# *LineVul: A Transformer-based Line-Level Vulnerability Prediction*

**Accepted at 2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)**

Michael
Fu

@Michael84437381

https://michaelfu1998-create.github.io/

Kla
Tantithamthavorn

@klainfo

http://chakkrit.com

MONASH University

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;
using namespace cv;

int main(int, char** argv)
{
    // Load the image
    Mat src = imread(argv[1]);

    // Check if everything was fine
    if (!src.data)
        return -1;

    // Show source image
    imshow("Source Image", src);

    // Change the background from white to black, since that will help later to
 extract
    // better results during the use of Distance Transform
    for( int x = 0; x < src.rows; x++ ) {
      for( int y = 0; y < src.cols; y++ ) {
          if ( src.at<Vec3b>(x, y) == Vec3b(255,255,255) ) {
            src.at<Vec3b>(x, y)[0] = 0;
            src.at<Vec3b>(x, y)[1] = 0;
            src.at<Vec3b>(x, y)[2] = 0;
          }
        }
    }

    // Show output image
    imshow("Black Background Image", src);
```

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;
using namespace cv;

int main(int, char** argv)
{
    // Load the image
    Mat src = imread(argv[1]);

    // Check if everything was fine
    if (!src.data)
        return -1;

    // Show source image
    imshow("Source Image", src);

    // Change the background from white to black, since that will help later to
 extract
    // better results during the use of Distance Transform
    for( int x = 0; x < src.rows; x++ ) {
      for( int y = 0; y < src.cols; y++ ) {
          if ( src.at<Vec3b>(x, y) == Vec3b(255,255,255) ) {
            src.at<Vec3b>(x, y)[0] = 0;
            src.at<Vec3b>(x, y)[1] = 0;
            src.at<Vec3b>(x, y)[2] = 0;
          }
        }
    }

    // Show output image
    imshow("Black Background Image", src);
```

# Challenges in Vulnerability Detection Practices

Software Vulnerabilities are costly and prevalent

But hard to detect and prevent

Static Analysis Tools have been proposed

But they are still inaccurate

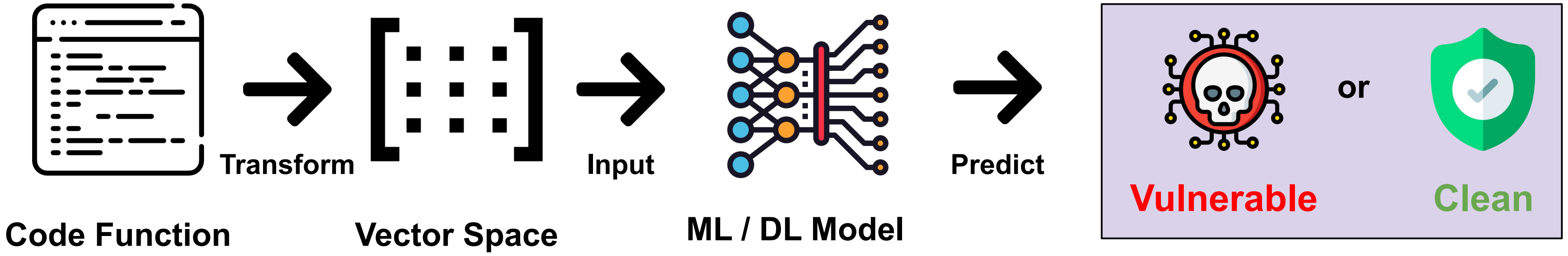
Within an project consists of millions lines of code

Security analysts may spend a huge amount of time

in order to locate the exact vulnerabilities

# Vulnerability Prediction Model: An Overview

## A model to predict if a function is a vulnerable function or not

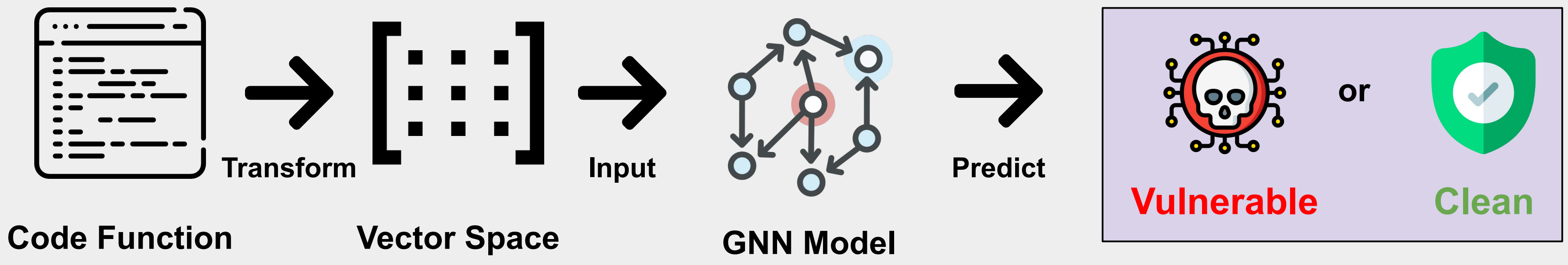


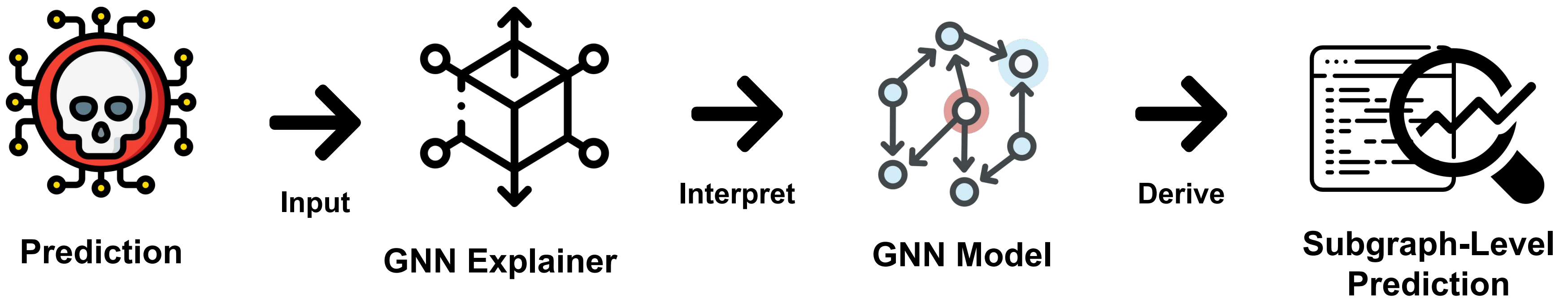
Now Security Analysts are able to locate which function is a potential vulnerable function

However, a vulnerable function may still contain many lines of code for Security Analysts to inspect

# IVDetect: A Fine-Grained Vulnerability Prediction Approach

## *A Function-Level Vulnerability Prediction Model*





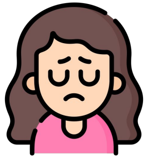
Now Security Analysts are able to locate which part of a function is a potential vulnerable function

# Example Subgraph-Level Prediction of IVDetect

## An example prediction of IVDetect using real world data

// Subgraph-Level Vulnerability Predictions by IVDetect

| third_party/WebKit/Source/core/frame/ImageBitmap.cpp<br>https://github.com/chromium/chromium/commit/d59a4441697f6253e7dc3f7ae5caad6e5fd2c778 | | IVDetect | Ground truth |
|---|---|---|---|
| 224 | `static sk_sp<SkImage> unPremulSkImageToPremul (SkImage* input) {` | 0 | 0 |
| 225 | `    SkImageInfo info = SkImageInfo::Make(input->width(), input->height(),` | 0 | 0 |
| 226 | `        kN32_SkColorType, kPremul_SkAlphaType);` | 0 | 0 |
| 227 | `    RefPtr<Uint8Array> dstPixels = copySkImageData(input, info);` | 0 | 0 |
| 228 | `    if (!dstPixels)` | 1 | 0 |
| 229 | `        return nullptr;` | 0 | 0 |
| 230 | `    return newSkImageFromRaster(` | 1 | 0 |
| 231 | `      info, std::move(dstPixels),` | 1 | 0 |
| 232 | `      static_cast<size_t>(input->width()) * info.bytesPerPixel());` | 1 | 1 |
| 233 | `}` | 0 | 0 |



Security Analysts still need to investigate the whole subgraph pattern to locate the exact vulnerable line

# 3 Limitations of the IVDetect Approach

IVDetect only trained on project-specific data

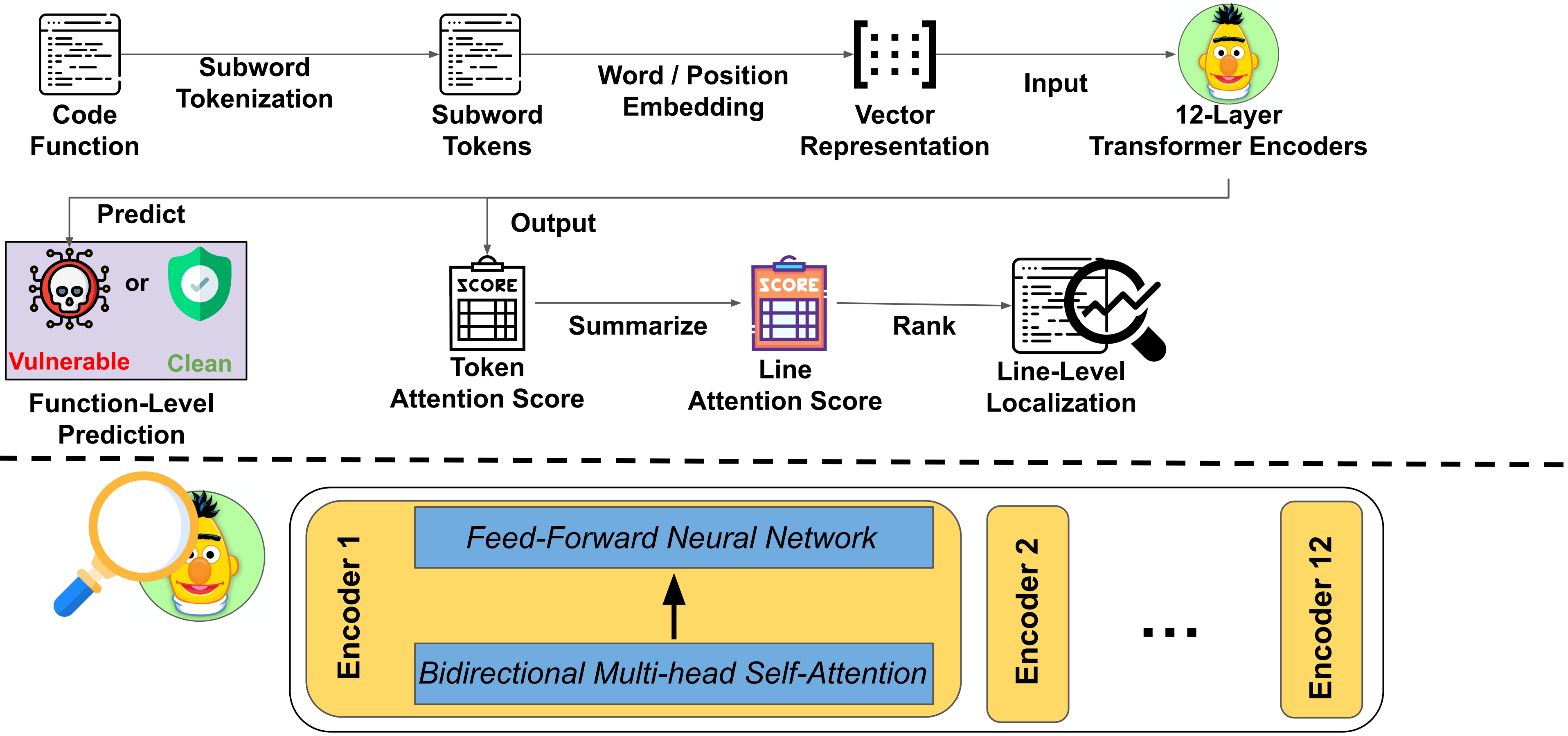
**Inaccurate Vulnerability Predictions**

The RNN-based feature extractor of IVDetect is not effective to capture the long-term dependencies of a sequence.

The subgraph-level vulnerability localization of IVDetect is still coarse-grained

**More Effort for the Security Analysts**

# LineVul: A Transformer-based Line-Level Vulnerability Prediction



# Example Line-Level Prediction of LineVul

## An example prediction of LineVul using real world data

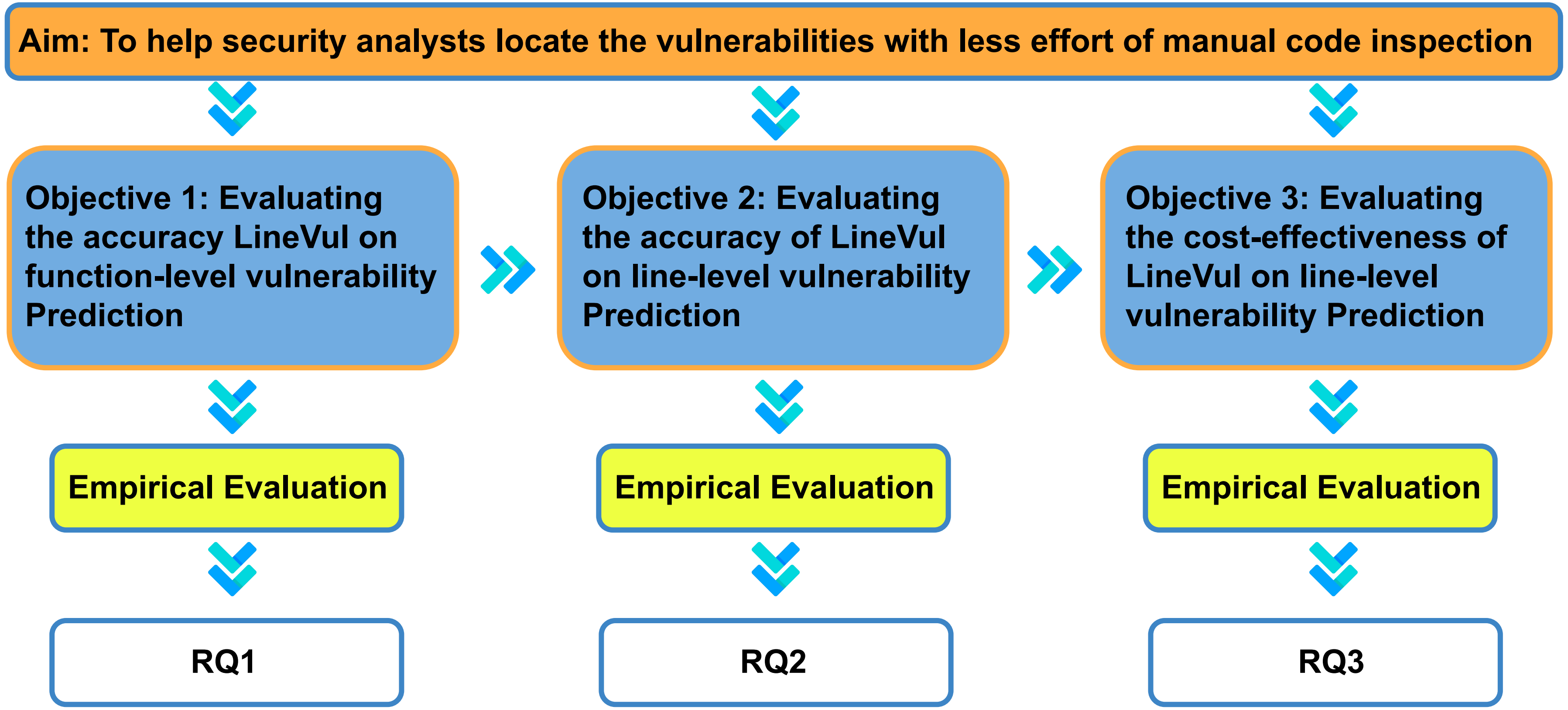
// Line-level Vulnerability Predictions by LineVul

| | third_party/WebKit/Source/core/frame/ImageBitmap.cpp https://github.com/chromium/chromium/commit/d59a4441697f6253e7dc3f7ae5caad6e5fd2c778 | LineVul | IVDetect | Ground truth |
|---|---|---|---|---|
| 224 | `static sk_sp<SkImage> unPremulSkImageToPremul (SkImage* input) {` | 0.8 | 0 | 0 |
| 225 | `SkImageInfo info = SkImageInfo::Make(input->width(), input->height(),` | 0.6 | 0 | 0 |
| 226 | `kN32_SkColorType, kPremul_SkAlphaType);` | 0.5 | 0 | 0 |
| 227 | `RefPtr<Uint8Array> dstPixels = copySkImageData(input, info);` | 0.8 | 0 | 0 |
| 228 | `if (!dstPixels)` | 0.3 | 1 | 0 |
| 229 | `return nullptr;` | 0.3 | 0 | 0 |
| 230 | `return newSkImageFromRaster(` | 0.5 | 1 | 0 |
| 231 | `info, std::move(dstPixels),` | 0.6 | 1 | 0 |
| 232 | `static_cast<size_t>(input->width()) * info.bytesPerPixel());` | 1 | 1 | 1 |
| 233 | `}` | 0 | 0 | 0 |

***Vulnerable Line***

**Security Analysts are able to locate the exact vulnerable line in a vulnerable function**

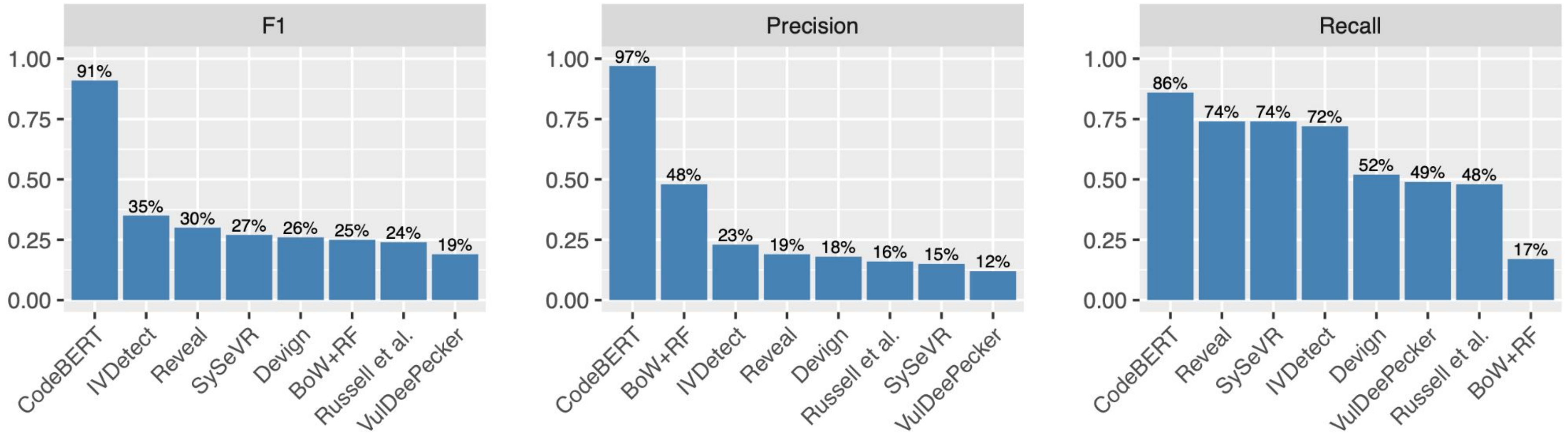
## Study Design


Aim: To help security analysts locate the vulnerabilities with less effort of manual code inspection
Objective 1: Evaluating the accuracy LineVul on function-level vulnerability Prediction
Objective 2: Evaluating the accuracy of LineVul on line-level vulnerability Prediction
Objective 3: Evaluating the cost-effectiveness of LineVul on line-level vulnerability Prediction
Empirical Evaluation
Empirical Evaluation
Empirical Evaluation
RQ1
RQ2
RQ3

RQ1

Function-level

## How accurate is our LineVul for function-level vulnerability predictions?



**Our LineVul achieves the highest F1-measure, Precision, and Recall when comparing with other baseline models including the SOTA IVDetect.**

RQ2

Line-level

## How accurate is our LineVul for line-level vulnerability localization?

***Metric 1 - Top-K Accuracy***

Evaluate the model performance by considering top-k lines in each function after the ranking procedure.

**Measure the line-level Precision of the model interpretation approach**

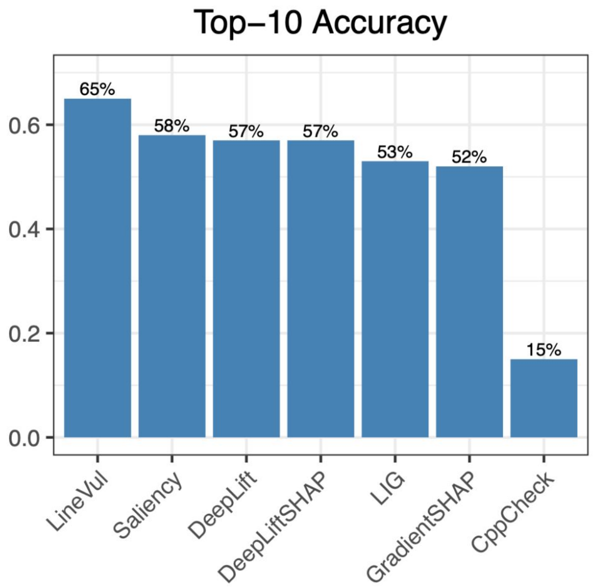
***Metric 2 - Initial False Alarm (IFA)***

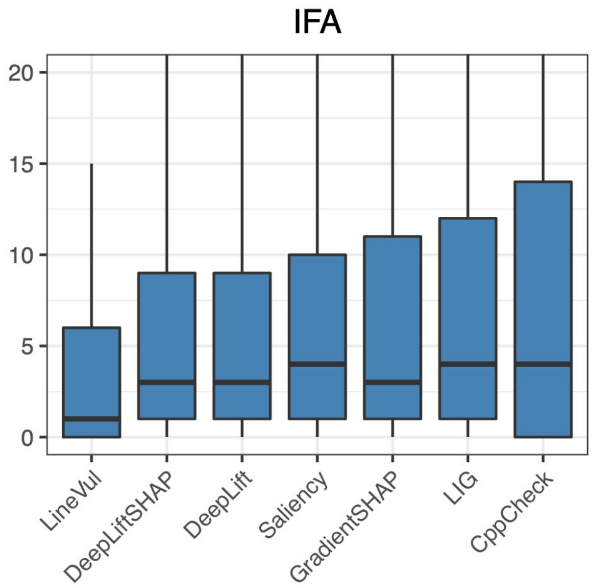
Evaluate how many lines do the model need to include when capturing the first vulnerable line in a function.

RQ2

Line-level

## How accurate is our LineVul for line-level vulnerability localization?





**The Attention Score Reasoning of LineVul achieves the best Top-10 Accuracy and IFA when comparing with other model interpretation approaches.**

RQ3

Line-level

## What is the cost-effectiveness of our LineVul for line-level vulnerability localization?

***Metric 1 - Effort@K%Recall***

**The number of inspected LOC when capturing the K% of vulnerable lines divided by the total LOC.**

**Measure the line-level cost-effectiveness of the model interpretation approach**

***Metric 2 - Recall@K%LOC***

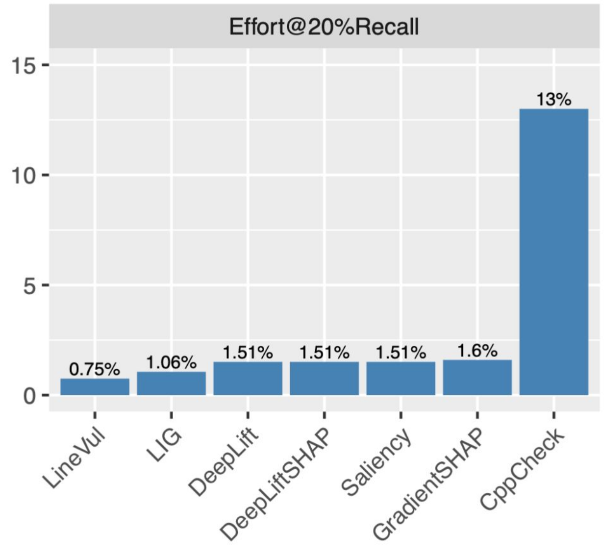
**Given a fixed amount of effort (K% LOC), how many vulnerable lines can be captured by the model.**

RQ3

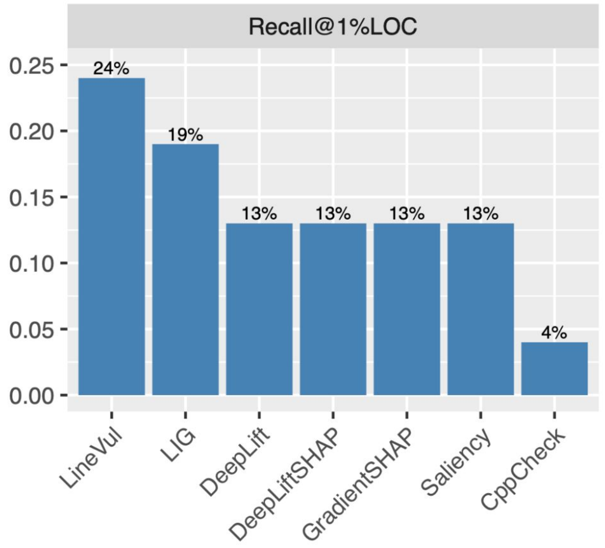
Line-level

## What is the cost-effectiveness of our LineVul for line-level vulnerability localization?

*Fixed amount of vulnerabilities*



*Fixed amount of effort*



**The Attention Score Reasoning of LineVul achieves the best Effort@20%Recall and Recall@1%Loc when comparing with other model interpretation approaches.**

## Take-Away Messages

1. Vulnerability Prediction Model is needed to mitigate the challenge of vulnerability detection and help security analysts locate the vulnerable code faster.

2. LineVul is one of the important advancement toward more accurate and finer-grained Vulnerability Prediction.

## Thank you very much for your listening

Full Paper: shorturl.at/hjkrK

To replicate our LineVul approach:

please go to https://github.com/awsm-research/LineVul

For any issues or collaboration,

please email: yeh.fu@monash.edu