USENIX Security '25 Artifact Appendix: LLMxCPG: Context-Aware Vulnerability Detection Through Code Property Graph-Guided Large Language Models

Ahmed Lekssays[1*], *Hamza Mouhcine*[1*], *Khang Tran*[2], *Ting Yu*[3], *Issa Khalil*[1]
[1]Qatar Computing Research Institute, [2]New Jersey Institute of Technology,
[3]Mohamed bin Zayed University of Artificial Intelligence
{alekssays, hmouhcine, ikhalil}@hbku.edu.qa,
kt36@njit.edu, ting.yu@mbzuai.ac.ae
* *Joint first authors with equal contribution*

A Artifact Appendix

A.1 Abstract

LLMxCPG is a novel framework that integrates Code Property Graphs (CPG) with Large Language Models (LLM) for robust vulnerability detection. Our CPG-based slice construction technique significantly reduces code size, ranging from 67.84% to 90.93%, while preserving vulnerability-relevant context. This approach enables a more concise and accurate representation of code snippets, facilitating the analysis of larger code segments, including entire projects. Empirical evaluation demonstrates LLMxCPG's effectiveness on verified datasets, achieving 15-40% improvements in F1-score over state-of-the-art baselines. Furthermore, LLMxCPG maintains high performance across both function-level and multi-function codebases and exhibits robust detection efficacy under various syntactic code modifications.

A.2 Description & Requirements

This section lists all the information necessary to recreate the experimental setup used to run the artifact. Where it applies, the minimal hardware and software requirements to run the artifact are specified.

A.2.1 Security, privacy, and ethical concerns

Our vulnerability detection research adheres to responsible disclosure protocols and established security research guidelines. We carefully balance the benefits of identifying security weaknesses against potential risks, and all discovered vulnerabilities are reported through appropriate channels, allowing sufficient time for patches before public disclosure. We maintain strict confidentiality throughout the research process to ensure our methods do not compromise system integrity or user privacy.

A.2.2 How to access

The source code, fine-tuned models, and testing datasets used in this study are publicly available to the community to foster research in this field. They can be accessed at:

- GitHub: https://github.com/qcri/1lmxcpg
- Zenodo: https://zenodo.org/records/15614095

A.2.3 Hardware dependencies

The fine-tuning of the models was performed on an NVIDIA A100-80GB GPU. While the fine-tuned model for query generation (LLMxCPG-Q) was limited to a 32K token context due to computational constraints, the base model supports a 128K token context, with versions supporting up to 1M tokens being available if substantial computational resources are provided. For optimal performance in evaluation, access to a GPU, ideally an NVIDIA A100 with at least 80GB of memory, is recommended.

A.2.4 Software dependencies

The artifact requires the following software components:

- Operating System: Ubuntu 22.04
- Joern: An open-source static analysis tool. A cluster of Joern servers deployed using Docker containers is used for parallel processing.
- LLaMA-Factory framework
- vLLM inference library
- Python 3.x and associated packages listed in the 'requirements.txt' file within the repository.

A.2.5 Benchmarks

The evaluation utilized several datasets:

- **Training Datasets: (both for query generation and vulnerability detection)**
  - FormAI-v2
  - PrimeVul
- **Generalizability/Test Datasets:**
  - SVEN
  - Repos Vul
  - 2025 Post-Knowledge-Cutoff (PKCO-25) CVEs dataset (compiled from CVEs published in 2025)

A.3 Set-up

This section includes all the installation and configuration steps required to prepare the environment for the evaluation of the artifact.

A.3.1 Installation

To install and set up the artifact, follow these steps:

1. Clone the LLMxCPG repository from GitHub: `git clone https://github.com/qcri/llmxcpg.git`
2. Navigate to the cloned directory: `cd llmxcpg`
3. Install the required Python dependencies. A 'requirements.txt' file is provided in 'requirements.txt': `pip install -r requirements.txt`
4. Set up Joern: The paper mentions deploying Joern servers using Docker containers for parallel processing. Follow the official Joern documentation for installation and setup within a Docker environment.

A.3.2 Basic Test

To run a simple functionality test of the LLMxCPG system (specifically the detection model), you can execute the 'detect_inference.py' script with a sample code snippet.

1. Run the inference script: `python inference/detect_inference.py {dataset}` where *dataset* is one of the following: *primevul*, *formai*, *sven*, *reposvul*, and *pkco*.
2. Expected output: The model should classify the code as either "VULNERABLE" or "SAFE". For a vulnerable input, the expected output is "VULNERABLE". Then, the Accuracy, Precision, Recall, and F1-Scores are calculated.

A.4 Evaluation workflow

This section includes all the operational steps and experiments which must be performed to evaluate if your artifact is functional and to validate your paper's key results and claims.

A.4.1 Major Claims

(C1): *LLMxCPG achieves high performance on function-level vulnerability detection across diverse datasets.* This is proven by the experiments (E1) and (E2) described in Section 4.3.2 "Function-level Vulnerability Detection" and Section 4.4.1 "Function-level Vulnerability Detection" (Generalizability subsection) , whose results are illustrated/reported in Table 3 and Table 5. Specifically, LLMxCPG achieves up to 0.8146 Accuracy and 0.8075 F1-score on the FormAI dataset , and a remarkable 20% improvement in accuracy over competing approaches on the SVEN dataset.

(C2): *LLMxCPG demonstrates robust generalization capabilities on project-level codebases and novel vulnerabilities that emerged after its knowledge cutoff.* This is proven by the experiment (E3) described in Section 4.4.2 "Project-level Vulnerability Detection" , whose results are illustrated/reported in Table 7. LLMxCPG achieved an F1-score of 0.610 and Accuracy of 0.634 on the Repos Vul dataset, and an F1-score of 0.617 and Accuracy of 0.600 on the 2025 Post-Knowledge-Cutoff (PKCO-25) dataset.

A.4.2 Experiments

(E1): *Function-level Vulnerability Detection Performance on Trained Datasets [30 human-minutes + 1 compute-hour]:*
How to: This experiment evaluates LLMxCPG's performance on function-level code snippets using the FormAI and PrimeVul datasets, which were part of the training data.
Preparation: Ensure the LLMxCPG environment is set up and the fine-tuned LLMxCPG-D model (based on QwQ-32B-Preview) is loaded. Use the test splits of the FormAI and PrimeVul datasets.
Execution: Run the LLMxCPG inference pipeline on the test sets of FormAI and PrimeVul by running `python inference/detect_inference.py primevul` and `python inference/detect_inference.py formai`.
Results: The expected outcome is high performance in vulnerability detection, as shown in Table 3. For FormAI, expected Accuracy is 0.8146 and F1-score is 0.8075. For PrimeVul, expected Accuracy is 0.7250 and F1-score is 0.6206.

(E2): *Generalizability to Unseen Function-level Datasets [30 human-minutes + 1 compute-hour]:*

How to: This experiment assesses the generalizability of LLMxCPG by evaluating its performance on the SVEN dataset, which was not part of the training data.
Preparation: Ensure the LLMxCPG environment is set up and the fine-tuned LLMxCPG-D model is loaded. Use the SVEN dataset for testing.
Execution: Run the LLMxCPG inference pipeline on the SVEN dataset: `python inference/detect_inference.py sven`
Results: The expected outcome is that LLMxCPG significantly outperforms the baselines, achieving a substantial improvement in accuracy. As shown in Table 5, LLMxCPG is expected to achieve an Accuracy of 0.6020 and an F1-score of 0.7048.

(E3): *Project-level Vulnerability Detection and Generalization to Post-Knowledge-Cutoff CVEs [45 human-minutes + 1.5 compute-hour]:*
How to: This experiment evaluates LLMxCPG's performance on complex, project-level code snippets from the Repos Vul dataset and its generalization to newly published CVEs (PKCO-25 dataset).
Preparation: Ensure the LLMxCPG environment is set up and the fine-tuned LLMxCPG-D model is loaded. Use the sampled Repos Vul dataset and the 2025 Post-Knowledge-Cutoff (PKCO-25) CVEs dataset for testing.
Execution: Run the LLMxCPG inference pipeline on both the Repos Vul and PKCO-25 datasets with `python inference/detect_inference.py reposvul` and `python inference/detect_inference.py pkco`.
Results: The expected outcomes are promising results despite the complexity of the datasets. As shown in Table 7, for Repos Vul, expected Accuracy is 0.634 and F1-score is 0.610. For PKCO-25, expected Accuracy is 0.600 and F1-score is 0.617.

A.5 Notes on Reusability

All source code and datasets used in this study are open-source, supporting reproducibility, transparency, and further research in the domain of software security. The artifact provides a valuable framework for future research by enabling the analysis of large code segments through its CPG-based slice construction technique. Additionally, LLMxCPG's ability to precisely identify and isolate security-critical changes between vulnerable and patched versions makes it particularly valuable for understanding vulnerability fixes and generating high-quality training data for vulnerability detection models. This functionality can be leveraged by other researchers to compile large, high-quality datasets for training vulnerability detection models in the future.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.