Graph-Grounded Verification of Large Language Model Reasoning in Code Analysis

Selen Erdoğan[#1]

[#]*Computer Engineering, 210104004131, Gebze Technical University Kocaeli*

[1]selenerdogan2019@gtu.edu.tr

***Abstract*— The integration of Large Language Models (LLMs) into software development workflows has accelerated productivity but introduced significant challenges regarding reliability and trustworthiness. These models can generate plausible yet factually incorrect explanations about code structure and logic, a phenomenon known as "hallucination." This project proposes the development of an automated pipeline to verify the factual accuracy of LLM-generated reasoning about Python source code. By parsing code into structural graphs the system will establish a verifiable "ground truth." It will then systematically map natural language claims from an LLM's reasoning to this graph and traverse it to validate each logical step.**

***Keywords*— Large Language Models, Code Analysis, Program Comprehension, Graph Theory, Natural Language Processing**

I. Introduction

The rise of powerful Large Language Models (LLMs) has transformed software development by enabling automated code generation, debugging, and explanation. Developers increasingly depend on these models to understand and maintain complex codebases. However, LLMs often produce hallucinations—statements that sound correct but are factually wrong. In code analysis, they may invent functions, misstate dependencies, or describe incorrect data flows. These errors erode trust and can mislead developers. Therefore, there is a growing need for automated systems that can verify an LLM's reasoning against the actual structure of the code.

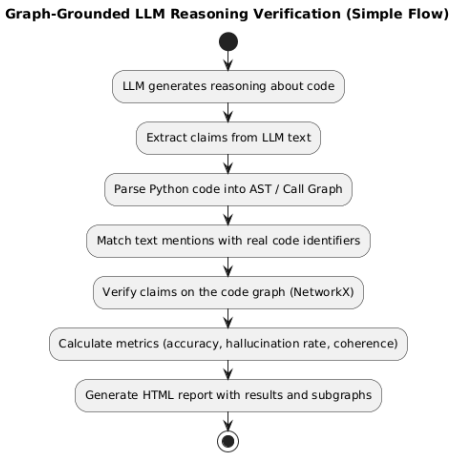

Figure 1. Main Workflow of Project

The main objective of this project is to verify large language models' reasoning about code by grounding their statements in the actual program structure. LLMs often produce confident but incorrect explanations, and detecting such errors is essential for building trustworthy AI systems. It increases the reliability of LLM outputs, facilitates error analysis, and provides a foundation for future evaluation frameworks. The project will involve code graph extraction, claim mapping, graph-based verification, metric computation, and HTML reporting. The expected result is a system that reliably verifies LLM reasoning about code and reports incorrect reasoning steps.

II. Methodology

The project will be executed by developing a modular pipeline composed of four main components. The methodology integrates principles from compiler design, graph theory, and natural language processing to systematically evaluate the

reasoning consistency of large language models over Python code.

A. Code Representation as Structural Graphs

The foundation of the verification system will be a reliable structural representation of the source code, serving as the reference "ground truth."

1. Parsing to Abstract Syntax Trees: The input Python projects will first be parsed into hierarchical tree structures that represent the syntactic composition of the code.

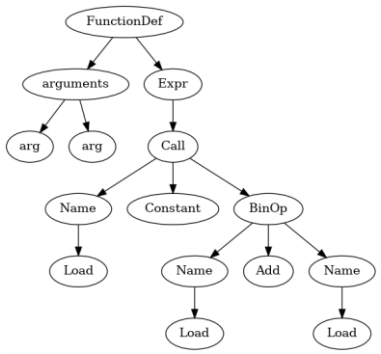

Figure 2. AST Trees Example

2. Graph Construction: From these trees, higher-level graphs such as call graphs and data-flow graphs will be derived to capture functional dependencies and data propagation patterns within the program. These graphs will constitute the structural layer upon which reasoning verification will be performed.

B. Entity Grounding and Mention Mapping

In this module, textual entities mentioned in the model's reasoning traces (functions, variables, classes, etc.) will be grounded to corresponding elements in the structural graphs. Both rule-based parsing techniques and semantic similarity measures will be combined to achieve robust mappings between textual mentions and actual code identifiers. Each mention will be validated by comparing its context and linguistic embedding with graph metadata, ensuring referential accuracy.

C. Reasoning Verification via Graph Traversal

Once textual entities have been grounded, the logical relations or dependencies claimed by the model (e.g., "function A calls function B") will be verified against the structural graphs. Graph traversal algorithms will be applied to determine whether these dependencies genuinely exist within the codebase. This step will allow the system to distinguish between reasoning steps that are structurally valid and those that are hallucinated or unsupported by the actual program.

D. Evaluation Metrics and Visualization

The performance of the reasoning verification process will be quantified using a set of reproducible metrics, including:

- Hallucination Rate: Proportion of invalid or non-existent structure references.
- Coverage: Degree to which actual code structures are represented in the reasoning chain.
- Step Validity: Logical correctness of each reasoning step.
- Chain Coherence: Consistency across consecutive reasoning steps.

Results will be summarized and visualized through an interactive reporting interface. Generated dashboards will include subgraph visualizations, metric tables, and comparative analyses to support reviewer calibration and reproducibility.

III. Conclusions

This project aims to develop a graph-grounded framework for verifying the reasoning of large language models in code analysis. By grounding model outputs in the actual program structure, the system will help identify hallucinations, measure reasoning validity, and improve transparency. The expected outcome is a reproducible evaluation tool that enhances trust and accountability in LLM-based software reasoning.

References

[1] Python Software Foundation. (2023) The Python ast library – Abstract Syntax Trees. [Online]. Available: https://docs.python.org/3/library/ast.html.

[2] Zhao *ve ark., Verifying Chain-of-Thought Reasoning via Its Computational Graph*, 2025

[3] Sistla *ve ark., Towards Verified Code Reasoning by LLMs*, 2025

[4] Ling *ve ark., Deductive Verification of Chain-of-Thought Reasoning*, NeurIPS 2023

[5] Vacareanu *ve ark., General Purpose Verification for CoT Prompting*, 2024