	Resource Limits	Docker container limited to <= X GB RAM and <= X CPU cores.	Launched with `-memory=Xg -cpus=X;` verify via container stats.
N3	Reproducibility	Runs are deterministic given identical repo state and config.	Multiple runs on the same issue yield identical metrics.
N4	Configurability	User can specify issue labels, timeouts, resource caps, and stage toggles via YAML or JSON.	Changing the config file alters agent behavior accordingly.
N5	Scheduling Config	Workflow can be scheduled via cron or manually triggered (GitHub Actions workflow_dispatch).	Adjusting the schedule in Actions YAML takes effect.
N6	Cost Budget	cost per issue run <= X	Average cost across issues <= X.
N7	Portability	The system can be deployed on any repository on GitHub.	The system can be deployed on any repository on GitHub.
S1	Filesystem Isolation	Prevent reads/writes outside the workspace directory (no escapes).	Attempts to access paths outside workspace/ are blocked and logged.
S2	Network Whitelist	Block all outbound network traffic except to configured LLM API endpoints.	Non-LLM outbound connections are refused by Docker network policy.
S3	Rollback on Failure	On test or policy failure, auto-reclone a fresh workspace copy before retry.	After failure, workspace resets to its pre-run state.
S4	Command Whitelisting	Only allow predefined shell commands (e.g., git, pytest, npm test); block others.	Forbidden commands (e.g., rm -rf /) are denied.

Table 3.2.: *Non-Functional (N1–N5) and Safety (S1–S4) requirements*

4. Methodology

4.1. Preparation

4.1.1. Dataset Selection

quixbugs, a small problem set in python [4] consisting of 40 carefully crafted python programs with bug.

mostly algorithmical problems hard to solve for humans without diving deep into the fucntionaltity of the desired programm

if archieved swe bench [2]

4.2. Evaluation Stragegy and Metrics

metrics to evaluate the system's performance, effectiveness in repairing software bugs and integration into a real world software developemnt lifecycle are crucial for understanding its impact and areas for improvement. Our evaluation will focus on quantitative and qualitative metrics, ensuring a comprehensive assessment of the system's capabilities.

4.2.1. Evaluation Strategy

The evaluation strategy will involve the following steps: 1. **Dataset Preparation**: Use the QuixBugs dataset, which contains a variety of small-scale software bugs in Python, to test the system's repair capabilities. 2. **System Configuration**: Set up the system in a controlled environment, ensuring that it integrates into a real world sfotware development envrioment. Having acess to nesseary resources and tools for the repair process. 3. **Execution of Repair Process**: Run the system through the bug fixing lifecycle, from issue creation to pull request generation, while monitoring its performance. 4. **Data Collection**: Collect data on the system's performance during the repair process, including execution time, success rates, and any encountered issues. Metrics listed **??**. 5. **Analysis of Results**: Analyze the collected data to evaluate the system's bug repairing capabilities and integration, focusing on both quantitative metrics and qualitative feedback from the repair process.

4.2.2. Metrics

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: - **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 Docker 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. - **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. - **Number of Code Issues**: Count the total number of code issues identified and attempted to be repaired by the system. - **Cost per Issue**: Calculate the cost associated with repairing each bug, considering factors such as resource usage, execution time, and any additional overhead.

- Metrics: - Full Execution Time in CI/CD - Execution Time of docker agent - Execution Time per Stage - Repair Success Rate - nubmer of Attempts - number of code issues - Cost per issue

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

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]

7. Discussion

7.1. Validility

7.2. Limitations

7.3. Potentials

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