# Source Code Vectorization using Graph Neural Network from Control Flow Graph

**Presented by:**
Riyad Morshed Shoeb
Roll: 1603013
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

**Supervised by:**
Sadia Zaman Mishu
Assistant Professor
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

June 19, 2022

# Contents



# Introduction

- Programs have special characteristics, (*e.g.* context-free syntactic structure, type constraints, the program's semantics, etc), unlike plain text.
- As a result, source codes do not have the form suitable to most learning techniques.
- So, it is necessary to transform the program to a suitable representation before a learning technique can be applied.

# Objective

- Represent code-fragments well to effectively capture syntactic and semantic information in source code for follow-up analysis.

# Motivation

- Source code classification
- Code clone detection
- Defect prediction
- Code summarizing/review
- Bug localization
- Code authorship classification

# Challenge

There are two challenges:

1. Representing a snippet in a way that enables learning across programs.
2. Learning which parts in the representation are relevant to prediction of the desired property, and learning the order of importance of the part.

## Token-based/Lexing methods

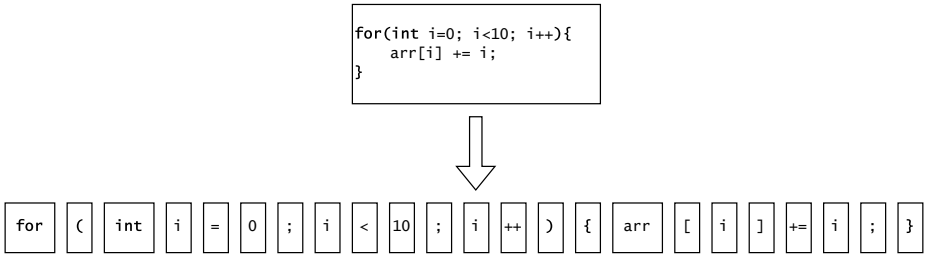


Figure 1: An example illustrating the lexing process [1]

### Token-based/Lexing methods

- These methods (*e.g.* [1]–[3]) use traditional vectorization techniques used in plain text (*e.g.* Tf-Idf, word2vec), or use lexical analysis tools, to generate tokens.
- The tokens are then fed to a model in a similar fashion to plain text classifiers.

**Problem with this approach:**

- Treating source code as natural language texts can miss semantic information of source code [4].

### Abstract Syntax Tree (AST)-based Methods

- These methods use a language-specific parser to extract syntactic paths from source code.
- Each of the path and leaf-values of a path-context is mapped to its corresponding real-valued vector representation, or its embedding.

There have been two methodologies proposed for AST-based techniques.

## Entire AST

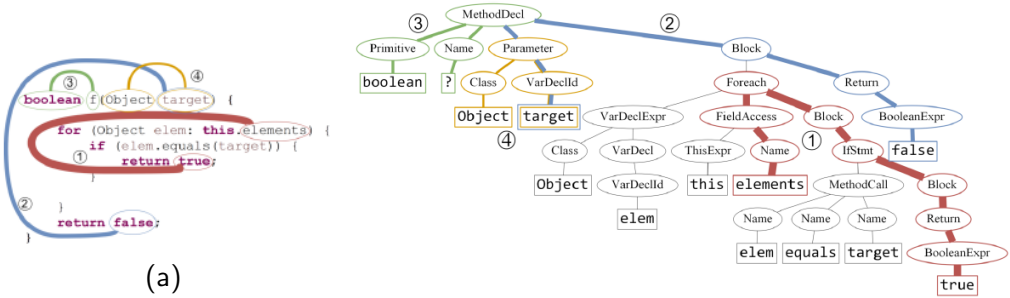


Figure 2: (a) A Java method [5], (b) Its AST [5]

### Entire AST

Given an AST,

- This method ([5]–[7]) works on the entire tree,
- Selects paths from root to leaves on-by-one,
- Assigns a path-context

**Problem with this approach:**

- AST sizes are usually large, so computation complexity is a serious issue.
- Existing models have long term dependency problem [4].

## Splitting AST

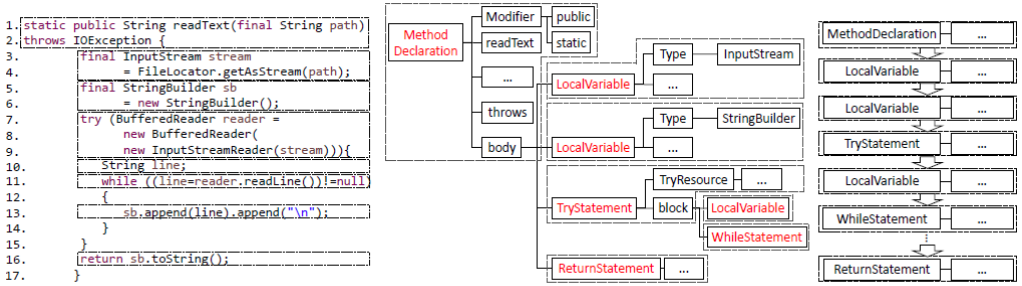


Figure 3: An example of AST statement nodes [4]

### Splitting AST

- In this method [4], a neural network splits the large AST of one code fragment into a set of small trees at the statement level.
- Performs tree-based neural embedding on all statement trees.
- It produces statement vectors, which can represent the lexical and statement-level syntactical knowledge.

**Problem with this approach [4]:**

- A definite efficiency is unknown.
- There is no improvement in performance.

# Methodology

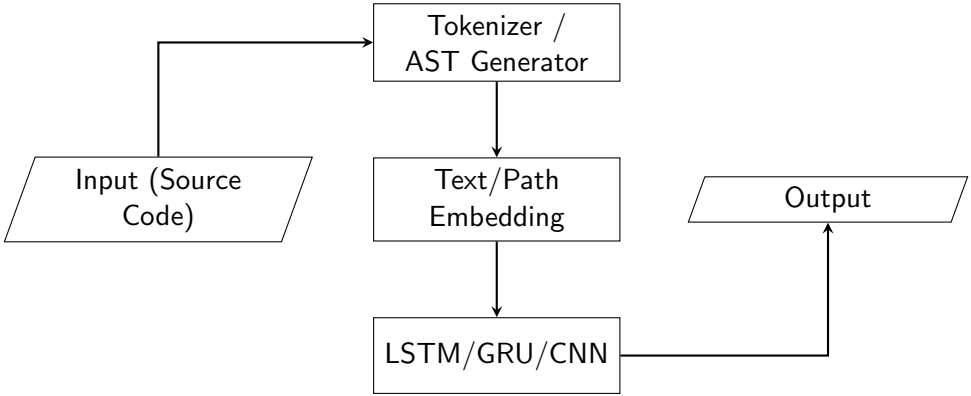


Figure 4: Work-flow of a typical Source Code Vectorization method

# Methodology

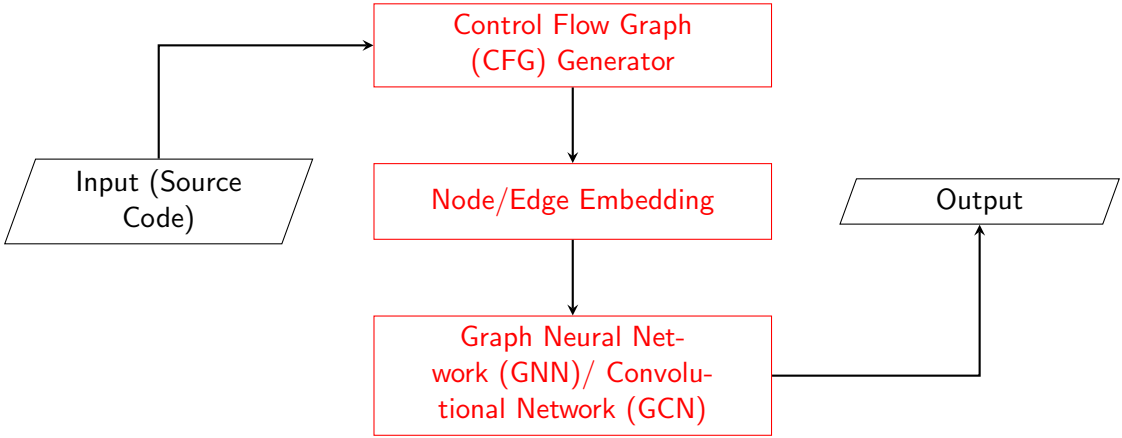


Figure 5: Work-flow of the proposed Source Code Vectorization method

# Experimental Analysis

## Predicting Java method name using Source Code Vectors

**Experimental Setup:**

1. Dataset: Java-med [5]
   - Contains about 4M examples.
   - 1000 top-starred Java projects from GitHub.
     - ★ 800 projects for training
     - ★ 100 projects for validation
     - ★ 100 projects for testing
2. Vectorization method: code2vec [5]

## Predicting Java method name using Source Code Vectors

```
void f(int[] array){
    boolean swapped = true;
    for (int i=0; i<array.length && swapped; i++){
        swapped = false;
        for (int j=0; j<array.length-1-i; j++){
            if (array[j] > array[j+1]){
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1]= temp;
                swapped = true;
            }
        }
}}
```

Listing 1: A Java method with an arbitrary name

Table 1: Possible names for the given method

| Predictions | |
|---|---|
| sort | 98.54% |
| bubbleSort | 0.35% |
| reverse | 0.25% |
| reverseArray | 0.23% |
| heapify | 0.15% |

# Conclusion and Future Work

- Existing methods either fails to capture code semantics,
- Or, Computation complexity is very high.
- Methods with lower complexity has not proved to efficient yet.
- CFGs can capture code semantics better.
- Graph node embedding promises better vectors.
- GNN/GCN can perform selection of high information nodes better.

# References


[1] J. Harer, L. Kim, R. Russell, *et al.*, "Automated software vulnerability detection with machine learning,", Feb. 2018.

[2] J. Cambronero, H. Li, S. Kim, K. Sen, and S. Chandra, "When deep learning met code search," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 964–974.

[3] C. Suman, A. Raj, S. Saha, and P. Bhattacharyya, "Source code authorship attribution using stacked classifier.," in *FIRE (Working Notes)*, 2020, pp. 732–737.

[4] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, "A novel neural source code representation based on abstract syntax tree," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, IEEE, 2019, pp. 783–794.


# References


[5] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "Code2vec: Learning distributed representations of code," *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–29, 2019.

[6] R. Compton, E. Frank, P. Patros, and A. Koay, "Embedding java classes with code2vec: Improvements from variable obfuscation," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 243–253.

[7] D. Azcona, P. Arora, I.-H. Hsiao, and A. Smeaton, "User2code2vec: Embeddings for profiling students based on distributional representations of source code," in *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*, 2019, pp. 86–95.


# Thank you