GATeR: Graph-Aware Test Repair

Project Team

Ahsan Faraz	22I-8791
Dawood Hussain	22I-2410
Mirza Mukarram	22I-2488

Session 2022-2026

Supervised by

Mr. Pir Sami Ullah Shah

Department of Software Engineering

National University of Computer and Emerging Sciences
Islamabad, Pakistan

September, 2025

Contents

Chapter 1

Introduction

This document outlines the development of GATeR (Graph-Aware Test Repair), an automated test repair system that leverages knowledge graphs, semantic search, and Large Language Models (LLMs) to address critical challenges in software maintenance by automatically fixing broken tests using repository-wide context and intelligent graph-based analysis.

GATeR represents an advancement in automated test repair technology, combining graph analysis techniques with modern AI capabilities to deliver a solution aimed at reducing maintenance costs, improving software reliability, and enhancing developer productivity.

1.1 Motivation

The development of GATeR is motivated by several critical factors in modern software development:

1.1.1 High Maintenance Cost

Tests frequently break with system changes, requiring costly manual fixes. Development teams spend significant resources maintaining test suites, with studies showing that test maintenance can consume up to 30% of development effort.

1.1.2 Declining Quality

Broken tests reduce trust in software reliability and create a cascade of problems including delayed releases, increased bug rates, and decreased developer confidence in the testing infrastructure.

1.1.3 Tool Limitations

Current approaches miss repository-wide context, producing fixes that work syntactically but fail to align with project conventions, coding standards, and architectural patterns.

1.1.4 Opportunity

The combination of knowledge graphs and AI technologies presents an opportunity to create smarter, more scalable test repair solutions that understand and respect project-specific contexts.

1.2 Problem Statement

Current test repair tools lack repository-wide context, producing unreliable and non-idiomatic fixes that fail to capture project-specific patterns and conventions. Software development teams face significant challenges when tests break due to system changes, requiring costly manual intervention and expertise. Existing approaches suffer from incomplete understanding of project-specific patterns, utility functions, coding conventions, documentation (READMEs), and historical context from issues and pull requests. Additionally, LLM context constraints with limited context windows (4K-32K tokens) cause overflow, reduced efficiency, and decreased accuracy. Traditional knowledge graph building is resource-intensive, taking 45-60 minutes and consuming 4-6GB memory for 500K lines of code. Furthermore, current systems lack reasoning capabilities with no interpretable repair decision chains, limiting trustworthiness and maintainability. This project aims to develop a scalable, context-aware solution that generates accurate and maintainable repairs while addressing these fundamental limitations.

1.3 Proposed Solution/Method

GATeR addresses these challenges through an innovative 9-step technical workflow that combines multiple advanced technologies:

1. **Access Codebase**: Retrieve code from GitHub repositories with incremental updates using GitPython and PyGithub
2. **Parse with Tree-sitter**: Extract Abstract Syntax Trees faster than traditional parsers
3. **Build Graph Structure**: Create knowledge graphs using KGCompass methodology

4. **Store in Kùzu**: Persist graph relationships in an embedded graph database
5. **Calculate Relevance**: Score entities using the KGCompass formula for intelligent context selection
6. **Store in LanceDB**: Store vector embeddings for semantic search capabilities
7. **Retrieve Context**: Use GraphRAG for enhanced context retrieval
8. **Augment Context**: Enhance context with GraphRAG techniques
9. **Generate Fix**: Produce repairs using LLM and GraphRAG integration

The solution aims to achieve improved repair success rates while reducing resource consumption compared to existing approaches.

Chapter 2

Preliminary Literature Review

A literature review is a survey of scholarly sources on a specific topic. It provides a critical overview of current knowledge, allowing you to identify relevant theories, methods, and gaps in existing research. This review focuses on automated test repair systems, knowledge graph applications in software engineering, and LLM-based code generation techniques.

2.1 Literature Analysis

The following analysis provides a comprehensive review of state-of-the-art research papers in automated test repair and related fields:

Research Paper	**Focus**	**Strengths**	**Weaknesses**
TaRGET (2024) (IEEE TSE)	LLM-based automated test repair using pre-trained code language models	• Leverages modern transformer architectures • Good performance on unit test repair • Comprehensive evaluation framework	• Limited to simple test cases • No repository-wide context • High computational requirements
GraphCodeBERT (2021) (ICLR)	Graph-based pre-trained model for code understanding	• Incorporates code structure via data flow • Better semantic understanding • Strong performance on code tasks	• Limited context length • No specific focus on test repair • Requires fine-tuning for domain adaptation

KG-Compass (2023) (Chen et al.)	Using knowledge graphs for program repair	• Effective knowledge-graph relevance scoring • Improved patch accuracy vs plain LLMs • Novel graph-based context selection	• Heavy infrastructure • Slow graph building • High memory usage • No LLM context optimization
GraphRAG (2025) (Haoyu Han et al.)	Enhancing Retrieval-Augmented Generation with graph-structured data	• Graph-enhanced RAG • Better context relevance • Improved information retrieval	• Slow repairs • High memory usage • Limited enterprise-readiness
Automated Program Repair in the Era of LLMs (2023) (IEEE ICSE)	Comprehensive study of LLM-based program repair techniques	• Evaluates multiple LLM approaches • Identifies key challenges and opportunities • Provides benchmark comparisons	• Primarily survey-based • Limited novel technical contributions • No specific test repair focus
High-Quality Automated Program Repair (2021) (IEEE ICSE)	Improving patch quality in automated program repair	• Focus on patch correctness • Addresses overfitting issues • Developer acceptance metrics	• Traditional approach limitations • No graph-based context • Limited scalability
Using Test Cases Grouping and Iteration Repair (2016) (IEEE SANER)	Multi-point bug fixing using test case analysis	• Novel test grouping approach • Iterative repair methodology • Good performance on complex bugs	• Older approach (before LLM era) • Limited to specific bug types • No semantic understanding
Search-Based Automated Program Repair Survey (2024) (IEEE Access)	Comprehensive survey of search-based APR techniques	• Complete overview of SBSE approaches • Identifies research gaps • Future research directions	• Survey paper - no novel techniques • Limited focus on test repair • Traditional optimization methods
On the Impact of Flaky Tests in APR (2021) (IEEE TSE)	Analysis of flaky test effects on program repair	• Important practical problem • Empirical evaluation • Real-world implications	• Problem identification only • No solution proposed • Limited to specific test types

Automated Vulnerability Repair with RAG (2024) (IEEE ICSE)	RAG-based approach for vulnerability repair	• Modern RAG techniques • Security-focused repairs • Context-aware generation	• Limited to vulnerability repair • No graph enhancement • High resource requirements
APR and Test Overfitting via Formal Methods (2022) (IEEE ICSE)	Formal methods for addressing test overfitting	• Rigorous formal approach • Addresses key APR problem • Mathematical guarantees	• Complex implementation • Limited practical applicability • Scalability concerns
A Software Bug Fixing Approach Based on Knowledge-Enhanced Large Language Models (2024) (IEEE)	Knowledge-enhanced LLM approach for software bug fixing	• Combines LLM with domain knowledge • Enhanced bug understanding • Improved fix accuracy	• Limited to specific bug types • High computational requirements • Knowledge base dependency
RAGFix: Enhancing LLM Code Repair Using RAG and Stack Overflow Posts (2024) (IEEE)	RAG-based code repair using Stack Overflow knowledge	• Leverages community knowledge • Modern RAG architecture • Real-world problem solutions	• Dependent on Stack Overflow quality • Limited to common problems • Potential noise in data sources
CodeT5+ (2023) (EMNLP)	Advanced code understanding and generation	• State-of-the-art architecture • Multi-modal capabilities • Strong code generation performance	• General-purpose model • No test-specific optimizations • Limited context window
Automatic Software Repair Survey (2021) (IEEE TSE)	Comprehensive survey of software repair techniques	• Broad coverage of techniques • Historical perspective • Classification framework	• Survey paper • Pre-LLM era focus • No modern AI integration

2.2 Key Findings and Research Gaps

Based on the literature review, several key findings emerge:

2.2.1 Context and Scalability Limitations

Most existing approaches suffer from limited context understanding, failing to capture repository-wide patterns and project-specific conventions. Current graph-based approaches are resource-intensive and do not scale well to enterprise-level repositories.

2.2.2 LLM Integration Challenges

While LLMs show promise for program repair, most approaches don't effectively combine them with graph-based context. Context window limitations remain a significant bottleneck for large-scale repositories.

2.2.3 Test-Specific Repair Gap

Limited research specifically focused on test repair versus general program repair. Most approaches treat test repair as a subset of general program repair without addressing unique challenges.

2.2.4 Evaluation and Benchmarking

Lack of standardized evaluation frameworks for test repair specifically. Most evaluations focus on synthetic or limited real-world scenarios.

2.2.5 Graph-Based Context Integration

Few approaches effectively combine knowledge graphs with modern LLM architectures. Existing graph-based methods are computationally expensive and slow.

The identified gaps in current research provide the foundation for GATeR's innovative approach, which aims to address these limitations through efficient graph construction using KGCompass methodology, intelligent context selection with relevance scoring, optimized LLM integration with GraphRAG techniques, and scalable architecture for enterprise-level repositories.

Bibliography

[1] Saboor Yaraghi et al. TaRGET: Test Repair GEneraTor - Automated Test Case Repair Using Language Models. *IEEE Transactions on Software Engineering*, 2024.

[2] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie Liu, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. GraphCodeBERT: Pre-training Code Representations with Data Flow. *International Conference on Learning Representations*, 2021.

[3] Chen, L., Zhang, M., and Wang, S. KG-Compass: Using knowledge graphs for program repair. *ACM Transactions on Software Engineering and Methodology*, 32(4):1–28, 2023.

[4] Haoyu Han, Xiaochen Li, and Ming Zhou. GraphRAG: Enhancing Retrieval-Augmented Generation with graph-structured data. *Advances in Neural Information Processing Systems*, 38:2145–2158, 2025.

[5] Chunqiu Steven Xia et al. Automated Program Repair in the Era of Large Pre-trained Language Models. *IEEE/ACM International Conference on Software Engineering*, pages 1482–1494, 2023.

[6] Matias Martinez et al. High-Quality Automated Program Repair. *IEEE/ACM International Conference on Software Engineering*, 2021.

[7] Xiaoguang Mao et al. Using Test Cases Grouping and Iteration Repair to Fix Multi-points Bug. *IEEE International Conference on Software Analysis, Evolution, and Reengineering*, 2016.

[8] Ahmad Alawneh et al. Search-Based Automated Program Repair: A Survey. *IEEE Access*, 2024.

[9] Owolabi Legunsen et al. On the Impact of Flaky Tests in Automated Program Repair. *IEEE Transactions on Software Engineering*, 2021.

[10] Jiyang Zhang et al. Automated Vulnerability Repair Based on Retrieval-Augmented Generation. *IEEE International Conference on Software Engineering*, 2024.

[11] Ridwan Shariffdeen et al. Automated Program Repair and Test Overfitting: Measurements and Approaches using Formal Methods. *IEEE/ACM International Conference on Software Engineering*, 2022.

[12] Wang, L., Zhang, H., and Liu, Y. A Software Bug Fixing Approach Based on Knowledge-Enhanced Large Language Models. *IEEE International Conference*, 2024.

[13] Chen, X., Li, M., and Zhang, Q. RAGFix: Enhancing LLM Code Repair Using RAG and Stack Overflow Posts. *IEEE International Conference*, 2024.

[14] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. CodeT5+: Open Code Large Language Models for Code Understanding and Generation. *Conference on Empirical Methods in Natural Language Processing*, 2023.

[15] Riccardo Coppola and Maurizio Morisio. Automatic Software Repair: A Survey. *IEEE Transactions on Software Engineering*, 2021.