# Automatic Name-based Software Bug Detection during Static Program Analysis

**Team Members:**
**Shashank Ghimire (THA076BEI032)**
**Shirshak Acharya (THA076BEI033)**
**Yunij Karki (THA076BEI044)**
**Dipu Dahal (THA076BEI045)**

Under the Supervision of
Er. Dinesh Baniya Kshatri

Department of Electronics and Computer Engineering
Institute of Engineering, Thapathali Campus

March 9, 2023

# Presentation Outline

- Motivation
- Introduction
- Objectives
- Scope of Project
- Project Applications
- Methodology
- Dataset Exploration
- Results
- Discussion of Results
- Future Enhancements
- Conclusion
- References

# Motivation

Programmer frustrated by manual analysis of bugs

```c
#include <stdio.h>

void setDimension(int x, int y)
{
 printf("Width: %d, Height: %d\n", x, y);
}

int main()
{
   int height = 7, width = 10;
   setDimension(height, width);
   return 0;
}
```

Code Snippet

```
setDimension(height, width)
```

Possible bugs

Tool to automatically detect some bugs in source code

# Introduction

- Name-based bugs refer to errors in the names of variables, function arguments, operators
- Static analysis is the detection of bugs in source code before compilation
- Learning-based approach has been adopted to scan source code to identify potential name-based bugs

# Objectives

- To create suitable training data consisting of correct and buggy code from a corpus of source code written in the C-programming language
- To train and evaluate a transformer model capable of name-related bug detection during static code analysis

# Scope of Project

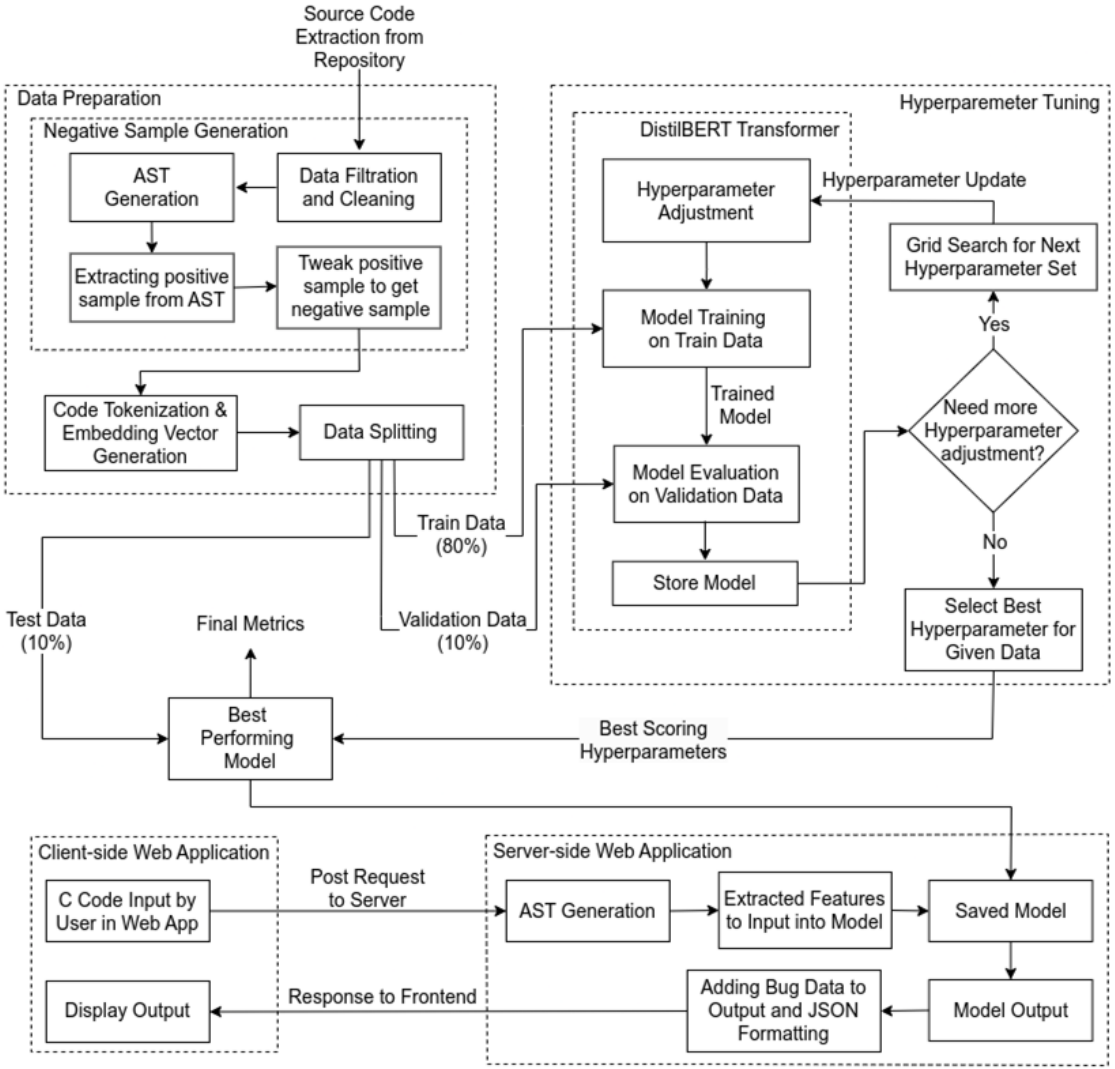
- Project Capabilities:
  - Automatically generates negative samples from correct code using AST
  - Localizes swapped function arguments & wrong binary operators

- Project Limitations:
  - Detects only two types of name-based bugs
  - Handles code written in C programming language
  - Behaves incorrectly if identifiers are named improperly

# Project Applications

- Bug Identification
  - Reduces the need for manual code review
- Improving Code Quality
  - Minimizes the number of errors and crashes
- Coding Competitions
  - Locates bugs in source code & acts as an automated judge
- Coding Interview
  - Identifies bugs in candidate's code quickly & accurately
- Boosting Coding Efficiency
  - Increases productivity and saves time for developers

# Methodology - [1] (System Block Diagram)



# Methodology - [2]
## (Working Principle)

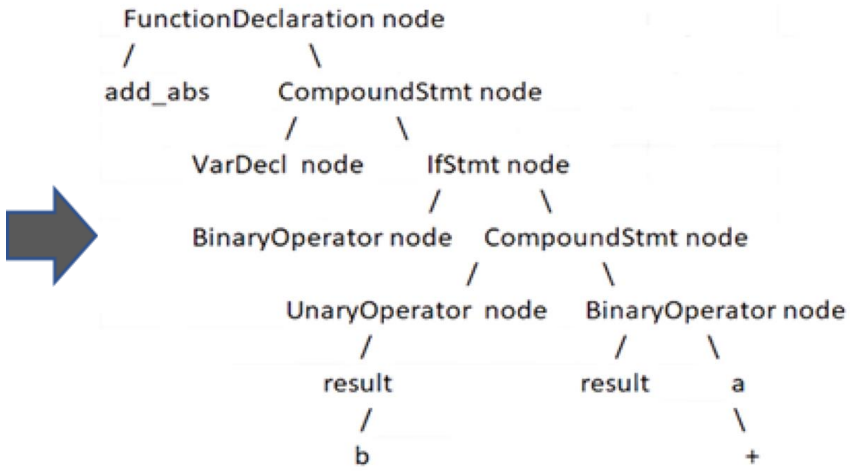
- Source code is extracted from C Code Corpus dataset
- Only the code snippets with less than 10k lines of code are taken and missing data are handled
- AST is generated and positive samples are extracted
- Dataset containing positive samples were split such that train (80%), validation (10%) & test (10%)
- Negative samples generated with the help of positive samples

# Methodology - [3]
## (Working Principle)

- CodeT5 tokenizer was fine-tuned on C code
- Code tokenization and Embedding vector generation was done
- Model was trained with DistilBERT & CodeT5 tokenizer
- Hyperparameters were tuned using Grid Search
- Transformer outputs result as 0 (correct) or 1 (incorrect)
- Client-side and server-side web app were implemented

# Methodology - [4]:
## (AST - General Structure of Function Declaration)

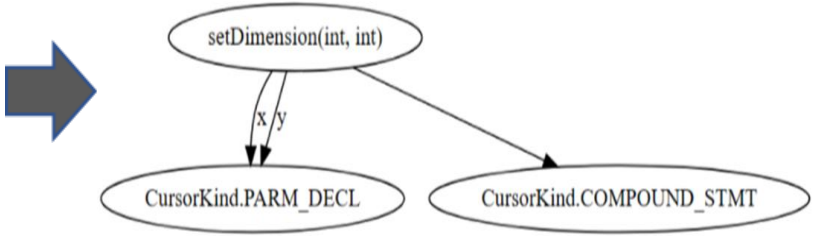
| Line no. | Code | AST Node |
|---|---|---|
| 1 | int add_abs (int a, int b){ | Function Declaration Node with the start of CompoundStmt Node |
| 2 | int result; | VarDecl Node |
| 3 | if (a + b < 0) { | Ifstmt Node and BinaryOperator Node with the start of another CompoundStmt Node |
| 4 | result = - (a + b); | UnaryOperator Node |
| 5 | } else { | End of CompoundStmt Node |
| 6 | result = a + b; | BinaryOperator Node |
| 7 | } | |
| 8 | return result;} | |



# Methodology - [5]:

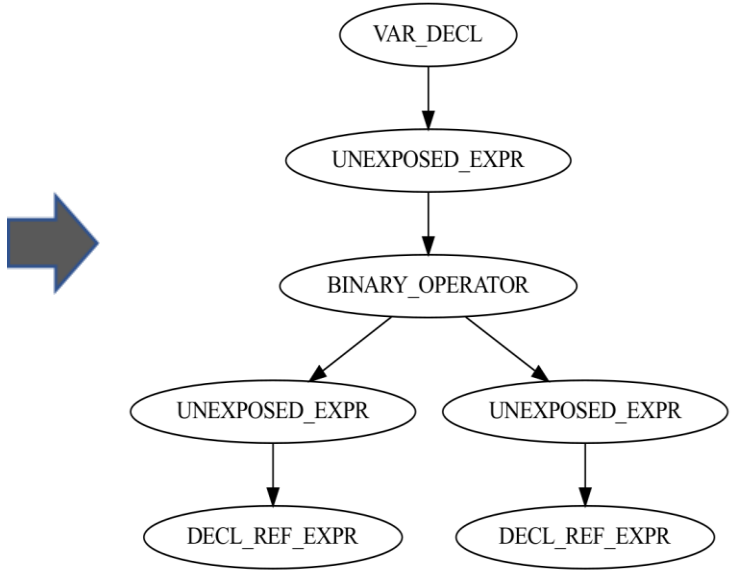
## (AST Generation - Function Arguments Swap)

| Line no. | Code | Comments |
|---|---|---|
| 1 | #include <stdio.h> | |
| 2 | void setDimension(int x, int y){ | Function declaration with parameters ‘x’ and ‘y’ |
| 3 | printf("Width: %d, Height:  %d\n", x, y); | Function body |
| 4 | } | |
| 5 | int main() | Main function |
| 6 | { | |
| 7 | int height = 7, width = 10; | Variable declaration |
| 8 | setDimension(width, height); | Function call with possible bug |
| 9 | return 0; | |
| 10 | } | |



# Methodology - [6]:
## (AST Generation - Wrong Binary Operator)

| Line no. | Code | Comments |
|---|---|---|
| 1 | #include <stdio.h> | |
| 2 | int main() | main function |
| 3 | { | |
| 4 | int a = 5, b = 7; | variable declaration i.e. ‘a’ and ‘b’ |
| 5 | float d = a * b; | binary operation with binary operator ‘*’ |
| 6 | printf("Result is: %d", d); | |
| 7 | return 0; | |
| 8 | } | |



# Methodology - [7]: (Possible Operator Exchanges)

| Operator | Possible Swaps |
|---|---|
| '==' | ['>', '<', '!=', '>=', '<=', '='] |
| '<' | ['>', '==', '!=', '>=', '<=', '<<', '^', '>>'] |
| '+' | ['-', '*', '/', '%', '+='] |
| '*' | ['+', '-', '/', '%', '*='] |
| '-' | ['+', '*', '/', '%', '-='] |
| '!= | ['>', '<', '==', '>=', '<=', '='] |
| '>' | ['==', '<', '!=', '>=', '<=', '<<', '^', '>>'] |
| '&': | ['\|', '^', '<<', '>>', '&&', '\|\|'] |
| '>=' | ['>', '<', '!=', '==', '<=', '<<', '^', '>>'] |
| '/' | ['+', '-', '*', '%', '/='] |
| '<=' | ['>', '<', '!=', '>=', '==', '<<', '^', '>>'] |
| '&&' | ['\|\|', '&', '\|'] |
| '<<' | ['\|', '&', '^', '>>', '&&', '\|\|', '<=', '<'] |
| '>>' | ['\|', '&', '^', '<<', '&&', '\|\|', '>=', '>'] |
| '\|\|' | ['&&', '\|', '&', '/'] |
| '%' | ['+', '-', '*', '/', '<', '>', '>=', '<=', '!=', '%='] |
| '\|' | ['&', '^', '<<', '>>', '&&', '\|\|', '/'] |
| '^' | ['\|', '&', '<<', '>>', '&&', '\|\|', '<', '>'] |

# Methodology - [8]:
## (Negative Sample Generation - Swapped Function Args)

- Positive sample extracted from AST for Swapped function args bug
- Label positive sample as 0
- Swap the operator and label sample as 1 (negative)

| function_name | arg1 | arg2 | arg_type | param1 | param2 | labels |
|---|---|---|---|---|---|---|
| setDimension | width | height | int | x | y | 0 |
| setDimension | height | width | int | x | y | 1 |

# Methodology - [9]:
## (Negative Sample Generation - Wrong Binary Operator)

- Positive sample extracted from AST for Wrong binary operator bug
- Label positive sample as 0
- Swap the operator and label sample as 1 (negative)

| left | operator | right | type_left | type_right | parent | grandparent | labels |
|---|---|---|---|---|---|---|---|
| i | < | linkCount1 | int | int | FOR_STMT | COMPOUND_STMT | 0 |
| i | <= | linkCount1 | int | int | FOR_STMT | COMPOUND_STMT | 1 |

# Methodology - [10]:
## (Input Embedding of Tokens)

- Process of mapping tokens to the vectors of real numbers
- Words with same meaning have a similar d-dimensional vector values
- Used to convert input tokens into vectors
- Can help to predict output for unseen data

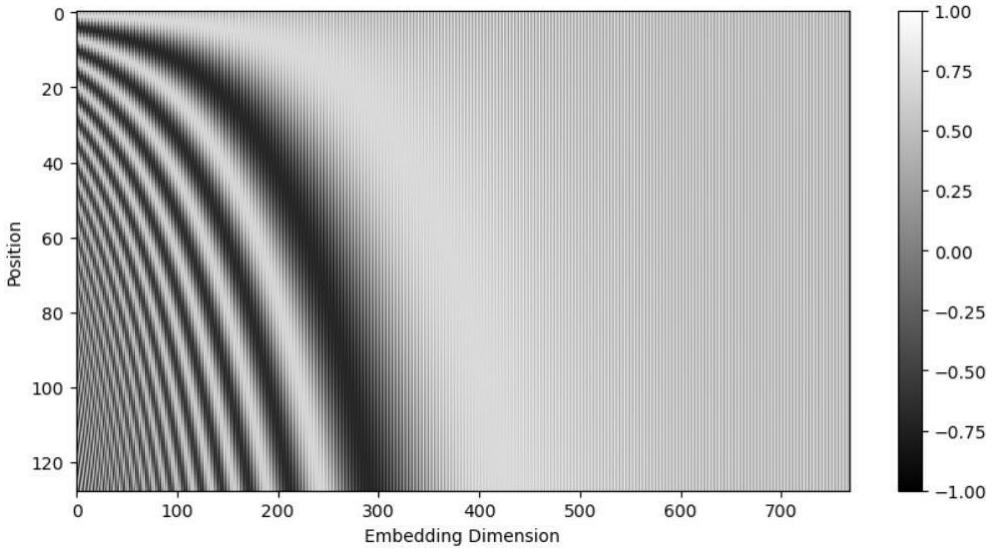
# Methodology - [11]:
## (Positional Encoding of Tokens)

- Preserves information about position of tokens in sequence

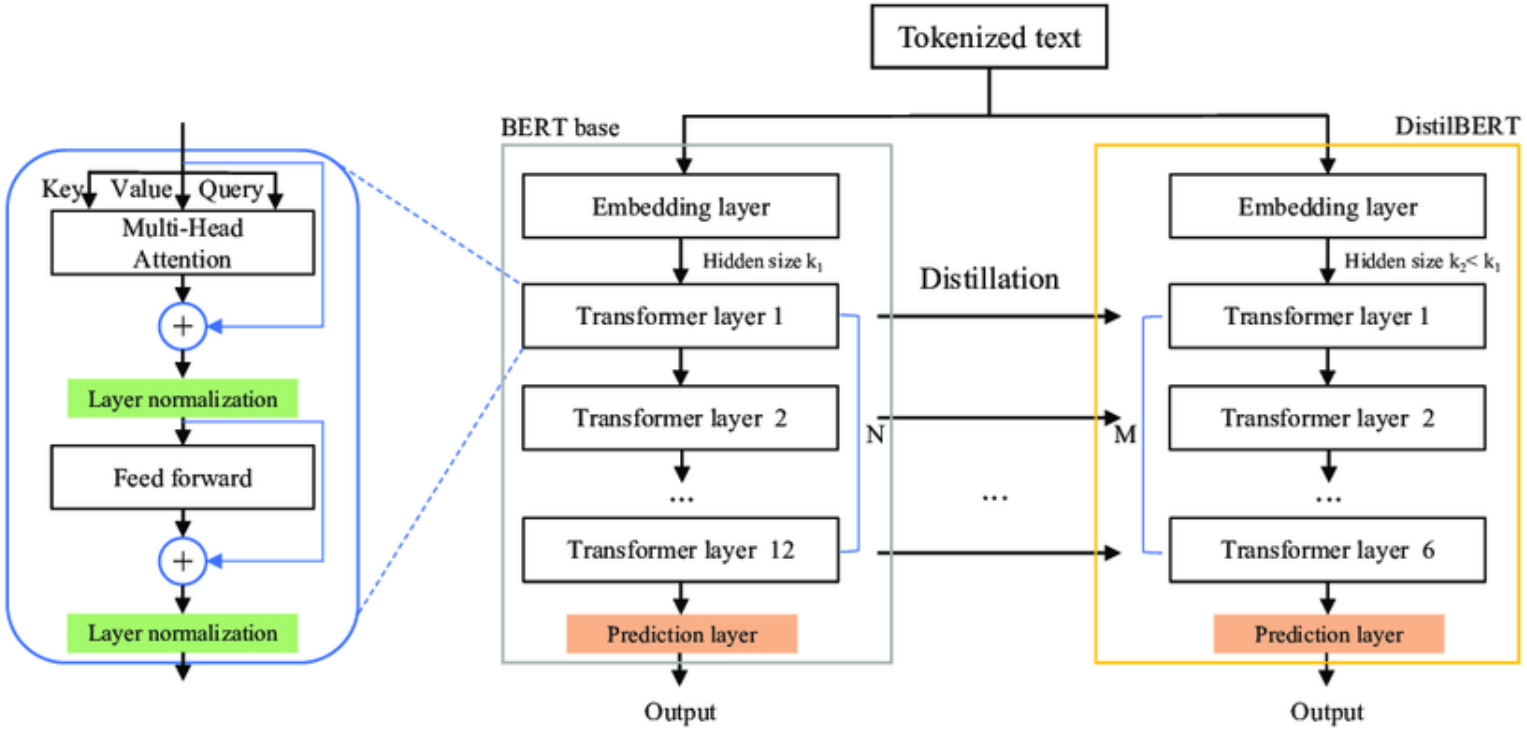
  $PE_{(pos,\ 2i)} = \sin(pos/10000^{2i/d})$

  $PE_{(pos,\ 2i+1)} = \cos(pos/10000^{2i/d})$

- Where ‘pos’ is token position, ‘d’ is length of positional encoding vector and ‘i’ ranges from 0 to d/2



# Methodology - [12]:
## (DistilBERT transformer model)



# Methodology - [13]:
## (Model Evaluation)

- Binary Cross Entropy Loss Function
  - Loss function for binary classification problem
  - $BCE = -(y \times \log(y') + (1 - y) \times \log(1 - y'))$
- F1 Score
  - considers both precision and recall
  - $$\text{F1 Score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

# Methodology - [14]:
## (Hardware and Software Requirements)

- Kaggle
  - Hosted Jupyter notebook service for model training
- GPU
  - Processor to accelerate training through parallel processing
  - GPU with 15.9 GB RAM used
  - Available GPUs: Nvidia T4 x 2 and P100

# Methodology - [15]:
# (Hardware and Software Requirements)

- Pytorch
  - Python ML framework used for model training
- Fast API
  - Python based web framework
  - Used at server-side for creating API
- React
  - Javascript based frontend web framework

# Dataset Exploration - [1]

- C code corpus dataset
  - Raw corpus data of C source files
  - 720,000 code files of C programming
  - 2.5 GB in compressed form and 12 GB when uncompressed
  - 486,534 total samples in swapped function arguments dataset
  - 1,111,986 total samples in wrong binary operator dataset

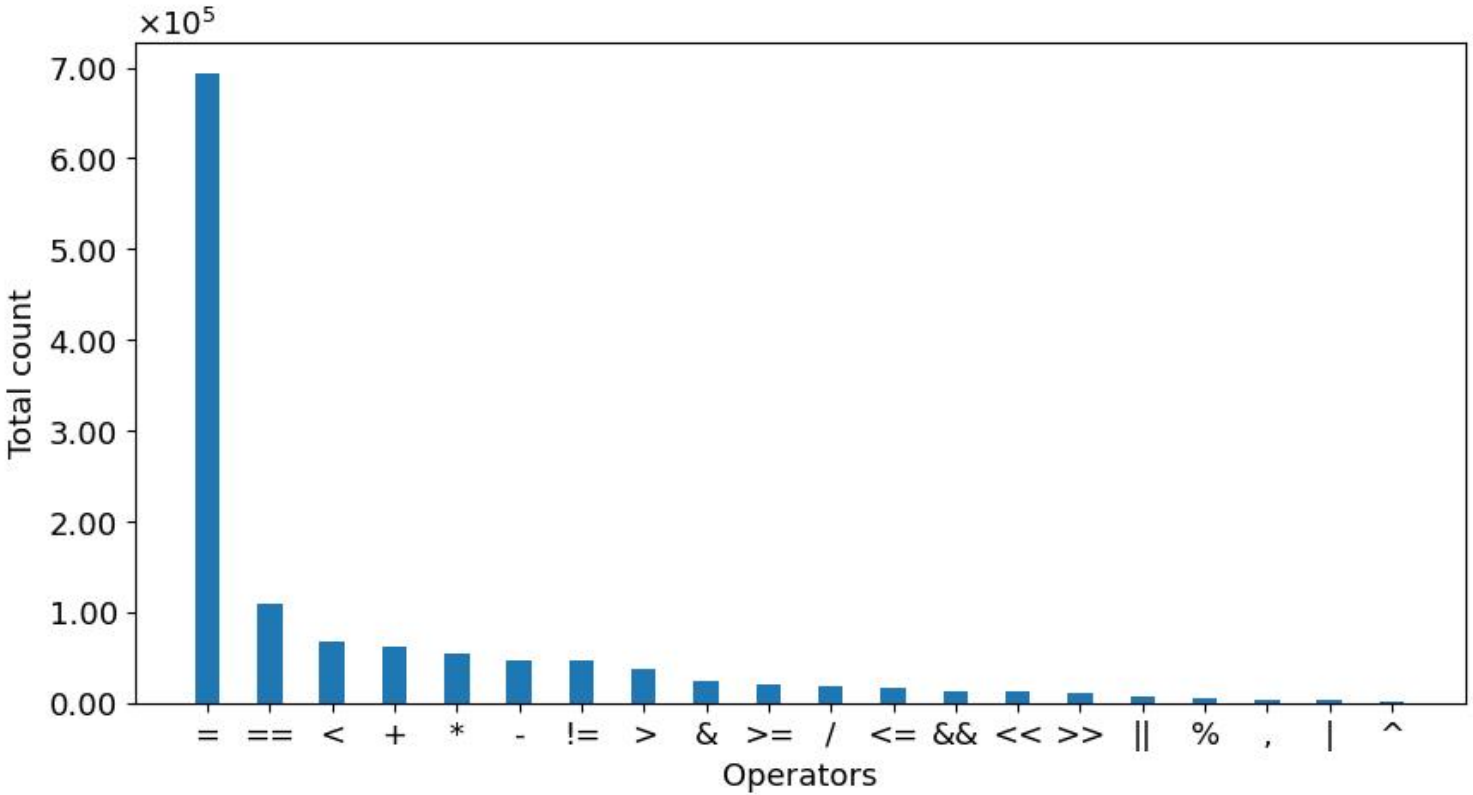
# Dataset Exploration - [2]
## (Swapped Function Arguments - Train Dataset Preview)

| function_name | arg1 | arg2 | arg_type | param1 | param2 | labels |
|---|---|---|---|---|---|---|
| printf | location | (%s) Bad data in stream\n | const char * | NaN | NaN | 1 |
| copysignf | minus_infty | minus_zero | float | NaN | NaN | 0 |
| pjsua_conf_connect | source | sink | int | NaN | NaN | 0 |
| strcmp | pi->msg2 | QUEUE | const char * | NaN | NaN | 0 |
| RADEON_ALIGN | *w | 4 | int | NaN | NaN | 0 |
| strstr | p | 100Mb | const char * | __haystack | __needle | 0 |
| strcmp | keyword | prefix | const char * | __s1 | __s2 | 1 |
| ovcamWriteRegister | 0x22 | 0x7B | unsigned char | subaddr | data | 1 |
| _cairo_array_index | font->scaled_font_subset->glyphs[i] | &font->charstrings_index | <dependent type> | NaN | NaN | 1 |
| DBG | Serial port %s released | port->dev | char * | NaN | NaN | 0 |

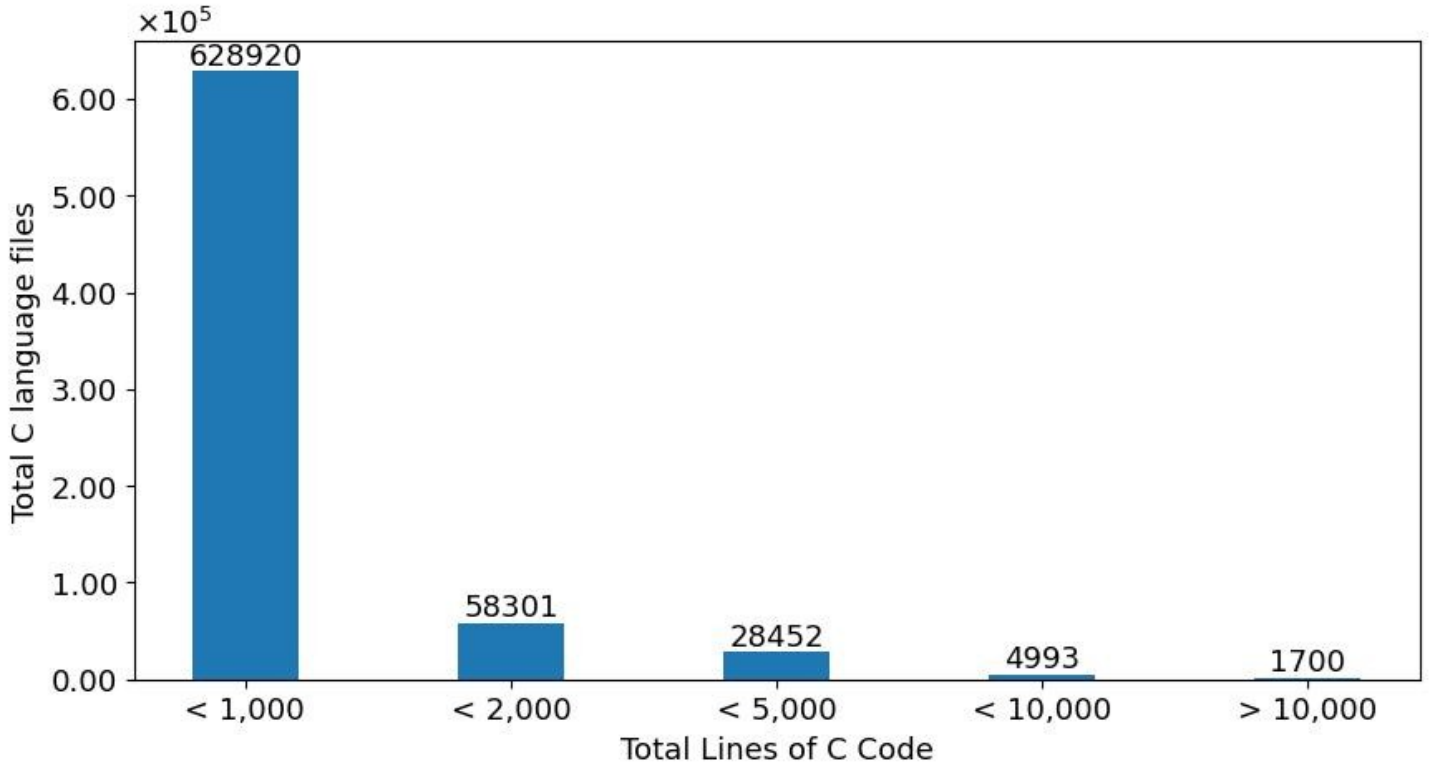
# Dataset Exploration - [3]
## (Wrong Binary Operators - Train Dataset Preview)

| left | operator | right | type_left | type_right | parent | grandparent | labels |
|---|---|---|---|---|---|---|---|
| cy2 | >= | bb_y2 | int | int | BINARY_OPERATOR | IF_STMT | 0 |
| len | >> | 4 | int | int | CONDITIONAL_OPERATOR | VAR_DECL | 1 |
| 1 | << | dio->blkfactor | int | unsigned int | UNEXPOSED_EXPR | BINARY_OPERATOR | 0 |
| MAX_N_PASSES | / | MAX_N_SAMPLES | unsigned long | unsigned long | PAREN_EXPR | BINARY_OPERATOR | 1 |
| LocaleNCompare("yresolution",property,11) | <= | 0 | int | int | IF_STMT | COMPOUND_STMT | 1 |
| decpt | /= | 100 | int | int | BINARY_OPERATOR | IF_STMT | 1 |
| dbf->type[k] | == | 'C' | int | int | IF_STMT | COMPOUND_STMT | 0 |
| buffer_position | != | 5 | int | int | BINARY_OPERATOR | IF_STMT | 1 |
| i | < | linkCount1 | int | int | FOR_STMT | COMPOUND_STMT | 0 |
| sum0 | + | sum1 | int | int | BINARY_OPERATOR | BINARY_OPERATOR | 0 |

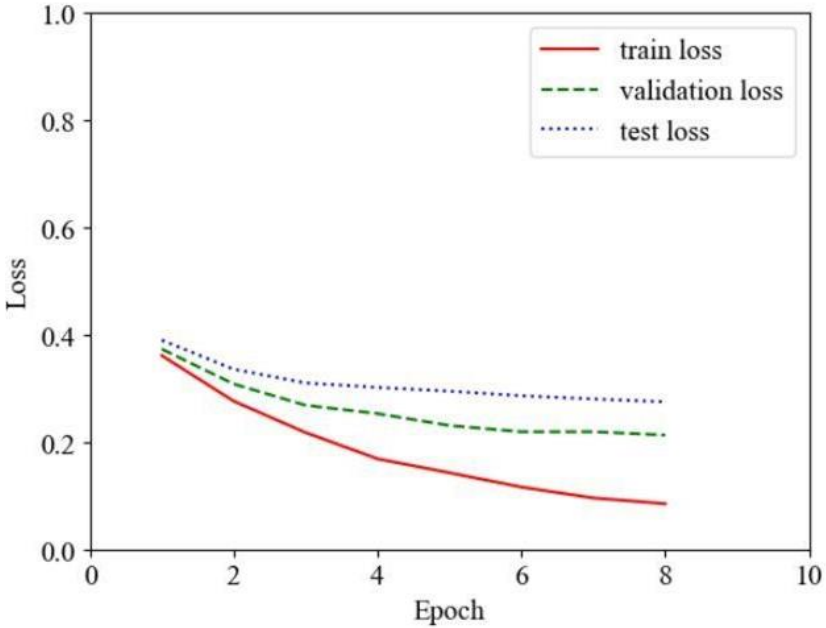
# Dataset Exploration - [4]
## (Bar Plot of Operator Count)

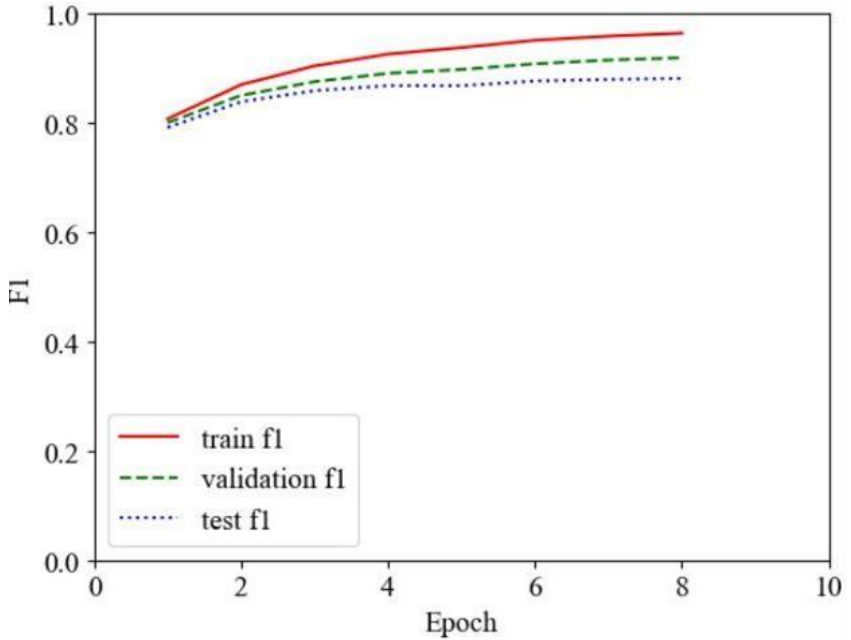


# Dataset Exploration - [5]
## (Count of Total Lines of Code)

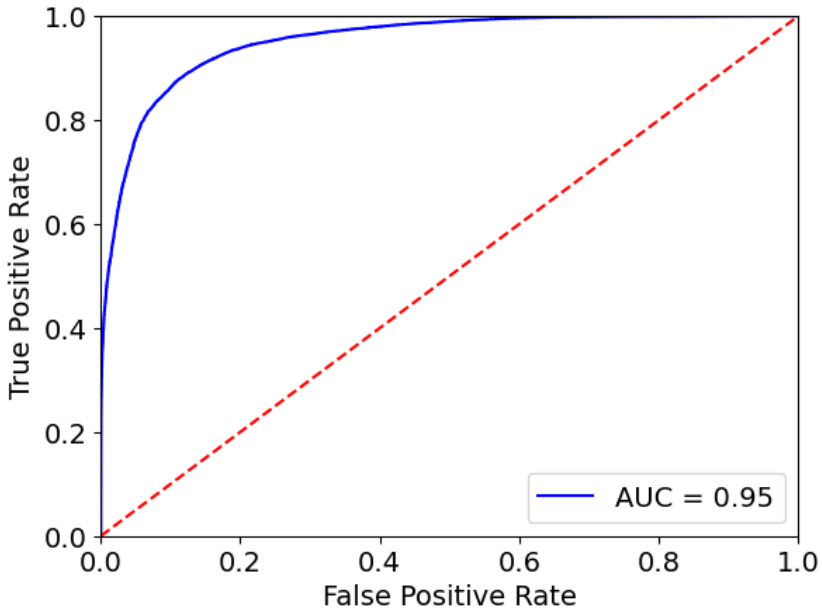


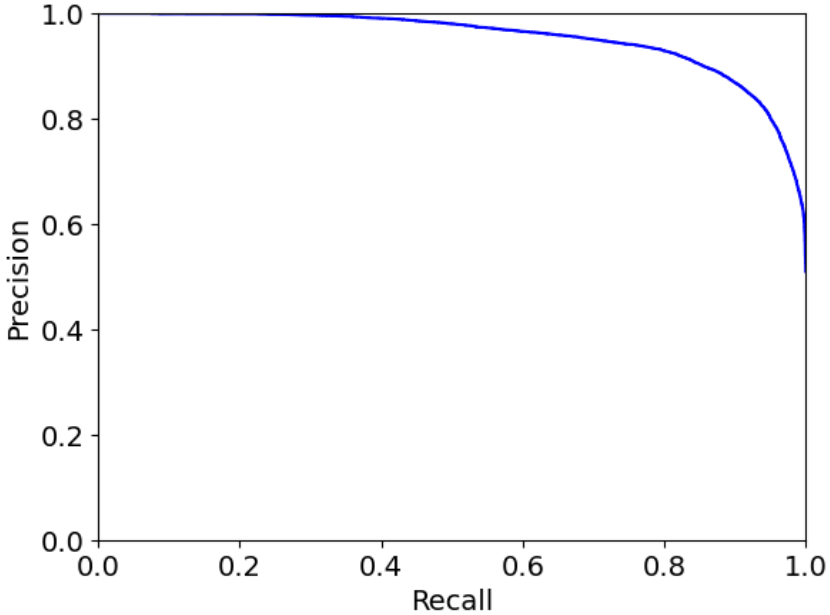
# Results - [1]
## (Swapped Function Arguments - Loss & F1 Score)



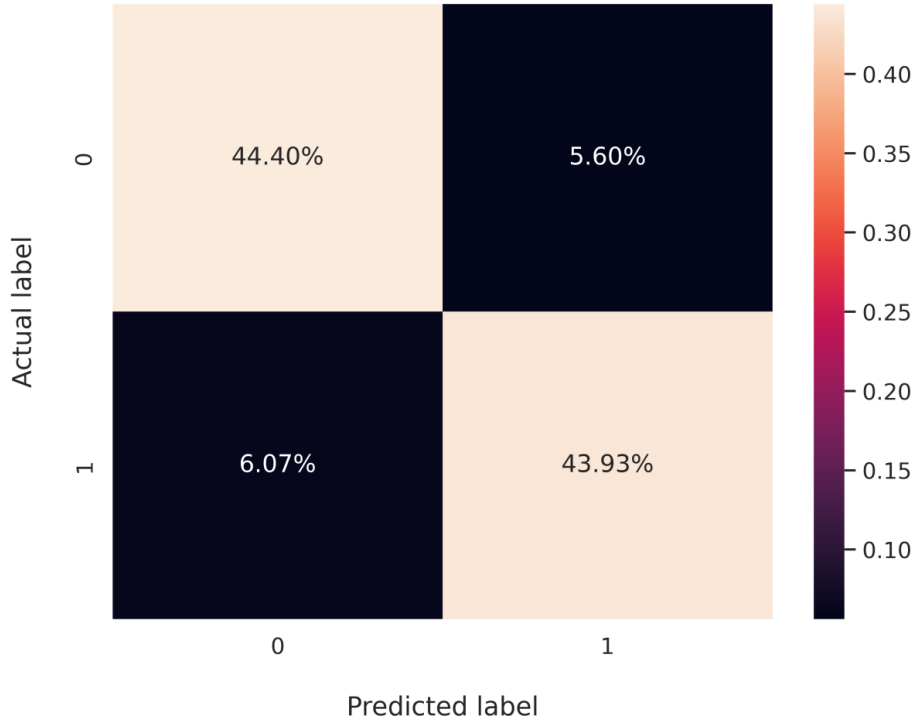



# Results - [2]
## (Swapped Function Arguments - ROC & PR Curve)



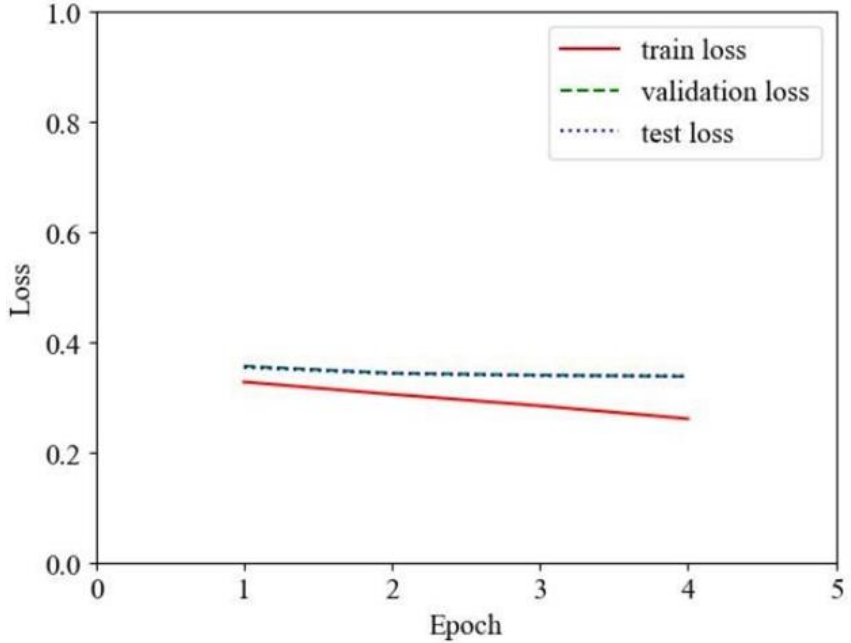



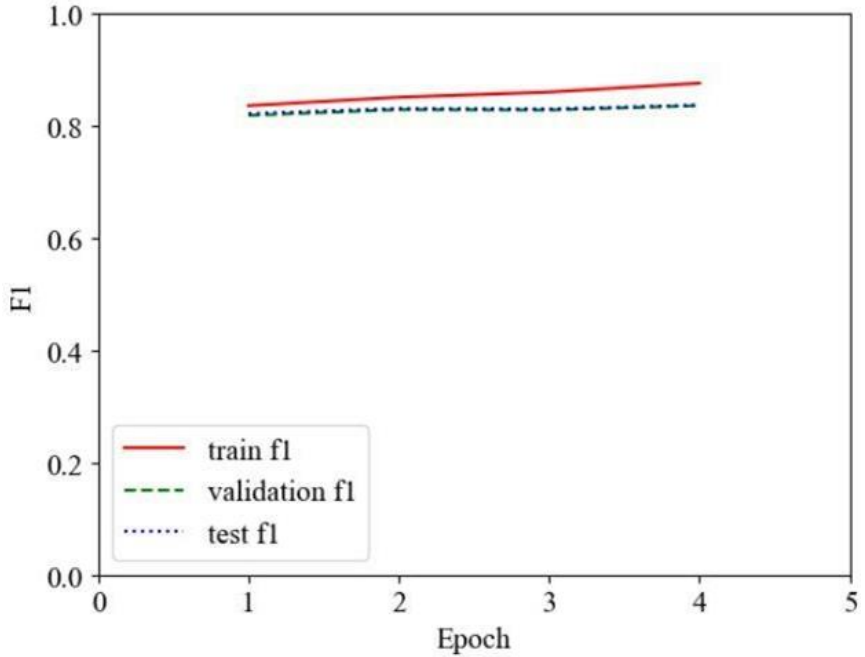
# Results - [3]
## (Swapped Function Arguments - Confusion Matrix)



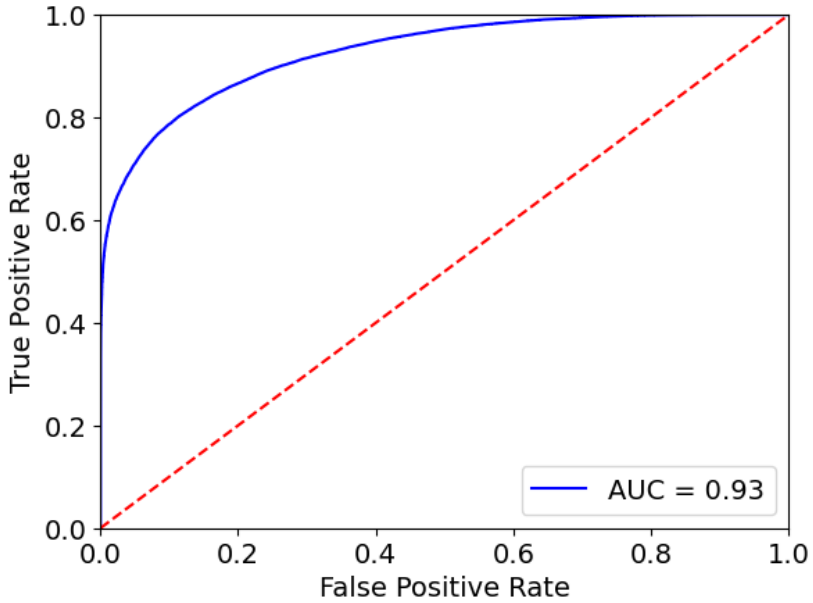

- True Positive = 43.93%
- True Negative = 44.40%
- False Positive = 5.60%
- False Negative = 6.07%
- Generally, in ML, 1 is taken as positive class so buggy prediction is positive here

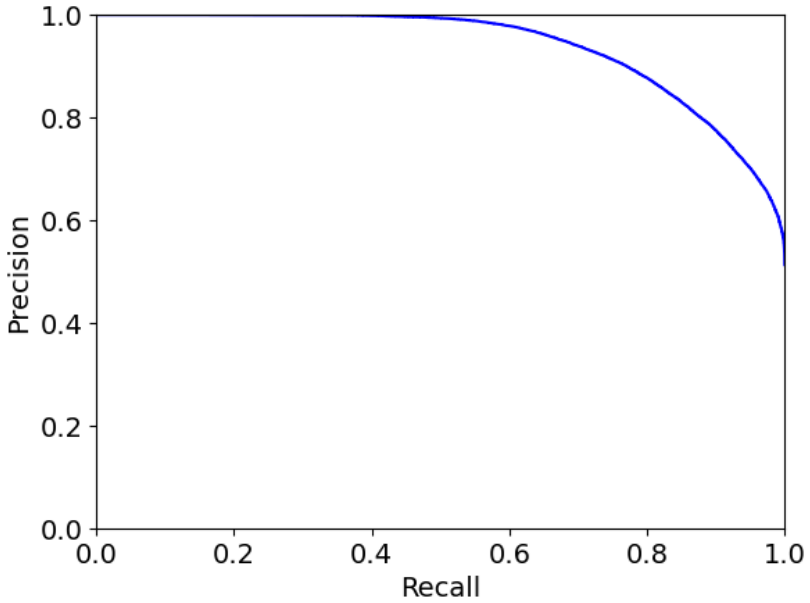
# Results - [4]
## (Wrong Binary Operator - Loss & F1 Score)



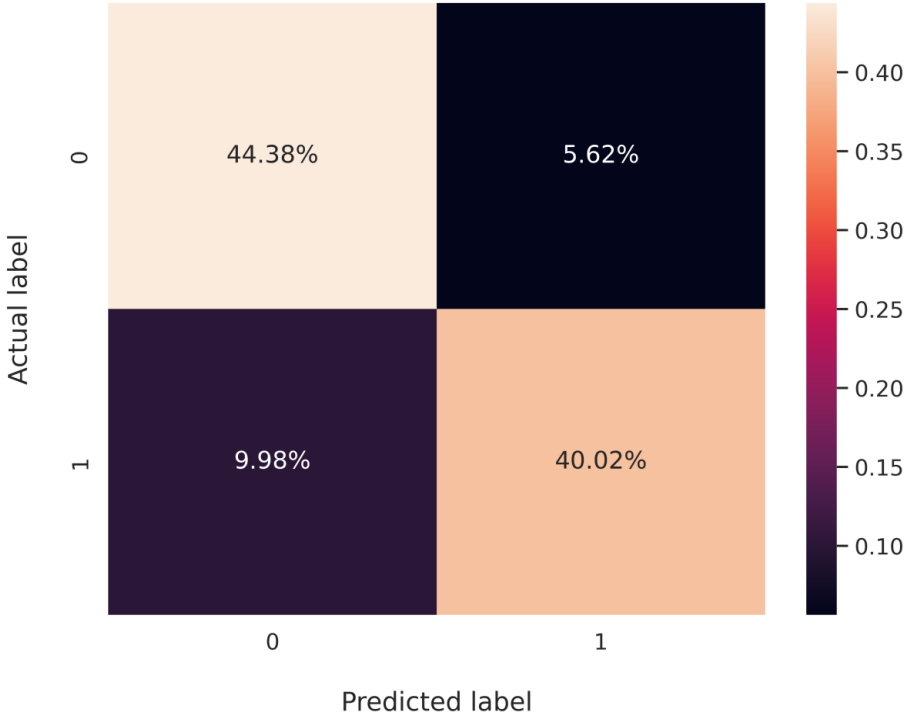



# Results - [5]
## (Wrong Binary Operator - ROC & PR Curve)



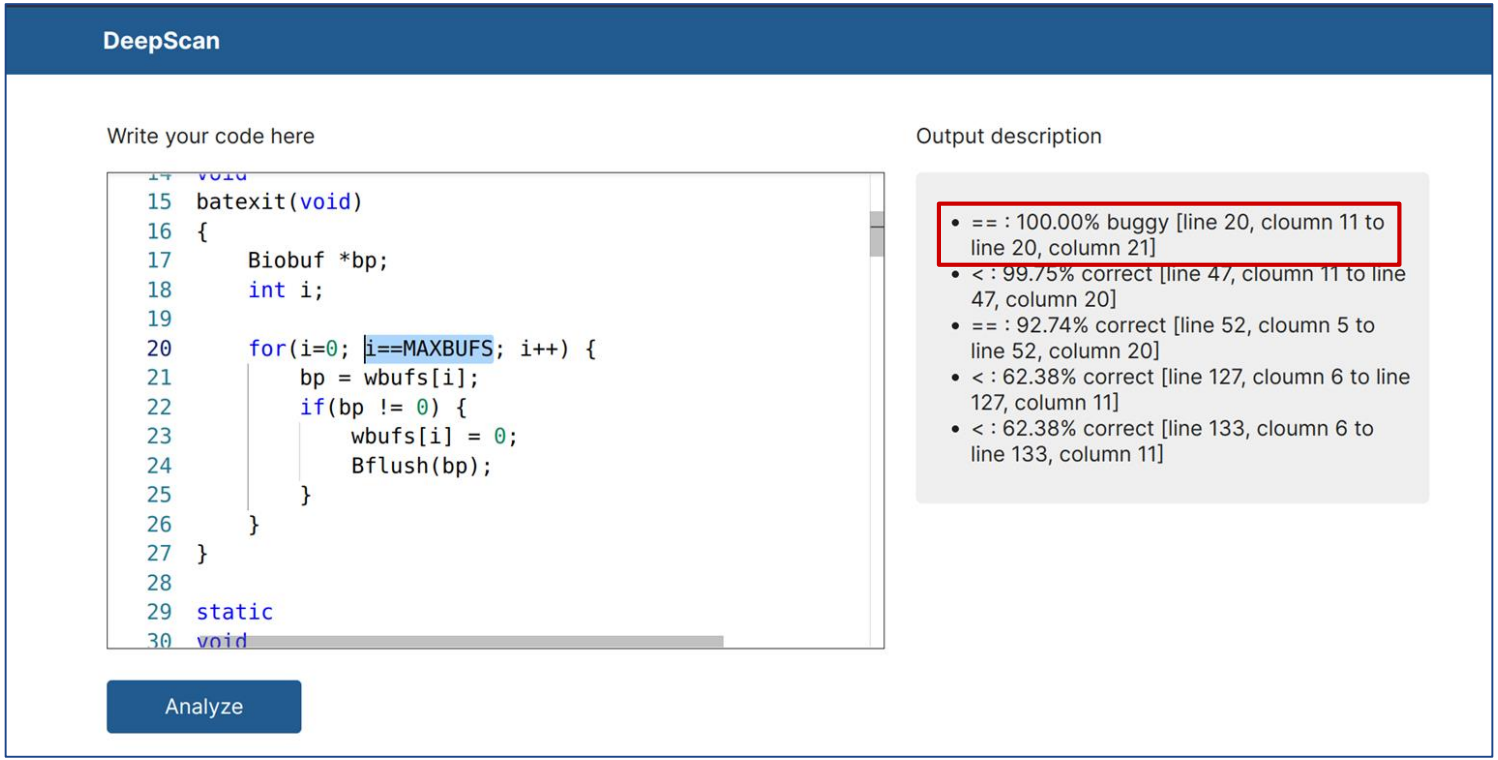



# Results - [7]
## (Wrong Binary Operator - Confusion Matrix)

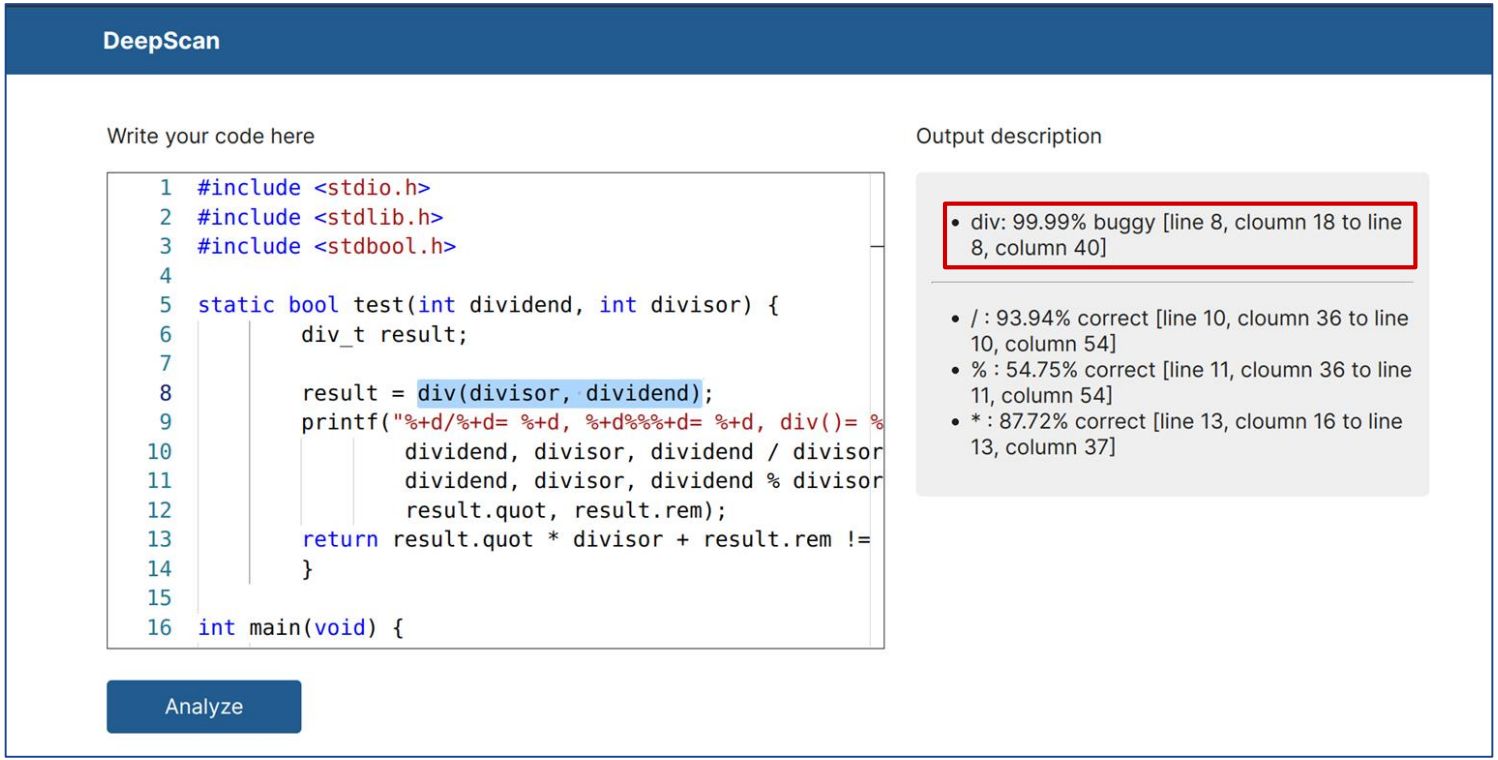



- True Positive = 40.02%
- True Negative = 44.38%
- False Positive = 5.62%
- False Negative = 9.98%
- Generally, in ML, 1 is taken as positive class so buggy prediction is positive here

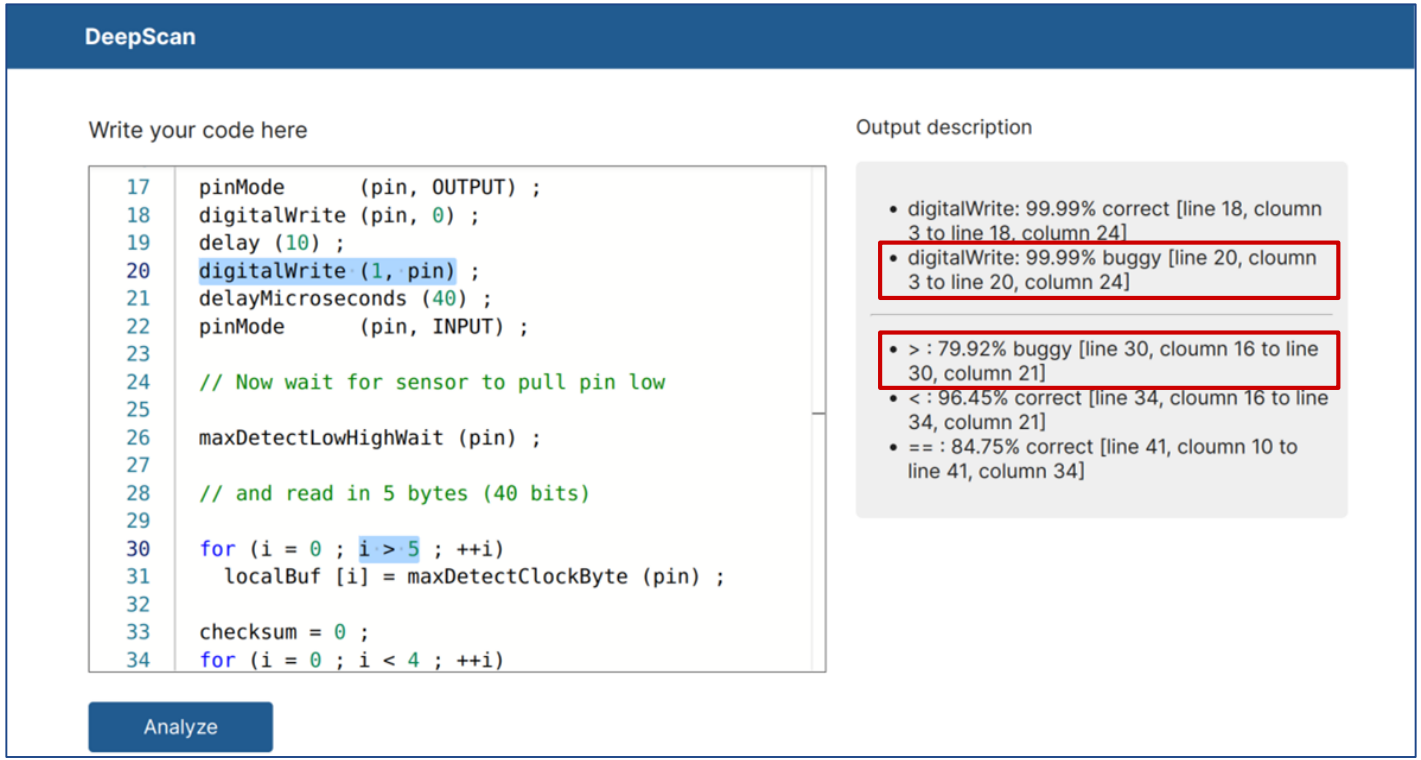
# Results - [6]
## (User Interface - Wrong Binary Operator Example)



# Results - [8]
## (User Interface - Swapped Function Arguments Example)



# Results - [9]
## (User Interface - Multiple Types of Bugs Example)



# Results - [10]
## (Comparison of Performance with Similar Projects)

| Type of Bug | DeepBugs | | This Project |
|---|---|---|---|
| | Random Embedding | Learned Embedding | Learned Embedding |
| Function Arguments Swap | 93.88% | 94.70% | 91.89% |
| Wrong Binary Operator | 89.15% | 92.21% | 84.39% |

- Comparable but slightly less accuracy can be due to:
  - Hyperparameter tuning with low number of hyperparameters
  - Use of different language (i.e. Javascript in DeepBugs)

# Discussion of Results

- Swapped function arguments model
  - Validation dataset: Accuracy = 91.89%, F1 score = 91.89%
  - Test dataset: Accuracy = 88.10%, F1 score = 88.12%
  - Training time: 45 minutes for 1 epoch (total 8 epochs)
- Wrong binary operator model
  - Validation dataset: Accuracy = 84.39%, F1 score = 83.60%
  - Test dataset: Accuracy = 84.40%, F1 score = 83.69%
  - Training time: 1 hour and 40 minutes for 1 epoch (total 4 epochs)
- The percentage of correctness or bugginess in the output is due to the softmax function

# Future Enhancements

- Exploring other transformer-based models
  - BERT, RoBERTa, ALBERT
- Introduction of other types of name based bugs
  - Wrong operator precedence, wrong binary operands
- Detection of bugs in multiple programming languages
  - Include name-based bug detection in languages other than C
  - Same concepts and steps used in this project can be applied

# Conclusion

- Suitable dataset consisting of correct and buggy samples of C language was created
- CodeT5 tokenizer and DistilBERT model were used
- Implemented two types of name-based bug detection models: swapped function arguments & wrong binary operator

# References - [1]


- Allamanis, Miltiadis *et al.,* “Self-Supervised Bug Detection and Repair.”, 2021. *Arxiv*, https://arxiv.org/abs/2105.12787. Accessed 3 Jan, 2023

- Saikat Chakraborty *et al.,* “Deep Learning-based Vulnerability Detection: Are We There Yet?”, 2021. *Arxiv*, *https://arxiv.org/abs/2009.07235*. Accessed 5 Jan, 2023

- Pewny, Jannik, and Thorsten Holz. “EvilCoder: Automated Bug Insertion.”, 2020. *Arxiv*, *https://arxiv.org/abs/2007.02326*. Accessed 29 December, 2022

- Karampatsis, Rafael-Michael *et al.*, “Big Code != Big Vocabulary: Open-Vocabulary Models for Source Code.”, 2020. *https://arxiv.org/abs/2003.07914*. Accessed 25 December, 2022

# References - [2]


- Pradel, Michael, and Koushik Sen, “DeepBugs: A Learning Approach to Name-based Bug Detection.”, 2018, *https://arxiv.org/abs/1805.11683*. Accessed 30 December, 2022

- A. Vaswani et al., “Attention Is All You Need”, 2017. *Arxiv*, *https://arxiv.org/abs/1706.03762*. Accessed 22 December, 2022

- "Kaggle: Your Machine Learning and Data Science Community", *https://www.kaggle.com*. Accessed 21 December, 2023

- C. Raffel *et al.*, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”, 2019. Arxiv, *https://arxiv.org/abs/1910.10683*. Accessed 3 January, 2023