Filtering False Positives from Static Analysers Using Graph Neural Networks

Author: Bc. Tomáš Beránek

Supervisor: prof. Ing. Tomáš Vojnar, Ph.D.

Motivation

Static analysis is commonly used in software development to detect vulnerabilities and errors, leveraging its ability to consider all possible program paths and uncover even rarely manifesting errors missed by tests. However, its major drawback is the high number of false positives. This work focuses on **Meta Infer**, a static analyzer whose results contain over 95 % false positives. The frequent need to verify these false positives often leads developers to disregard the results of static analysis. The aim of this project is to enhance the utility of static analysis by **ranking the errors** detected by Infer based on their likelihood of being real.

Proposed Solution Overview

The proposed system uses deep **graph neural networks** (GNNs) for **ranking Infer outputs,** utilizing the D2A dataset, which is not originally in graph format. The project involves creating a **Training Pipeline** to develop **Graph D2A**—a graph version of the D2A dataset for GNN training. It also includes **trained models** for ranking Infer outputs and an **Inference Pipeline** to generate graphs from real-world programs using Infer outputs, enabling fully automatic operation and ranking of reports for real-world C programs.

Why use GNNs?

- **Code properties**, such as syntax or control flow, are **best expressed with graphs**.
- Graphs like AST and DFG are essential for various compiler tasks, proving their usefulness.
- **GNNs achieve top results in tasks involving error and vulnerability detection.**
- GNNs handle **variable input sizes**, which can pose problems for other NN architectures.
- Numerous tools exist for transforming source code into graph formats.

Why represent source code with ECPGs?

- GNNs are trained on **Extended Code Property Graphs** (ECPG).
- CPGs are **commonly used for vulnerability detection** tasks.
- **ECPGs enhance CPGs by incorporating data types, Call Graphs, and more.**
- Various tools can generate CPG graphs from various languages.

Training Pipeline

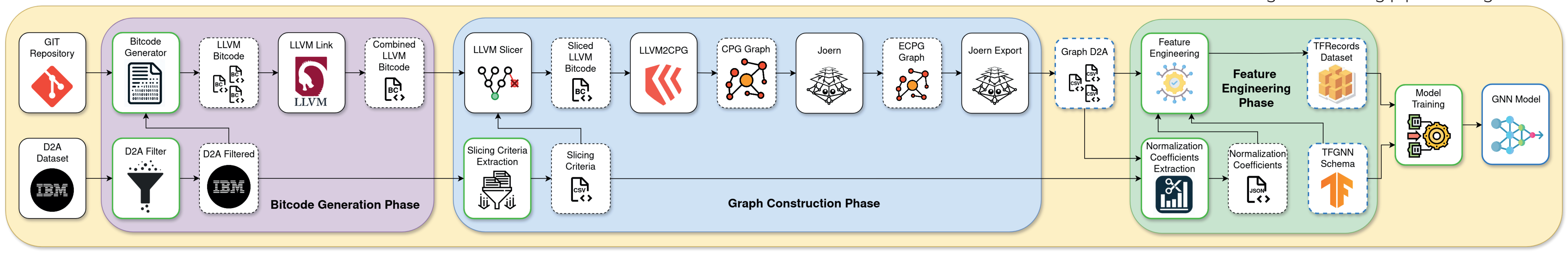


Figure 1: Training pipeline diagram.

Inference Pipeline

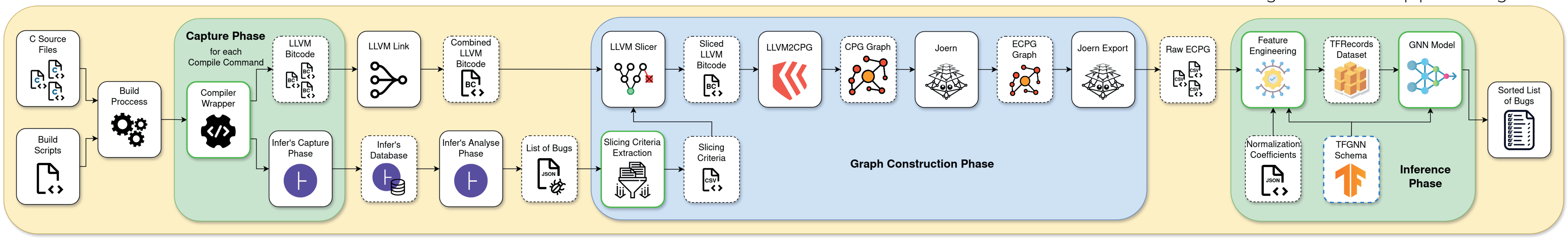


Figure 2: Inference pipeline diagram.

GNN Architecture

- The architecture of the best-performing model includes **Albis** GNN layers.
- The "head" of the model combines outputs from GNN layers with **context features**.
- Node features of the input graph are used to initialize hidden states.
- The model operates over **heterogeneous multigraphs**, specifically ECPGs.

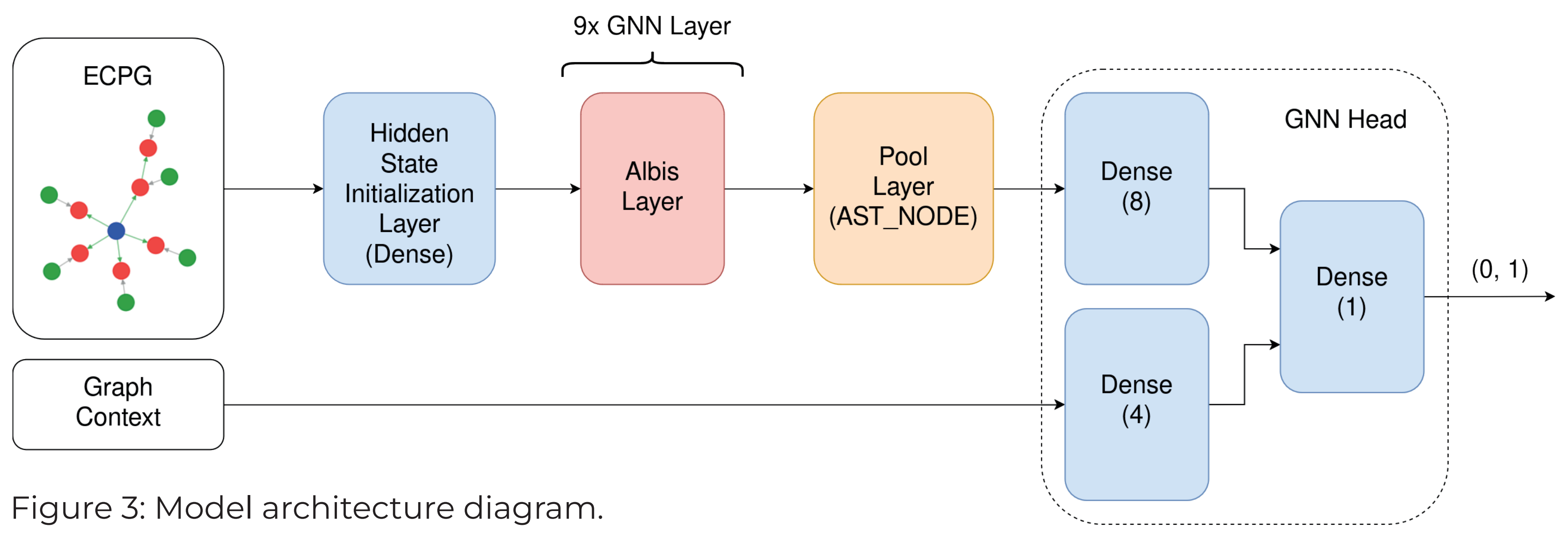


Figure 3: Model architecture diagram.

Experimental Evaluation

- The models are currently in the phase of architecture selection and hyperparameter tuning.
- Nevertheless, results from the currently best model, whose architecture is described above, are available.
- This model was trained on the **httpd**, **libtiff**, and **nginx** projects.
- Results were generated using test data from these same projects, thus this constitutes a form of **self-analysis**.
- A realistic use-case scenario for this test data would look as follows:
  - Take, for example, 5 % (182 samples) of the top samples:
    - (**without ranking**) only 5 samples would be TP,
    - (**with ranking**) 29 samples would be TP, which is ~6x increase.

Figure 4:

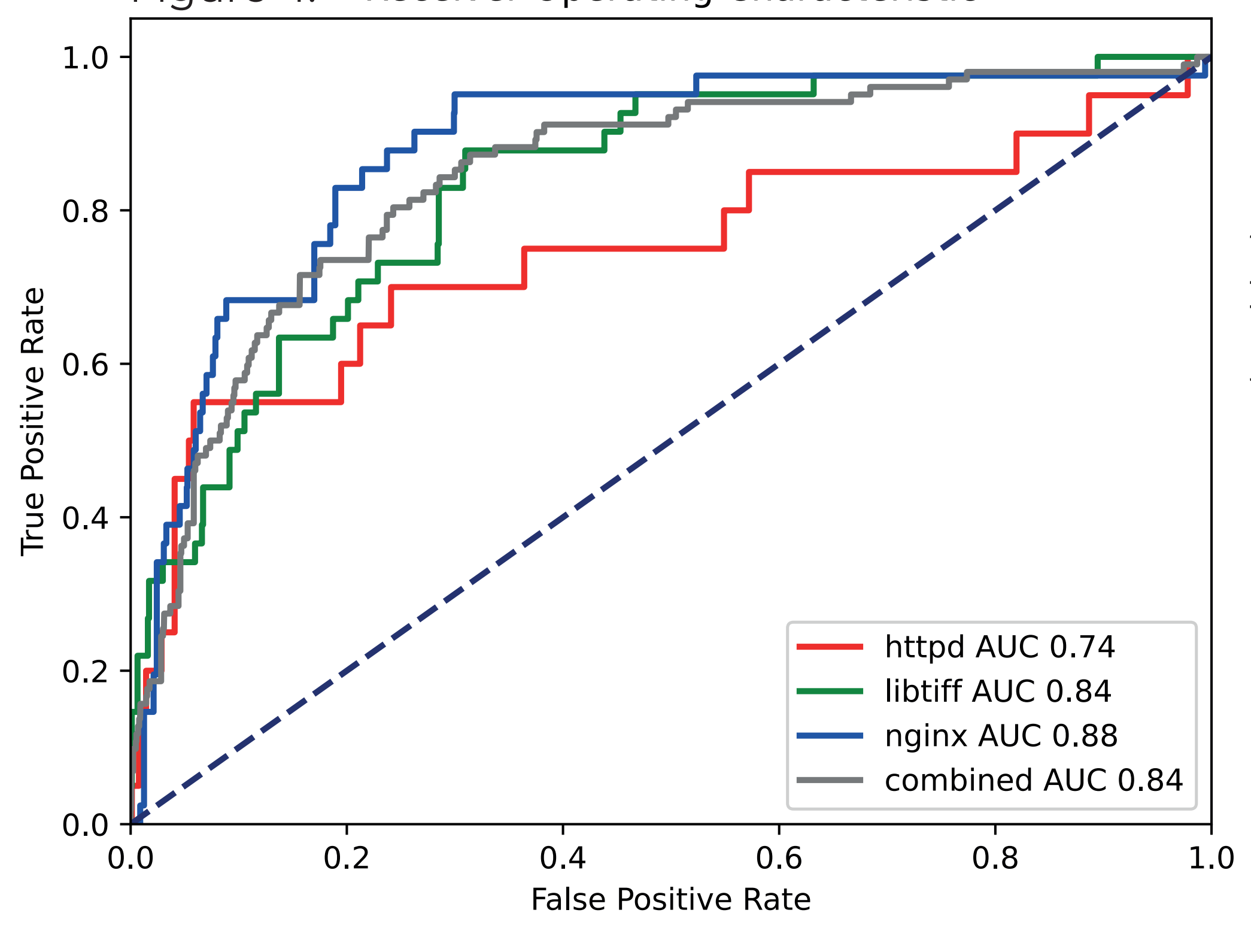
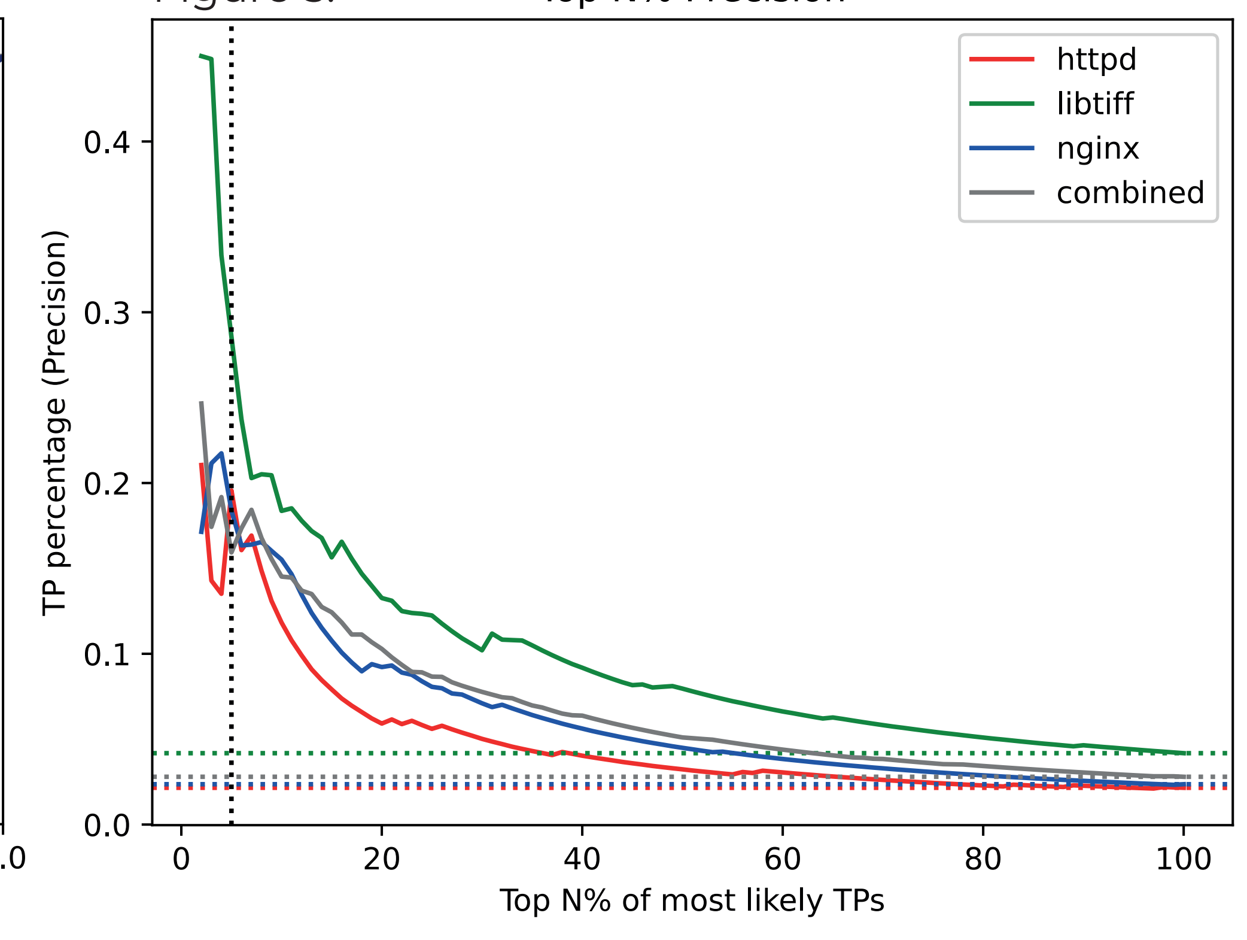


Figure 5:



Email: xberan46@stud.fit.vutbr.cz

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

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Red Hat

Excel@FIT 2024