A study of machine learning approaches to cross-language code clone detection

Daniel Perez M2

Chiba Shigeru Lab

Clone detection

Detect **duplicated code** in programs

- Single language: programs in same language
- Cross language: programs in different languages

Python function

```python
def add(a, b):
    return a + b
```

Motivation for cross-language

Refactoring **large systems**

- Sub-systems often use **multiple languages**
- Code duplication may occur **across sub-systems**

Java function

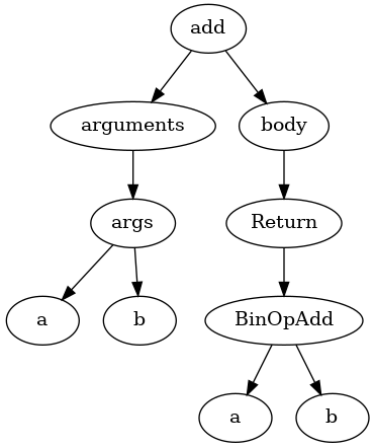
```java
int add(int a, int b) {
    return a + b;
}
```

Current approaches to clone detection

- **Token based** approach
  - Simple
  - Fast
  - Can lack expressiveness

```
["def", "add", "(", "a",
"b", ")", ":", "\n", "\t",
"return", "a", "+", "b"]
```

- **Tree based** approach
  - Powerful
  - Slow, usually $O(n^3)$ or more



Current approaches are designed for clone detection in **single language**

Case study: SourcererCC

Overview

- Current **state-of-the-art** clone detection tool
- Uses **token based** approach
  - Create reverse index of tokens
  - Match tokens in the code fragment with the index
- Performs well for copy-paste induced code clones

Limits for cross-language code clones

- Cross-language code clones usually share fewer tokens
- Would at least need some kind of cross-language mapping

Difficulty of cross-language clone detection

Factorial in Java

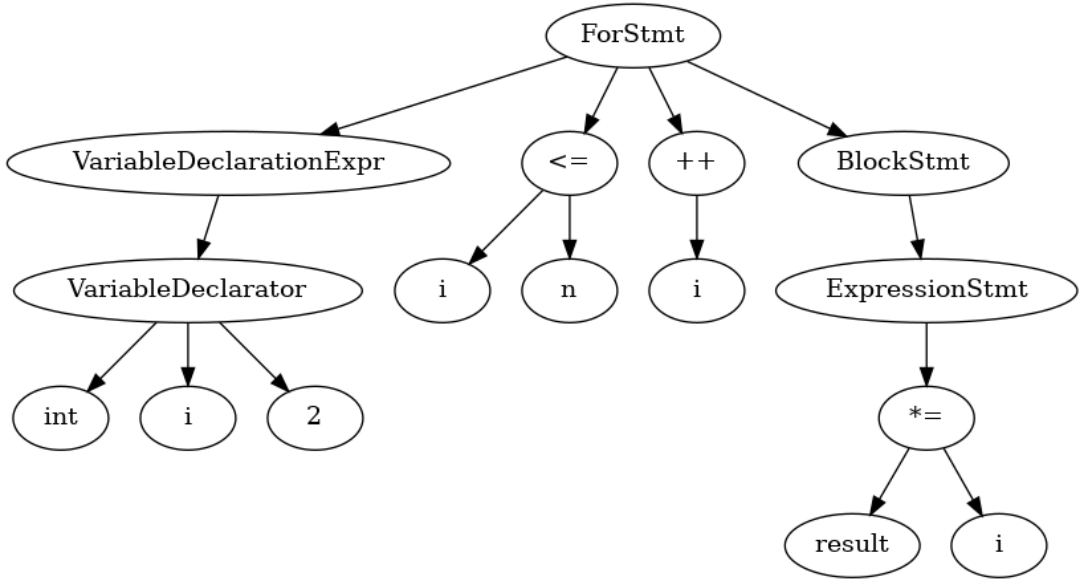
```java
public int factorial(int n) {
  int result = 1;
  for (int i = 2; i <= n; i++) {
    result *= i;
  }
  return result;
}
```

Factorial in Python

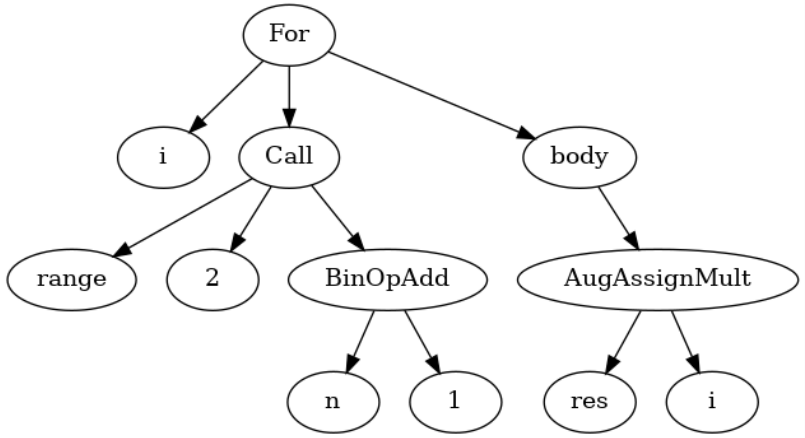
```python
def fact(n):
  res = 1
  for i in range(2, n + 1):
    res *= i
  return res
```

Difficulty of cross-language clone detection

Java factorial loop AST



Python factorial loop AST



Our proposal

General idea

Learn AST structure representation

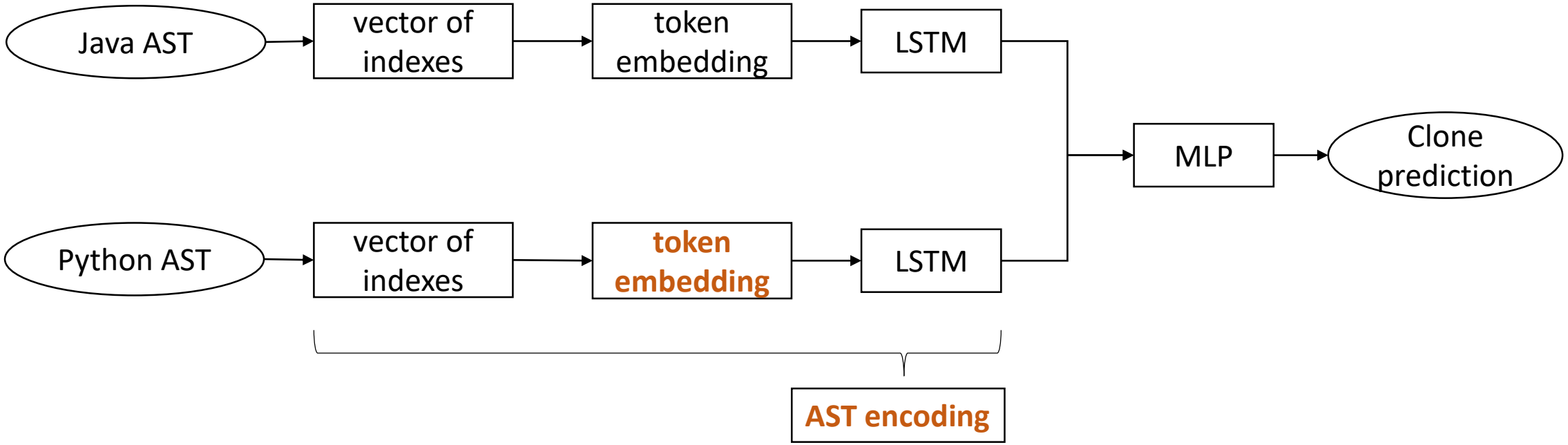
Use learned representation for cross-language clone detection

Overview

- Find a token-level vector representation
- Use end-to-end supervised machine learning to learn AST representation

System overview

System currently supports Java and Python

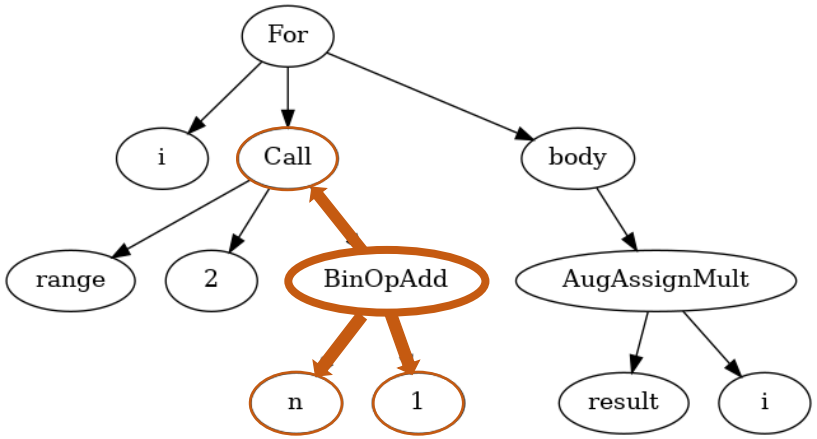


https://github.com/tuvistavie/suplearn-clone-detection

Token embeddings generation

We propose **tree-based skipgram**

```
For, i, Call, range, 2, BinOpAdd,
n, 1, body, AugAssignMult, result
```



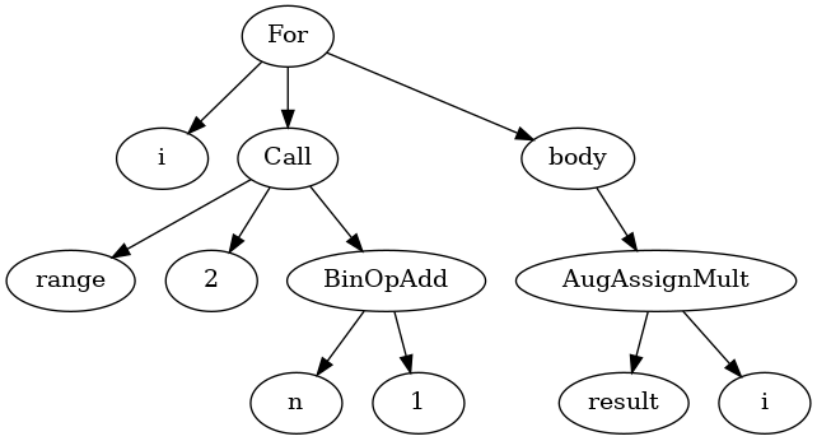
1. Generate a vocabulary
2. For each “target” node, generate “context” nodes
3. Feed (target, context) as input, output of a single layer MLP
4. Use hidden layer as embeddings

https://github.com/tuvistavie/bigcode-tools

AST encoding

Feed AST to recurrent neural network

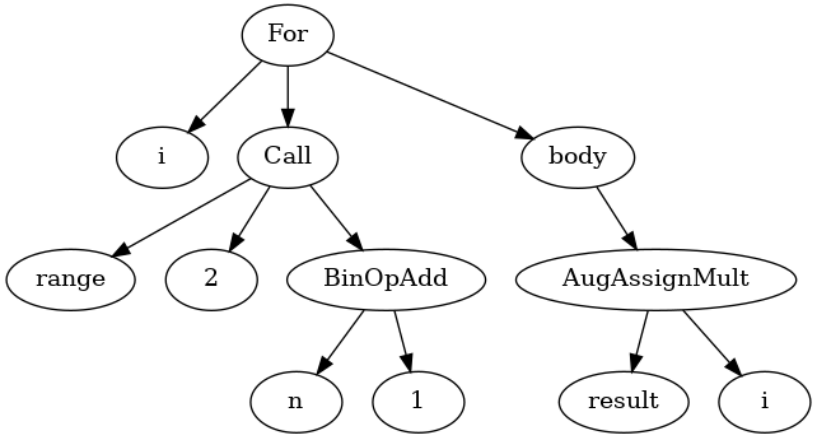
1. Generate tokens sequence from AST using depth-first search
2. Map each token to its vector representation
3. Feed the sequence to an LSTM



Prefix depth-first traversal gave us the best results

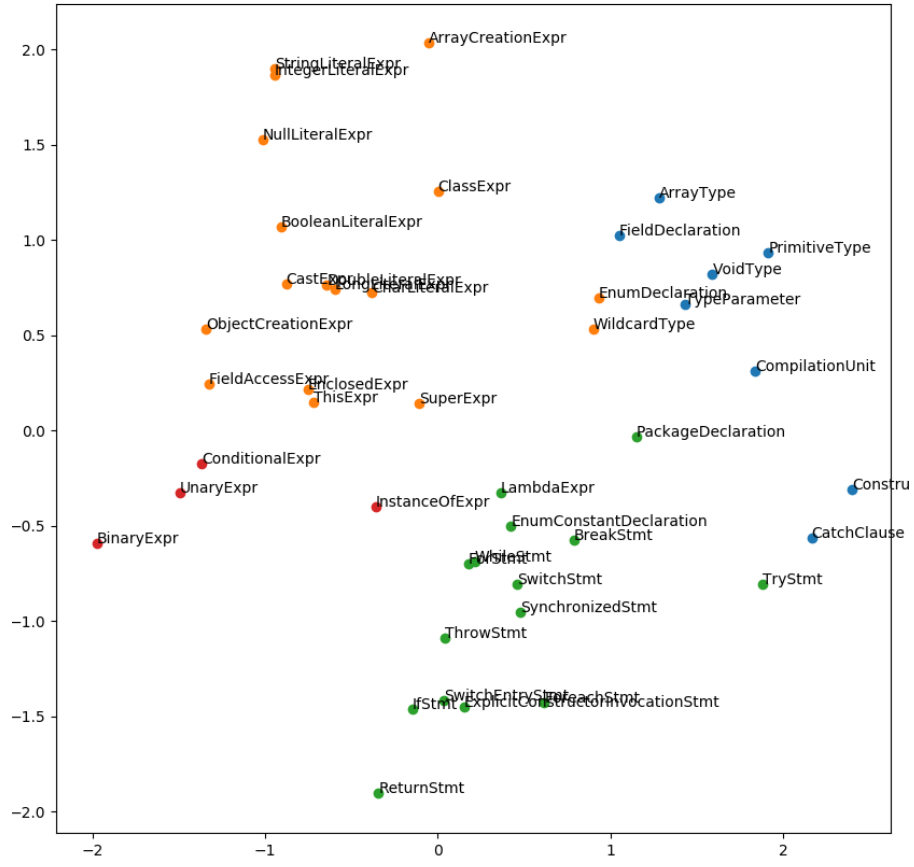
Token embeddings generation experiment

- Dataset
  - Java: all Apache projects
    - ~400k files
    - Apache2 license
  - Python: popular projects on GitHub
    - ~150k files
    - Non-viral license (MIT, BSD, Apache)
- Hyper parameters
  - Use identifiers or not
  - Ancestors window size: 1, 2, 3
  - Children window size: 0, 1, 2, 3
  - Include siblings or not
  - ML related parameters



Java token embeddings results

- Semantic somewhat preserved
  - Statement, expressions, declarations clustered more or less correctly
  - e.g. Token closest from ForStmt is WhileStmt
- Hyperparameters results
  - Two level of ancestors works well
  - One level of children is enough
  - Siblings add too much noise



Clone detection experiment goals

1. Evaluate the effectiveness of trained embeddings
2. Tune our model and evaluate its capacity to learn clones
3. Compare our model performance with other clone detection tools

Dataset creation for clone detection

- Dataset must have following properties
  - Python and Java implementation
  - Implement same functionality/program
- **Competitive programming** is a good candidate
  - Scraped AtCoder[1] website

Measure	Value
Problems count	576
Files count	44,620
Avg. Files/problem	77

[1] https://atcoder.jp

Clone detection experiment results

Training and test with 20% of clones in samples

- Cross-language is harder than single-language detection
- Pre-trained embeddings help improving the model
- Removing identifiers loses information

Java-Python clone detection

	F1	Precision	Recall
Pretrained embeddings	0.66	0.55	0.83
Untrained embeddings	0.61	0.49	0.82
No identifiers	0.51	0.40	0.71

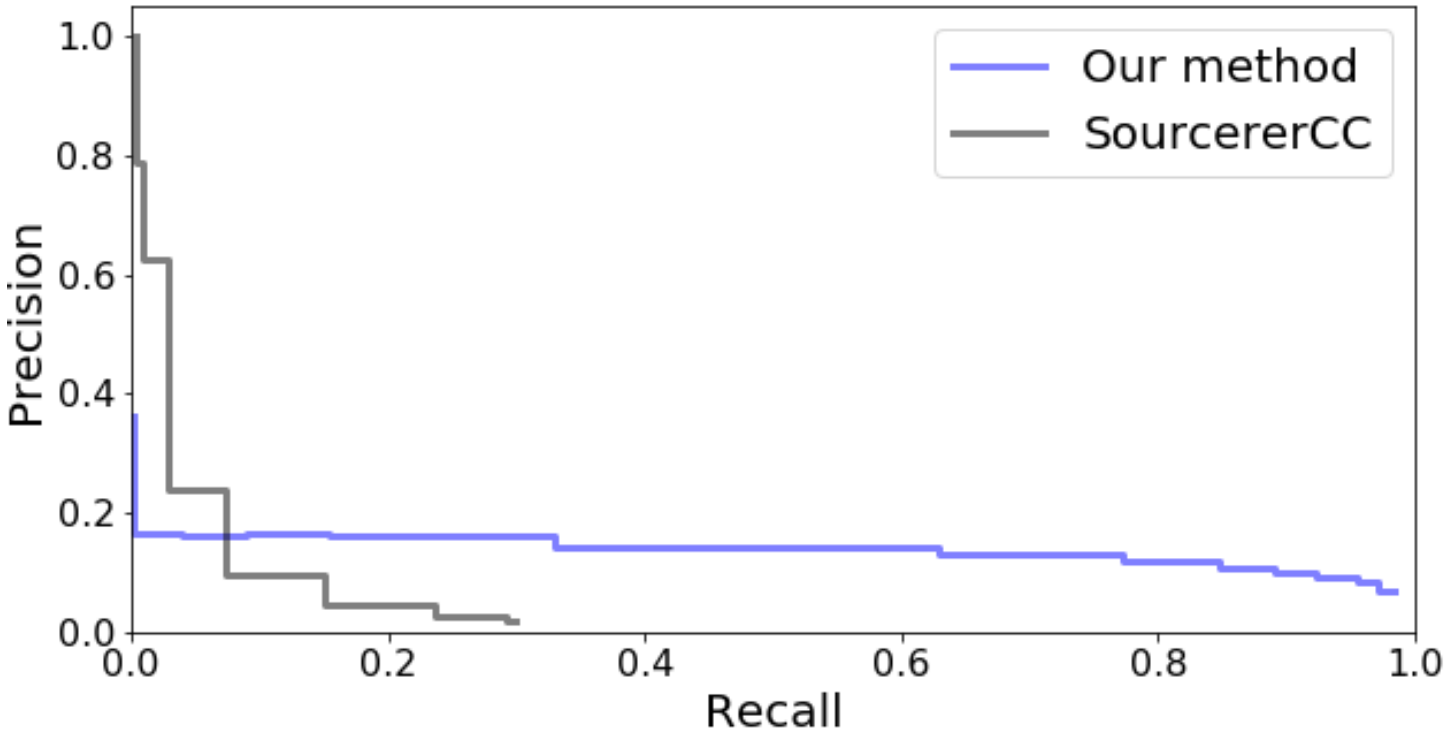
Java-Java clone detection

	F1	Precision	Recall
Pretrained embeddings	0.77	0.67	0.92
Untrained embeddings	0.74	0.65	0.85
No identifiers	0.69	0.56	0.90

Comparison with SourcererCC

Java code clone experiments using our test set, about 1000 files
Our method currently only takes pairs of code: 1000^2 samples for this experiment

Precision/recall curve



	F1	Precision	Recall
Our method	0.21	0.12	0.85
SourcererCC	0.05	0.63	0.03

Related work

- Hui-HuiWei & al, Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code, IJCAI'17
  - **Single-language** clone detection
  - Supervised learning approach
  - Tree-LSTM like model
  - Generates hash-code from AST

- Kraft & al, Cross-Language Clone Detection, SEKE'2008
  - Uses **common intermediate representation** between languages
  - Detects clones between VB.NET and C#

Future work

- Make **better use** of the **AST structure**
  - Try **other models** for encoding AST (e.g. Gated Graph Sequence NN)

- Improve the system to be able to run in **linear time**
  - Add trainable hash layer to the model
  - Index hashed vectors using a reversed-index

Summary

- Proposed a method to **learn AST structure**
  - Method and hyperparameters set to generate token level embeddings
  - End-to-end supervised learning model

- Applied our idea to detect **cross-language code clones**
  - Cross-language code clone detection dataset
  - Trained model on code clone dataset

Source code available at

https://github.com/tuvistavie/bigcode-tools

https://github.com/tuvistavie/suplearn-clone-detection

Supporting slides follow

What is clone detection?

Detecting **duplicated code** in programs

Base technology for **many applications**

- Intra-project: find duplicates in a single project
  - Find refactoring opportunities
  - Quality evaluation metrics
- Inter-project: find duplicate from other projects
  - Copyright violation
  - Code search
  - Detect malicious software

Program representations

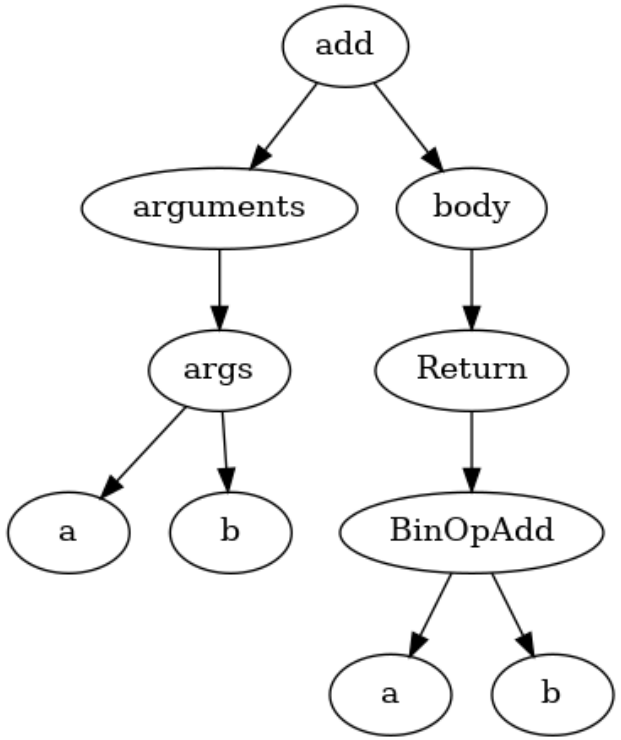
Source program

```
def add(a, b):
    return a + b
```

Token Representation

```
["def", "add", "(", "a",
"b", ")", ":", "\n", "\t",
"return", "a", "+", "b"]
```

AST Representation

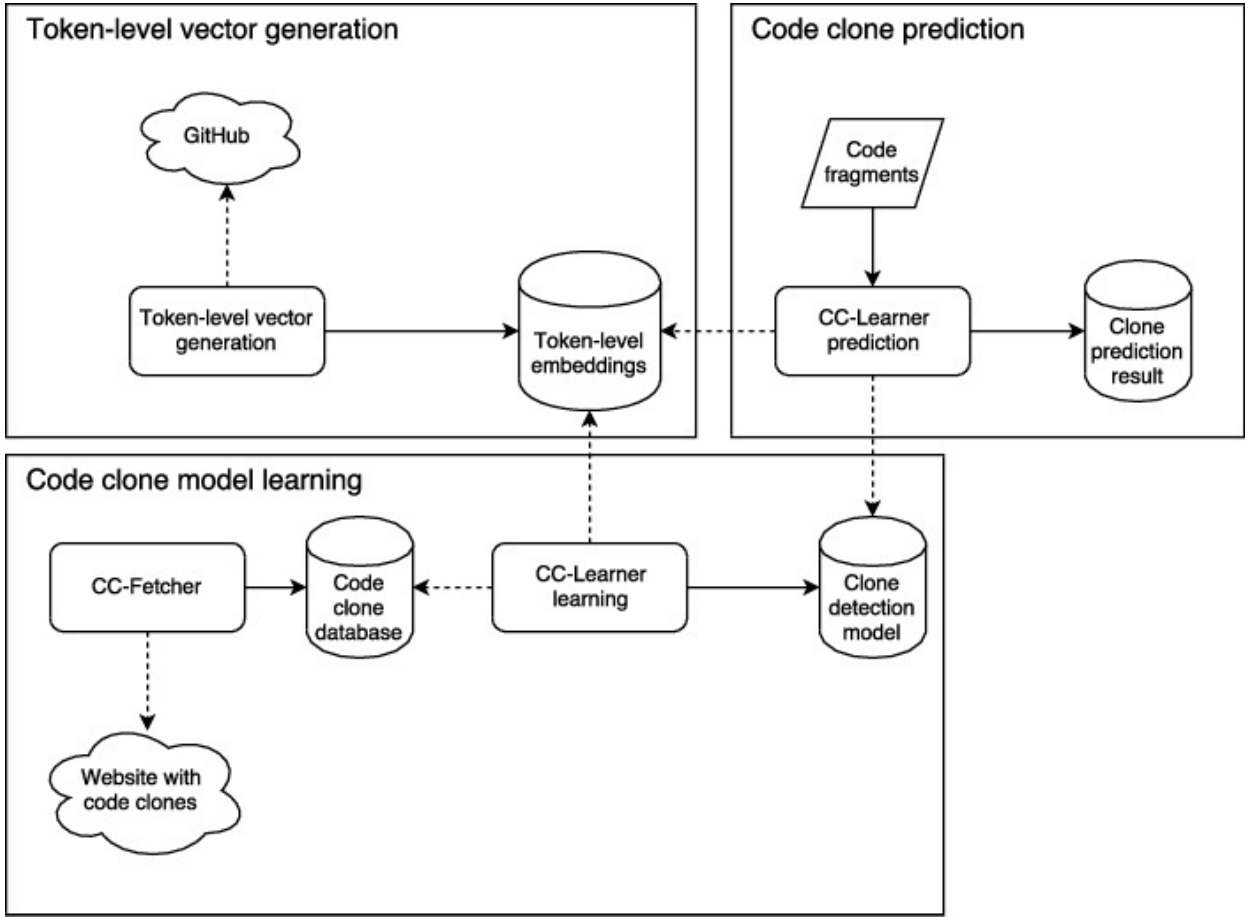


Type of clones

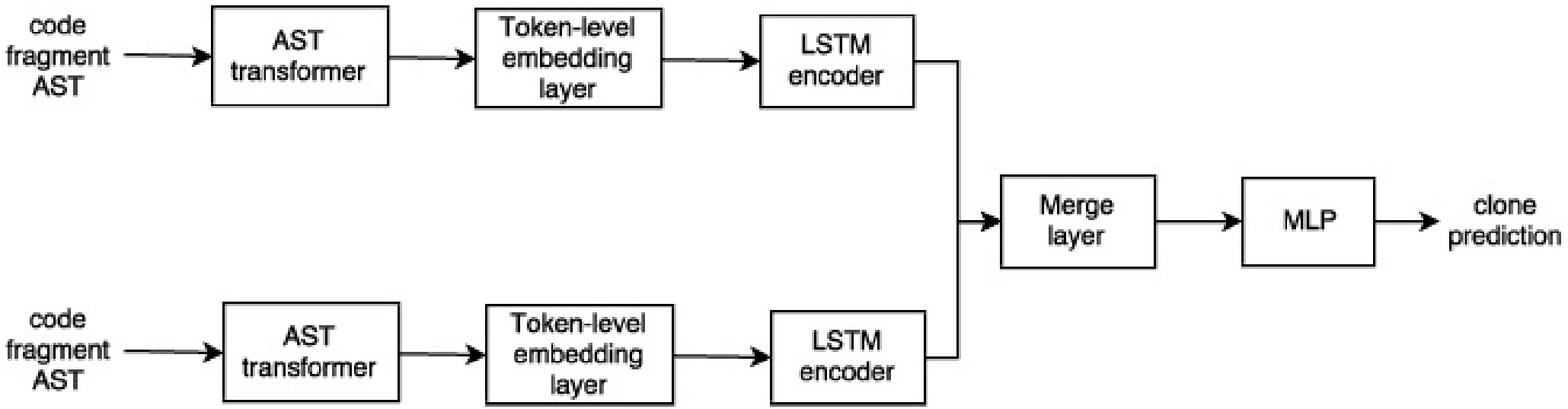
4 types of code clone

- Type I: changes in spacing and comments
- Type II: changes in identifier and literals
- Type III: syntactically similar with changes in statements
- Type IV: syntactically dissimilar with same functionality

System overview



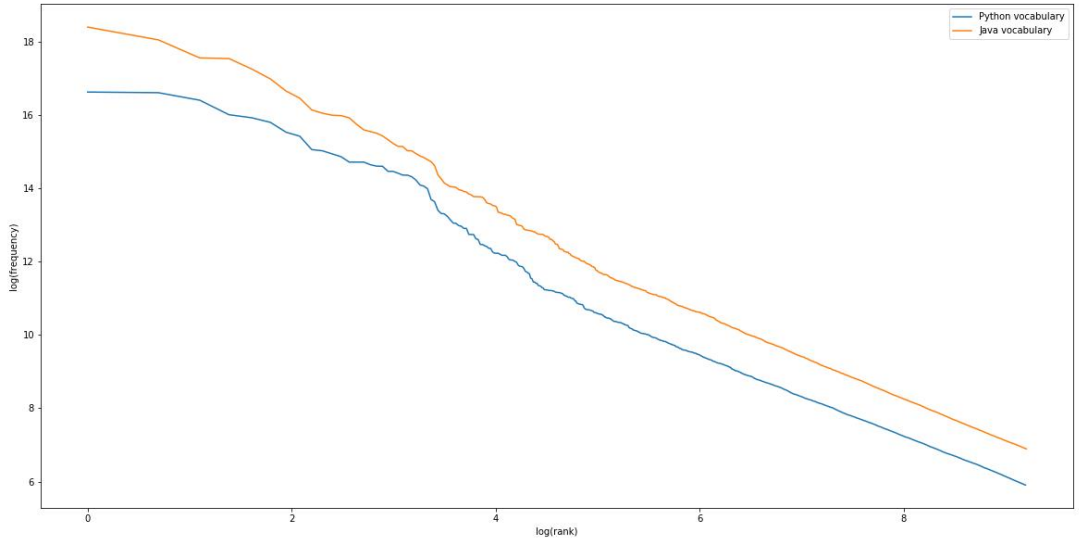
Model overview



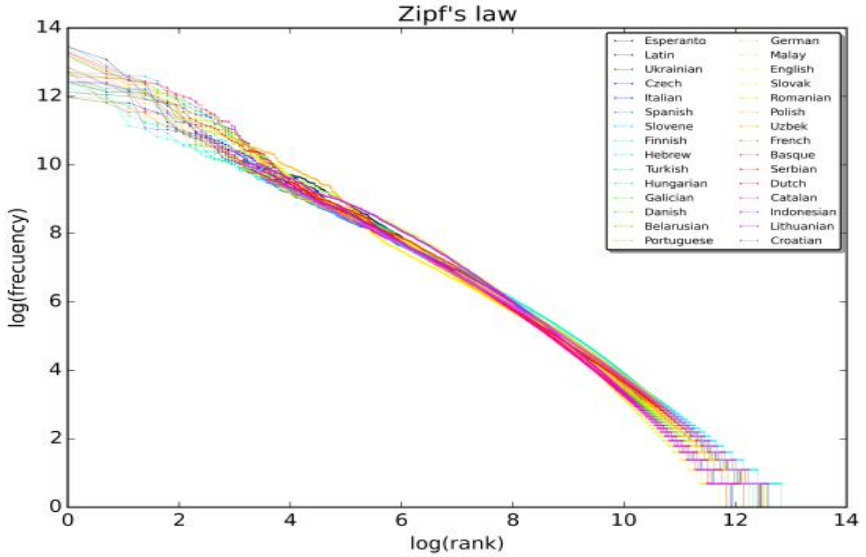
Zipf’s law

Frequency of any word is inversely proportional to its rank

Programming language frequency/rank log graph (not normalized)

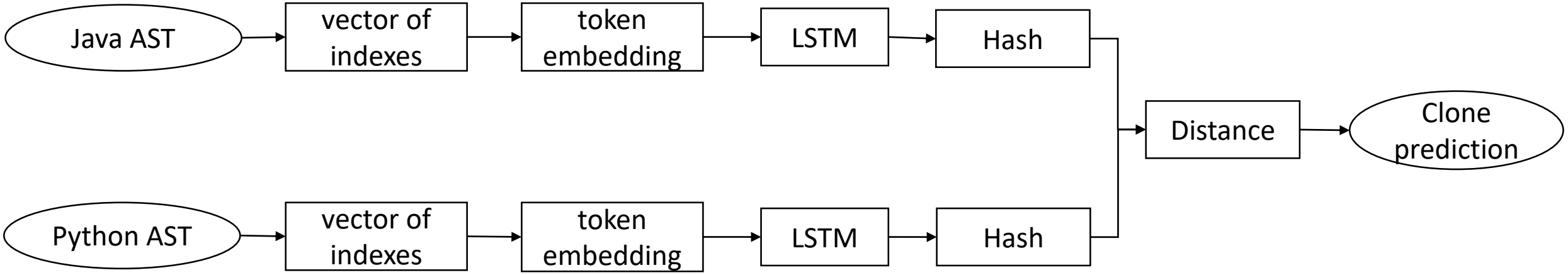


Natural languages frequency/rank log graph



Source: `https://en.wikipedia.org/wiki/Zipf’s_law`

Model with hash layer



Source code normalization

- Remove identifiers
- Remove custom types

```
public int add(Custom a, int b) {
    return a.doSomething(b);
}
```

```
public int METHOD(TYPE VAR, int VAR) {
    return VAR.METHOD(VAR);
}
```

Clone false-positive example

Code fragment 1

```java
import java.util.Scanner;

public class Main{
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int ans = Integer.MAX_VALUE;
    for (int i = 1; i <= n; i++) {
      int j = n / i;
      ans = Math.min(
        ans, Math.abs(i - j) + n - i * j);
    }
    System.out.println(ans);
  }
}
```

Code fragment 2

```java
import java.util.*;

class Main {
  public static void main(String[] args){
    Scanner read = new Scanner(System.in);
    int x = Integer.parseInt(read.nextLine());
    int c = 0;
    int sum = 0;
    while (sum < x) {
      c++;
      sum += c;
    }
    System.out.println(c);
  }
}
```

Embeddings hyperparameters

Parameter	Value
Ancestors windows size	2
Descendants window size	1
Siblings	False
Embeddings size	100
Vocabulary size	10000

Clone detection hyperparameters

Parameter	Value
Vocabulary size	10000
Embeddings dimension	100
LSTM	Stacked bidirectional — {100, 50}
Multilayer perceptron	1 layer, 64 units
Optimizer	Adam

Deckard, Jiang & al., ICSE'07

- Tree-based clone detection
- Rule-based vector generation for AST
- Cluster vectors using LSH
- Support multiple languages
- Designed for single-language clone detection
  - Assume clones ASTs have similar structure

SourcererCC, Sajnani & al., ICSE'16

- Token-based clone detection
- Uses reversed-index to index tokens
- Optimized for large-scale
  - Fast
  - Low memory
- Assumes clones have large number of tokens in common
  - Works best for clones introduced by copy-paste