Trustworthy AI-Assisted Programming: Detection and Repair of Unreliable Code

Jian Wang

Nanyang Technological University, Singapore
jian004@e.ntu.edu.sg

Abstract

The widespread adoption of AI-assisted coding tools has fundamentally transformed software development, enabling rapid code generation but simultaneously introducing new risks to software reliability. This thesis addresses the critical challenge of ensuring trustworthy AI-assisted programming through two complementary approaches: detecting AI-generated code and advancing automated program repair. I present four major contributions: (1) The first comprehensive empirical study of AI-generated code detection across 2.24 million samples with fine-tuning-based improvements, (2) Defects4C, the first large-scale executable C/C++ bug benchmark with 248 real-world bugs, (3) Novel automated repair methods combining deep learning and LLM-based approaches with extensive empirical evaluation, and (4) A semantic enhancement framework that incorporates execution traces to improve LLM reasoning for program repair. These contributions establish new foundations for trustworthy, semantically grounded automated program repair in the era of AI-assisted development.

Introduction

Software systems underpin critical infrastructures across finance, healthcare, transportation, and communication. The rapid adoption of AI-assisted coding tools like GitHub Copilot and GPT-based code generators has dramatically increased development velocity, with organizations reporting 30-42% productivity improvements. However, this acceleration introduces a fundamental challenge (Eiras et al. 2024): while AI tools increase code production speed, they simultaneously introduce new reliability risks through subtle bugs, security vulnerabilities, and compliance issues.

The core problem manifests in two dimensions. First, AI-generated code cannot be inherently trusted to meet production standards (Ferdaus et al. 2024). Multiple studies demonstrate that LLMs frequently generate code containing defects that may not be immediately apparent, alongside raising legal concerns regarding copyright and licensing. Second, the acceleration of development exacerbates maintenance costs. As AI tools enable faster code production, organizations face an expanding volume of potential defects, creating a paradox where initial productivity gains are offset by an increased maintenance burden.

Addressing these challenges requires coordinated action on two complementary fronts as illustrated Figure 1: detection and repair. Without reliable detection mechanisms, organizations cannot distinguish human-written code from machine-generated fragments to ensure quality or compliance (Mohamed et al. 2024). Once defects are identified, automated program repair (APR) is essential to mitigate the rising maintenance costs. However, current solutions in both areas face fundamental limitations, particularly in their ability to reason about the semantic behavior of programs.

This thesis investigates three interconnected problems. 1, we examine how to reliably distinguish AI-generated code from human-written software to enable targeted quality assurance. 2, we explore how to automatically generate correct patches for bug, specifically in C/C++ where existing benchmarks and methods are lacking. 3, we investigate how to improve the semantic understanding of Large Language Models to move beyond surface-level pattern matching and enable reliable, execution-aware program repair.

Current Progress

Detecting AI-Generated Code. To address the challenge of identification, (Wang et al. 2024a) conducted the first comprehensive empirical study evaluating thirteen AIGC detectors across a large-scale dataset of 2.24 million samples (1.08 million code-related, 1.16 million natural language) . The study revealed that existing detectors suffer significant accuracy degradation on code compared to text. This work proposed fine-tuning-based methods that achieved substantial improvements, increasing detection accuracy by up to 23.5% on code snippets and 15.8% on documentation.

Automated Patch Generation. To enable reliable repair for C/C++ systems, (Wang et al. 2024b) first constructed **Defects4C**, the first large-scale executable C/C++ bug benchmark curated from 9 million commits, containing 248 real-world bugs and 102 vulnerabilities with full test suites . Using this foundation, it developed a dual deep learning framework that couples BiLSTM-based fault localization with retrieval-augmented transformer patch generation. Additionally, (Wang et al. 2025b) conducted the first large-scale empirical evaluation of state-of-the-art LLMs for C/C++ repair . The results demonstrated that while fine-tuning yields

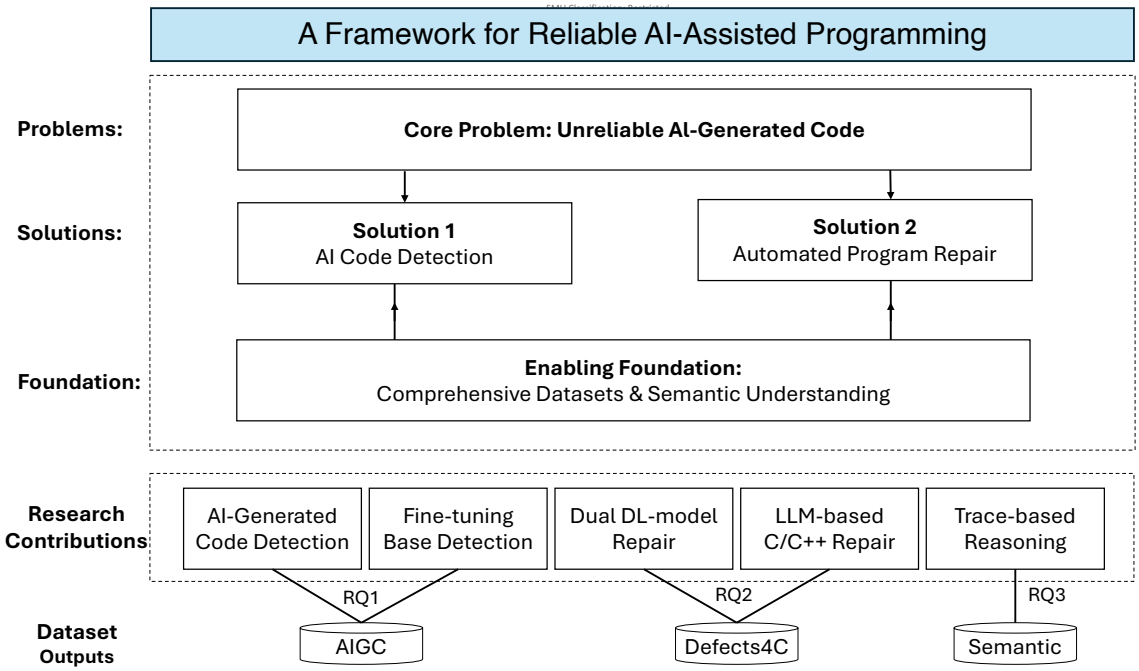

Figure 1: Overview of the thesis: addressing unreliable AI-generated code through detection and repair solutions, enabled by comprehensive datasets and semantic understanding.

improvements, performance significantly lags behind Java benchmarks, highlighting that statistical learning alone is insufficient for robust repair.

Semantic Enhancement for Repair. To bridge the gap in semantic reasoning, (Wang et al. 2025a) developed a semantic enhancement framework that incorporates dynamic program semantics—specifically execution traces—into both model training and inference . Extensive empirical evaluation demonstrates that this semantic-informed approach consistently improves model performance, achieving an average 12.3% improvement patch's correctness and 8.7% improvement generated functional correctness. The framework supports flexible integration strategies, including parameter-efficient fine-tuning, to enable practical deployment.

Future Directions

First, **detection methods** must evolve to handle the adversarial nature of rapidly advancing models, moving towards adaptive frameworks that utilize execution-based signals rather than surface patterns. Second, bridging the **semantic gap** in repair requires hybrid neurosymbolic architectures and scalable methods to incorporate runtime behavior without prohibitive computational overhead. Finally, the ultimate goal extends toward **verified AI development**, integrating formal verification guarantees with human-AI collaborative systems to ensure code correctness and maintainability.

References

Eiras, F.; Petrov, A.; Vidgen, B.; Schroeder, C.; Pizzati, F.; Elkins, K.; Mukhopadhyay, S.; Bibi, A.; Purewal, A.; Botos, C.; et al. 2024. Risks and opportunities of open-source generative AI. *arXiv preprint arXiv:2405.08597*.

Ferdaus, M. M.; Abdelguerfi, M.; Ioup, E.; Niles, K. N.; Pathak, K.; and Sloan, S. 2024. Towards trustworthy ai: A review of ethical and robust large language models. *arXiv preprint arXiv:2407.13934*.

Mohamed, Y. A.; Mohamed, A. H.; Kannan, A.; Bashir, M.; Adiel, M. A.; and Elsadig, M. A. 2024. Navigating the ethical terrain of ai-generated text tools: a review. *IEEE Access*.

Wang, J.; Liu, S.; Xie, X.; and Li, Y. 2024a. An Empirical Study to Evaluate AIGC Detectors on Code Content. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 844–856.

Wang, J.; Liu, S.; Xie, X.; Siow, J. K.; Liu, K.; and Li, Y. 2024b. RATCHET: Retrieval Augmented Transformer for Program Repair. In *Proceedings of the 35th International Symposium on Software Reliability Engineering (ISSRE)*, 427–438.

Wang, J.; Xie, X.; Hu, Q.; Liu, S.; and Li, Y. 2025a. Do Code Semantics Help? A Comprehensive Study on Execution Trace-Based Information for Code Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP*.

Wang, J.; Xie, X.; Hu, Q.; Liu, S.; Yu, J.; Kong, J.; and Li, Y. 2025b. Defects4C: Benchmarking Large Language Model Repair Capability with C/C++ Bugs. In *ASE*.