Dynamic Code Execution Traces and Static Analysis in LLaVul for Vulnerability Classification

Assignee Research

May 31, 2026

Abstract

This report synthesises findings from 14 peer-reviewed papers addressing the following research question: How does the integration of dynamic code execution traces with static analysis visualizations in LLaVul impact its vulnerability classification accuracy on the Big-Vul dataset compared to static-only. Increasing complexity in software systems places a growing demand on reasoning tools that unlock vulnerabilities manifest in source code. Many current approaches focus on vulnerability analysis as a classifying task, oversimplifying the nuanced and context-dependent real-world. 12 claims were extracted from source literature; 0 were independently verified against retrieved documents. An automated multi-reviewer quality assessment produced a score of 3.1/10. This report is a machine-generated literature synthesis and does not constitute original research.

1 Introduction

This paper examines: LLaVul: A Multimodal LLM for Interpretable Vulnerability Reasoning about Source Code. Research question: How does the integration of dynamic code execution traces with static analysis visualizations in LLaVul impact its vulnerability classification accuracy on the Big-Vul dataset compared to static-only multimodal training?.

2 Methodology

Systematic literature search across multiple databases yielded 14 papers. Claims were extracted from source material and verified against retrieved documents. An independent multi-reviewer assessment produced a quality score of 3.1/10.

3 Results

14 papers retrieved. 12 claims extracted; 0 independently verified. Quality review score: 3.1/10.

4 Limitations

This report is a machine-generated literature synthesis and does not constitute original research. Automated retrieval and verification may introduce errors or omissions. Review scores reflect automated assessment, not human peer review. Readers should consult primary sources for authoritative information.

5 Extracted Claims

Claim	Verified	Confidence
The LLaVul-QA fine-tuning dataset was constructed using approximately 160,000 vulnerable functions collected from the CV	×	0.04
C, C++, and JavaScript were the most common programming languages containing vulnerabilities in the collected dataset.	×	0.05
The LLaVul code encoder utilizes CodeSage-small, a transformer-based model with 6 layers and 130 million parameters.	×	0.04
The LLaVul LLM encoder uses the Vicuna 1.5 model with 7 billion parameters.	×	0.04
LLaVul achieved a BLEU-4 score of 0.168 on the LLaVul test set.	×	0.02
LLaVul achieved a ROUGE-L score of 0.350 on the LLaVul test set.	×	0.02
LLaVul achieved a BERT Score of 0.910 on the LLaVul test set.	×	0.02
CodeT5-base-multi-sum achieved a BERT Score of 0.821, ranking second among the tested models.	×	0.01
The estimated training time for the LLaVul pre-training stage is approximately 5 days.	×	0.02
The estimated training time for the LLaVul fine-tuning stage ranges from 12 to 72 hours.	×	0.04
The generation of QA pairs for the fine-tuning dataset took multiple weeks using resources including 4 A100s, V100s, and	×	0.04
For the vulnerability CVE-2017-12595, LLaVul identified the impact as a denial of service via buffer under-read and appl	×	0.03

References

- http://arxiv.org/abs/2312.05434v1
- http://arxiv.org/abs/2509.17337v1
- http://arxiv.org/abs/2601.08691v1