Evaluating Reliability of Static Analysis Results Using Machine Learning

Author: Ing. Tomáš Beránek **Supervisor: prof. Ing. Tomáš Vojnar, Ph.D.**

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

The presented system was developed with financial support from Red Hat.

1. Motivation

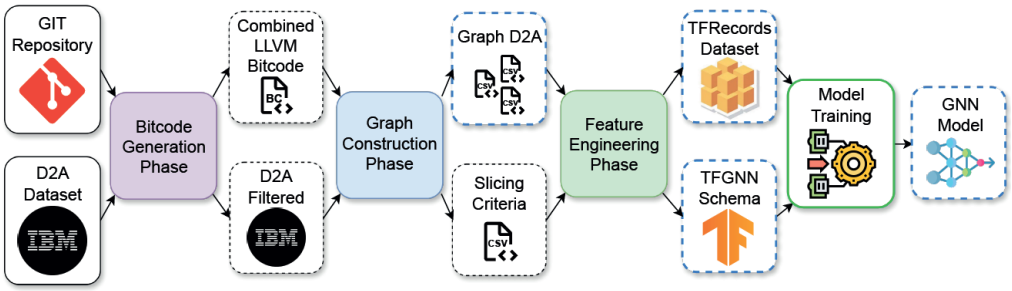
Static analysis is a widely adopted technique in software development for identifying various types of errors and security vulnerabilities. Its strength lies in the ability to consider all potential execution paths, allowing the detection of rare and otherwise elusive issues that might be missed during traditional testing processes. This is particularly crucial in **high-stakes domains such as aerospace, medicine, and cybersecurity**, where even minor software defects can lead to catastrophic failures or security breaches. However, a significant limitation of static analysis is its tendency to generate a **large volume of false positives** (i.e., false alarms).

This thesis focuses on **Meta Infer**, a static analysis tool used by companies such as Meta, Microsoft, Amazon, and Mozilla. Infer is known to produce **over 95 % false positives** [1]. The prevalence of these false positives **often results in developers disregarding static analysis results**, thereby undermining its practical utility, even in critical applications.

The objective of this thesis is to leverage the power of AI to enhance the effectiveness of Meta Infer by introducing a ranking system for the reported errors, prioritizing those with a higher probability of being true positives (i.e., actual issues and not false alarms).

2. The Reports Ranking System

The **reports ranking system** utilizes **Graph Neural Networks** (GNNs). The **D2A dataset** [5], which contains labeled reports from Infer, serves as a foundation for this work. To convert D2A into a graph-based format, a novel **Training Pipeline** was developed:



The pipeline consists of three main phases:

- **Bitcode Generation**: Each D2A sample is recompiled and LLVM bitcode is generated.
- **Graph Construction**: For each sample (LLVM bitcode), a raw ECPG graph is constructed.
- **Feature Engineering**: Feature engineering is applied to each raw ECPG graph, transforming it into a novel graph format—**Extended Code Property Graph**—which extends existing CPGs [4] with additional information such as data types, call graphs, etc.

At the end of the pipeline, the GNN models, which are constructed from MtAlbis layers—a modification of standard **VanillaMPNN layers** [2]—are trained.

To facilitate the practical use of these models, an **Inference Pipeline** was also developed. This pipeline can connect to a running compilation process of C (and a subset of C++) programs, to **automatically trigger Infer analysis, ECPGs construction, and model inference.**

3. Experimental Evaluation

For the experimental evaluation, three models with the best validation AUROCC were selected, along with ensemble models composed of the top 3 and top 6 models. For comparison, we chose the models **vote** [5], **C-BERT** [3], and the state-of-the-art model **vote-new** [3] developed by the authors of the D2A dataset (IBM Research). AUROCC was selected as the comparison metric, as it was the only common metric since all existing models are closed source.

The experiments show that the **developed GNN models are capable of matching state-of-the-art models**, and in the case of the NGINX project, even surpassing them. A **major drawback** of the developed models (as well as all compared existing models) is the **lack of functionality for cross-analysis** (i.e., training on one project and inference on another). However, the results show that the **models can be used effectively for self-analysis** (i.e., training on the same project where inference is performed) on projects with sufficient history for dataset generation.

Comparison metric: AUROCC on test data.

Model	httpd	libtiff	nginx
vote	0.77	0.89	0.77
c-bert	0.82	0.94	0.89
vote-new	0.90	0.98	0.93
Model 8	0.80	0.95	0.94
Model 10	0.79	0.91	0.91
Model 13	0.74	0.87	0.83
3-soft-vote	0.80	0.96	0.95
6-soft-vote	0.83	0.96	0.94

4. Contributions

- An open-source **Training Pipeline** that transforms D2A from a textual to a graph format.
- The design of a **novel graph format — Extended Code Property Graphs**, which experiments have shown to be a highly effective representation of source code.
- **Graph D2A** — D2A transformed into the ECPGs, which can support further research in this field. Graph D2A has been uploaded to the Zenodo open repository.
- Trained open-source **GNN models** that are capable of competing with closed-source state-of-the-art models. **The models can be used for error detection even without a static analyzer.**
- An open-source **Inference Pipeline** that enables fully automated Infer analysis, generation of ECPGs, and model inference during the compilation of any C (and a subset of C++) program.

5. References

[1] Beránek, T. Practical Application of Facebook Infer on Systems Code. Brno, CZ, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Available at: https://www.fit.vut.cz/study/thesis/24187/.

[2] Ferludin, O.; Eigenwillig, A.; Blais, M.; Zelle, D.; Pfeifer, J. et al. Tf-gnn: Graph neural networks in tensorflow. ArXiv preprint arXiv:2207.03522, 2022.

[3] Pujar, S.; Zheng, Y.; Buratti, L.; Lewis, B.; Chen, Y. et al. Analyzing source code vulnerabilities in the D2A dataset with ML ensembles and C-BERT. Empirical Software Engineering. Springer, 2024, vol. 29, no. 2, p. 48. Available at: https://link.springer.com/article/10.1007/s10664-023-10405-9.

[4] Yamaguchi, F.; Golde, N.; Arp, D. and Rieck, K. Modeling and discovering vulnerabilities with code property graphs. In: IEEE. 2014 IEEE Symposium on Security and Privacy. 2014, p. 590–604.

[5] Zheng, Y.; Pujar, S.; Lewis, B.; Buratti, L.; Epstein, E. et al. D2A: A dataset built for ai-based vulnerability detection methods using differential analysis. In: IEEE. 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). 2021, p. 111–120.