Code Clone Detection

Group 5 | Siddharth | Aamir

Github Repository -https://github.com/sedflix/code_clone_detection

What's "clone"?

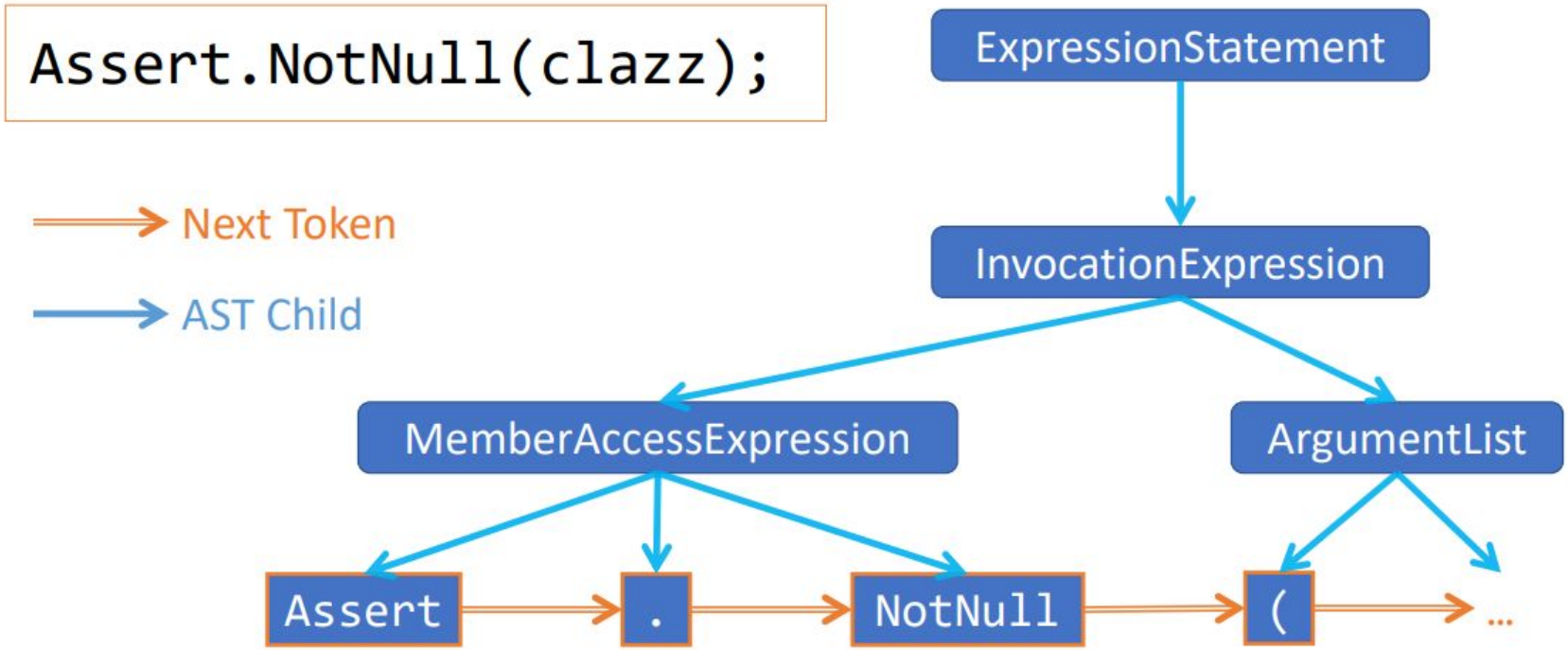
Source code	Type-1 clone	Type-2 clone
int main() { int x = 1; int y = x + 5; return y; }	int main() { int x = 1; int y = x + 5; return y;//output }	int func2() { int p = 1; int q = p + 5; return q; }

Type-3 clone	Type-4 clone
int main() { int s = 1; int t = s + 5; t = t/++s; return t; }	int func4() { int n = 5; return ++n; }

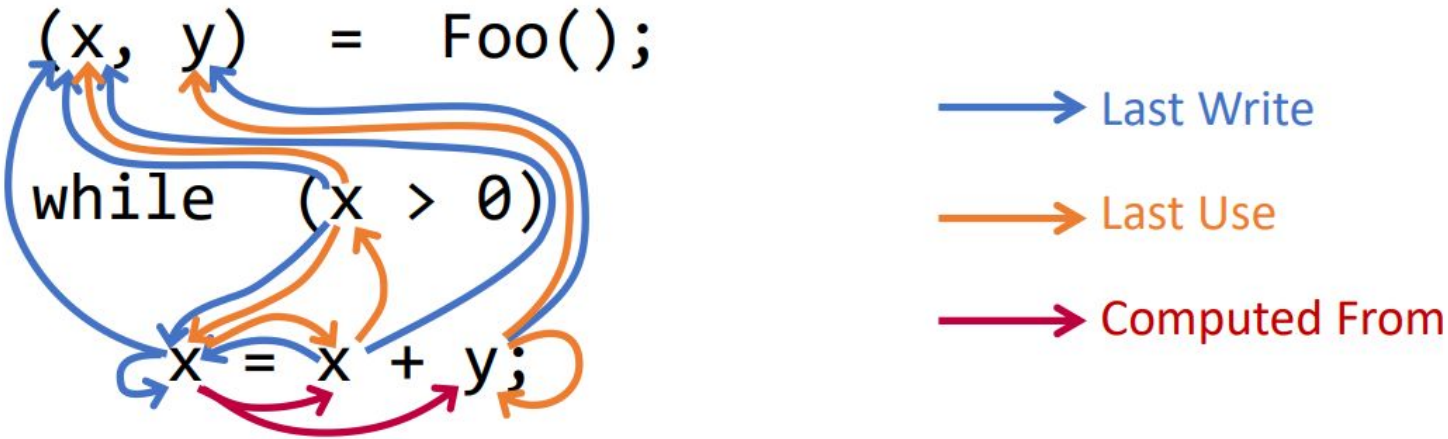
Duplicates in Open Source

Dataset	# Files (x1000)	% duplicates
C# ICLR'19	28.3	10.6
Concode- Java*	229.3	68.7
Java GitHub Corpus	1853.7	24.8
Java-Small	79.8	4.7
Java-Large	1863.4	20.2
JavaScript-150k	112.0	20.7
Python-150k	126.0	6.6
Python docstrings v1*	105.2	9.2
Python docstrings v2*	194.6	31.5

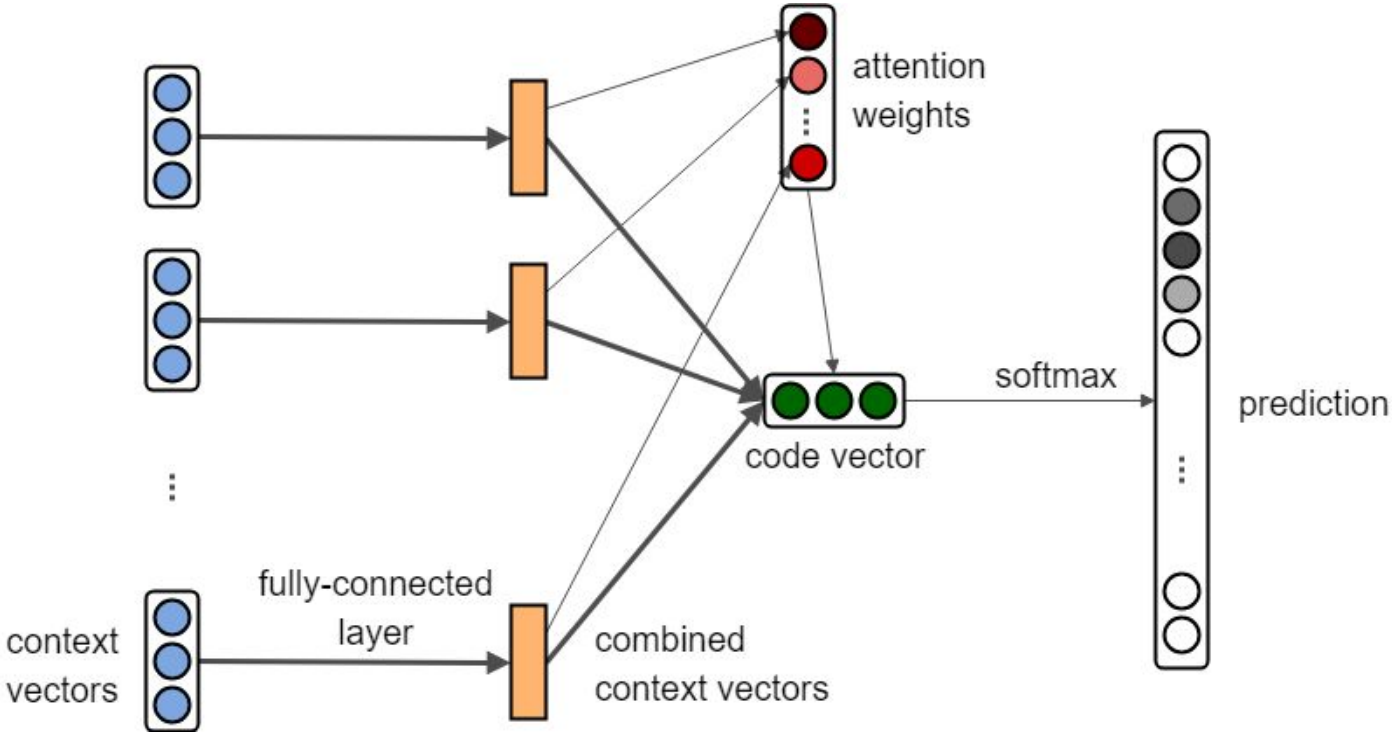
Representing Code as Graph: AST



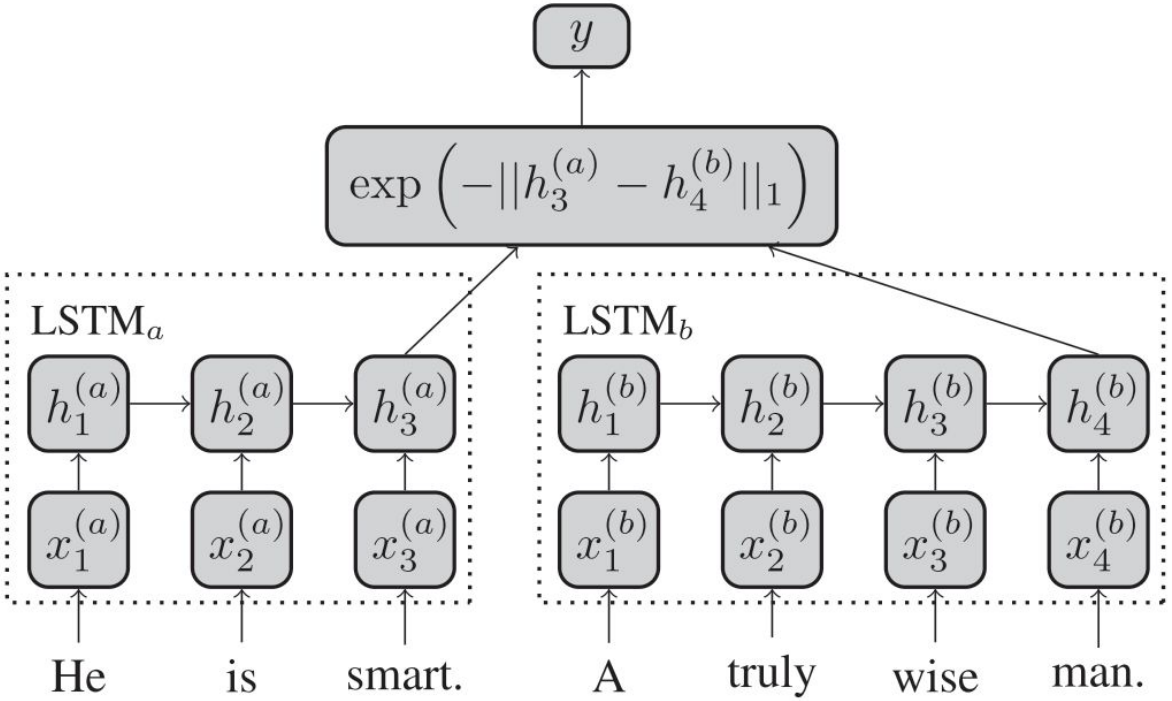
Representing Code as Graph: Dataflow



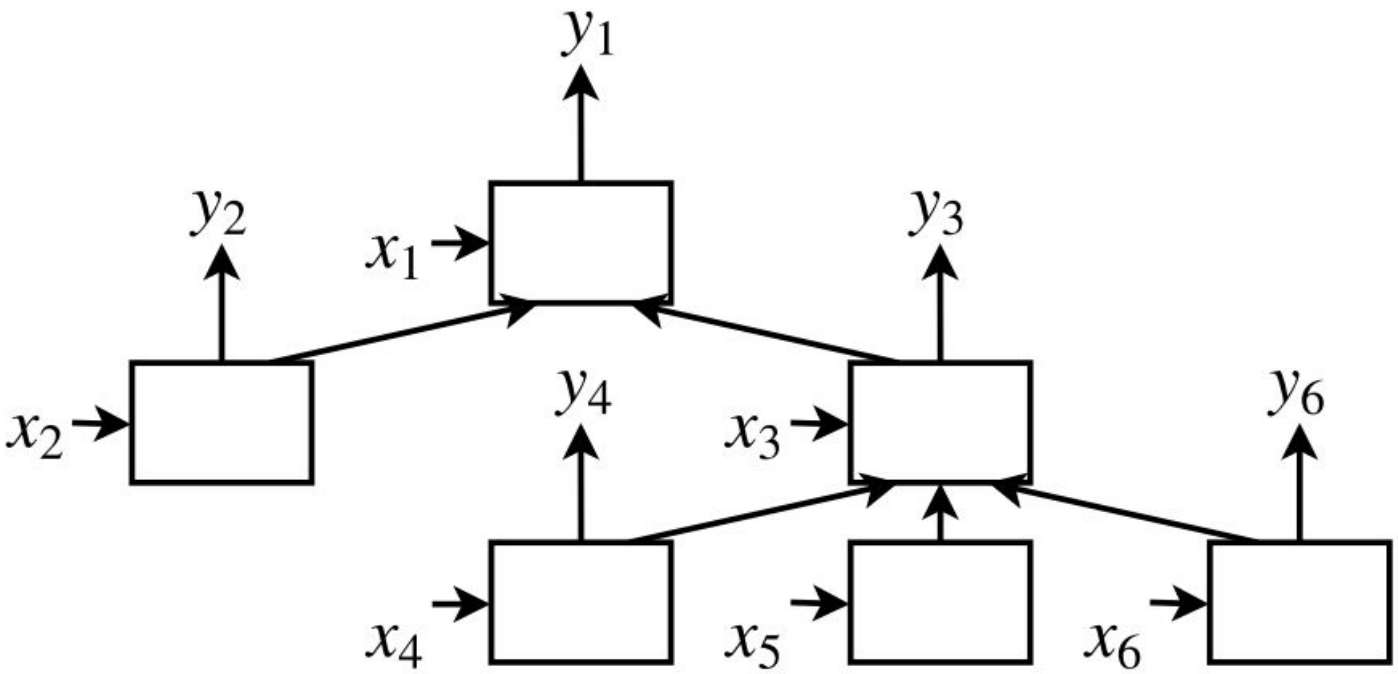
Building Blocks: code2vec



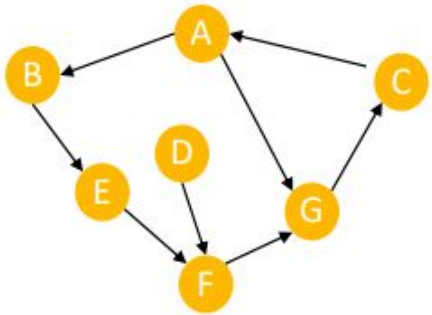
Building Blocks: Siamese RNN



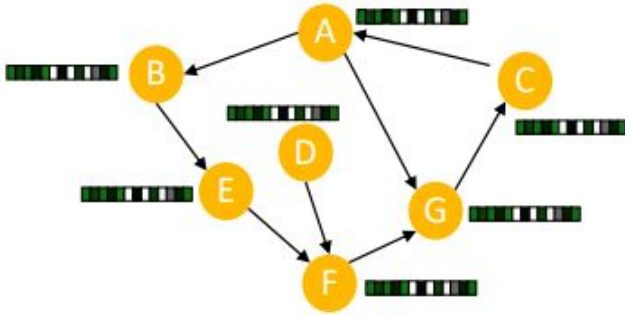
Building Blocks: Tree RNN



Building Blocks: GNN: Setup

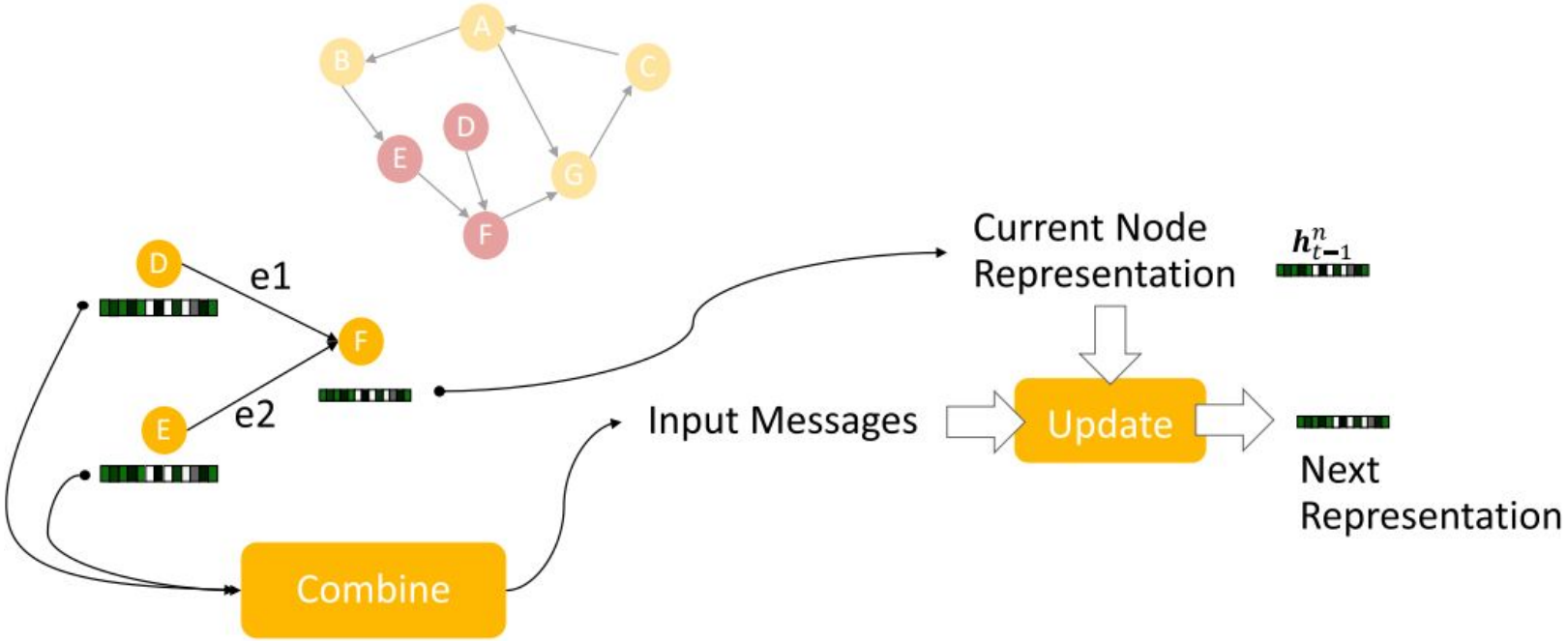


Graph Representation
of Problem

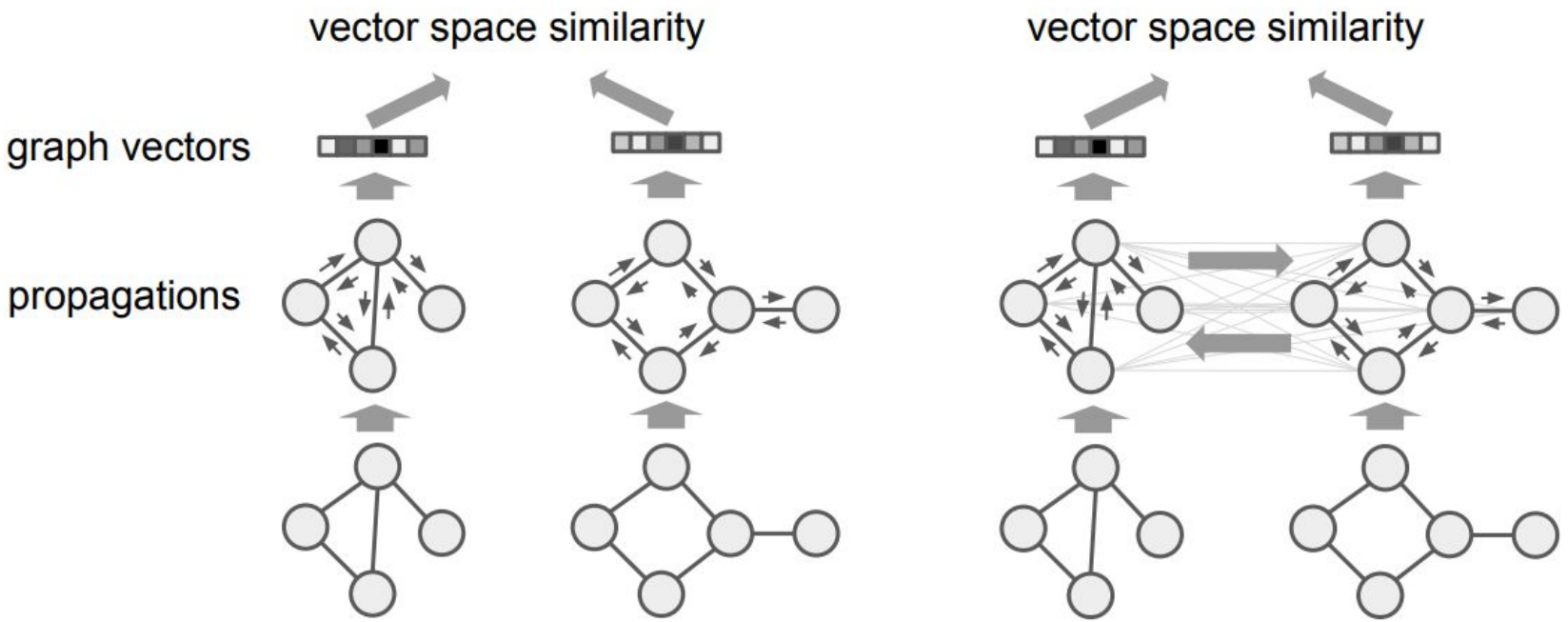


Initial Representation
of each node

Building Blocks: GNN: Message Passing



Building Blocks: Graph Matching



GCN Propagation Rule

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$$

- $H^{(l+1)}$ - Hidden features at next layer
- D - Degree Matrix of A
- A - Adjacency Matrix with Self loops
- $H^{(l)}$ - Hidden features of current layer
- W - Weight matrix
- σ - Non linear activation function (Relu)
- $*H^{o}$ - X (initial representation of node)

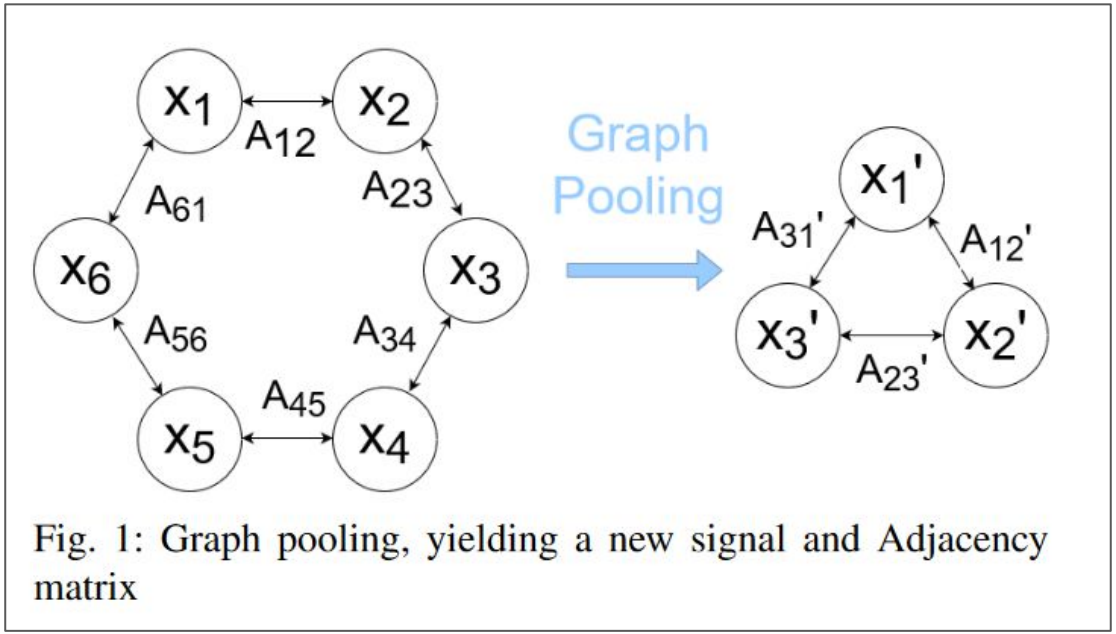


Fig. 1: Graph pooling, yielding a new signal and Adjacency matrix

https://arxiv.org/pdf/2004.03519v1.pdf

Top-k Pooling: Top-k Pool pools using a trainable projection vector $p^{(l)} \in R^f$ and select the top-k indices of the projection $X^{(l)}p^{(l)}$ and the corresponding edges in $A^{(l)}$.

Dataset: Big Clone Bench

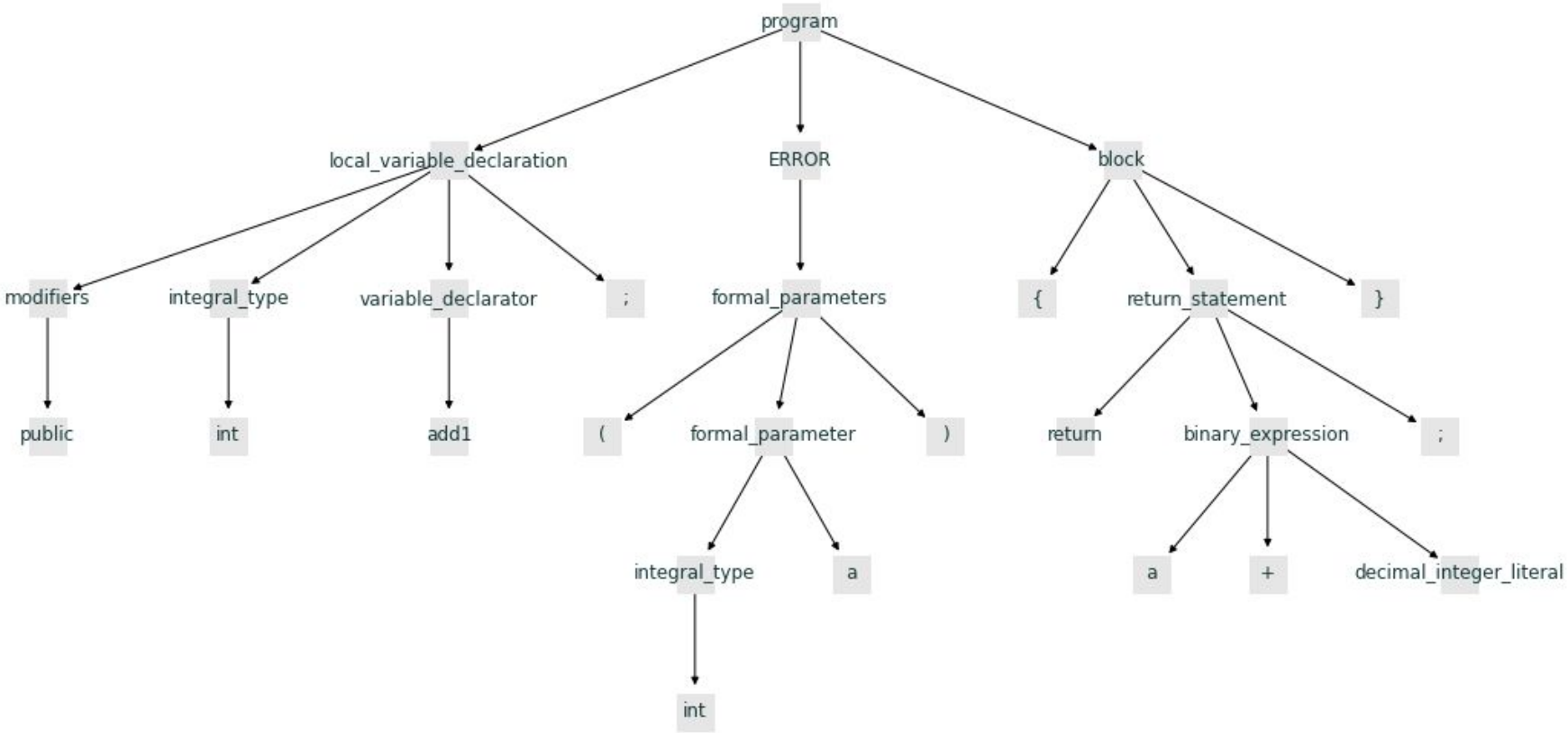
- Code Fragments: 59,688
- True clone pairs: 95.7%
- Max Tokens: 16,253
- Avg tokens: 227
- Max AST depth: 192
- Avg AST depth: 9.9
- Max AST nodes: 15,217
- Avg AST nodes: 206

Data Preprocessing

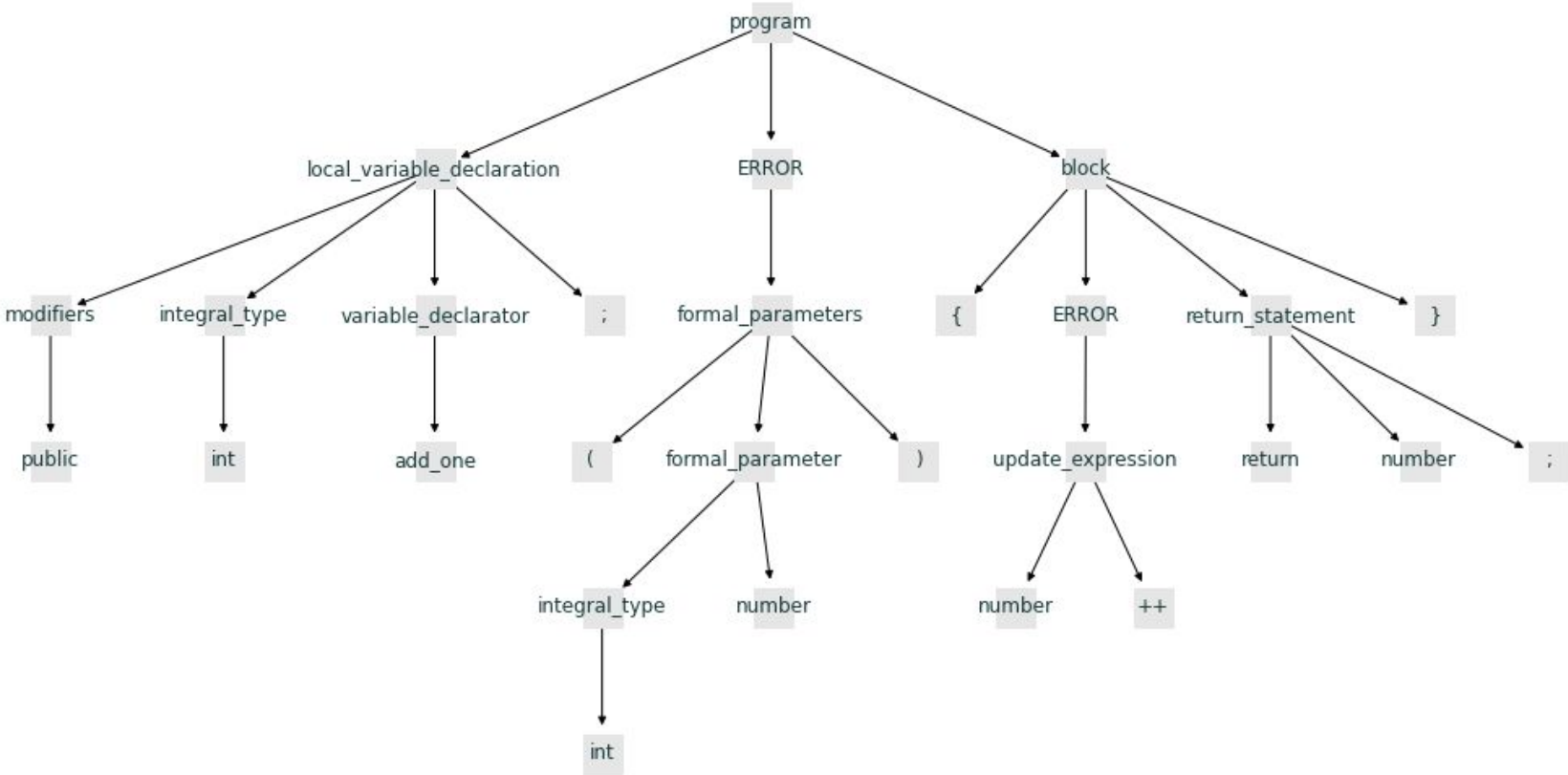
- Code2Vec embedding for dataset
- TreeSitter -> for getting AST from code at any level
- FastText Embedding for dataset - We have trained fasttext embeddings on the inorder ordering of the AST because around 50% unique tokens were OOV when using code2vec

Example 1 :
Simple add 1 to given number program

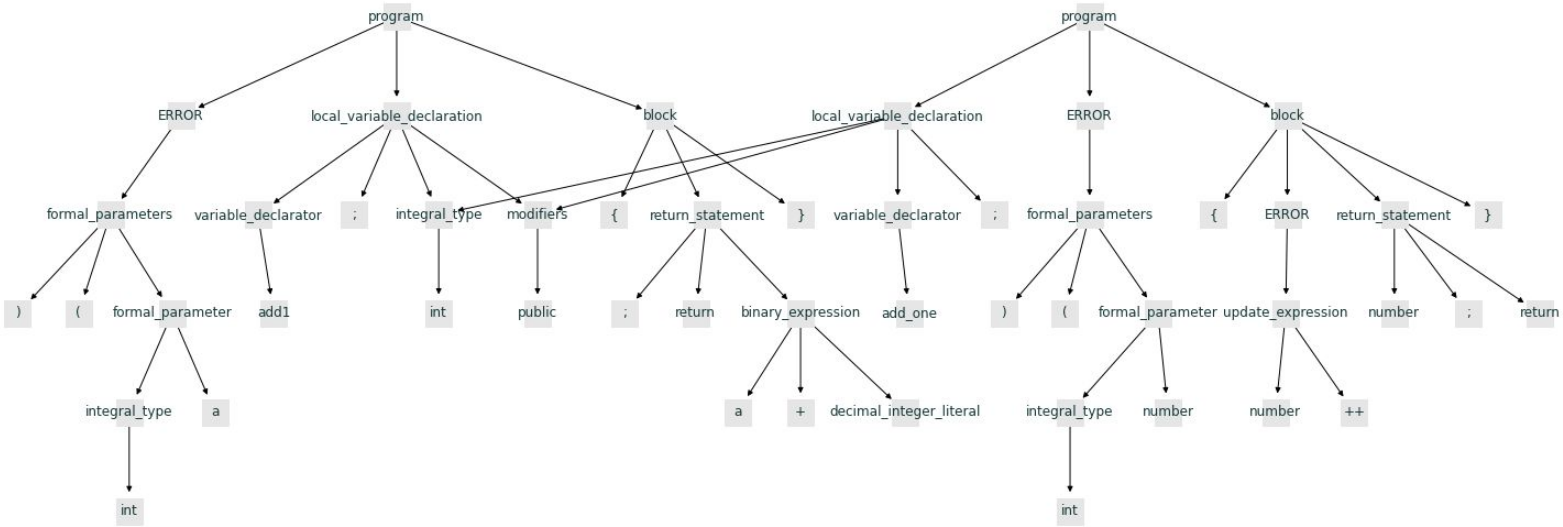
Program 1


program
local_variable_declaration
ERROR
block
modifiers
integral_type
variable_declarator
;
formal_parameters
{
return_statement
}
public
int
add1
(
formal_parameter
)
return
binary_expression
;
integral_type
a
a
+
decimal_integer_literal
int

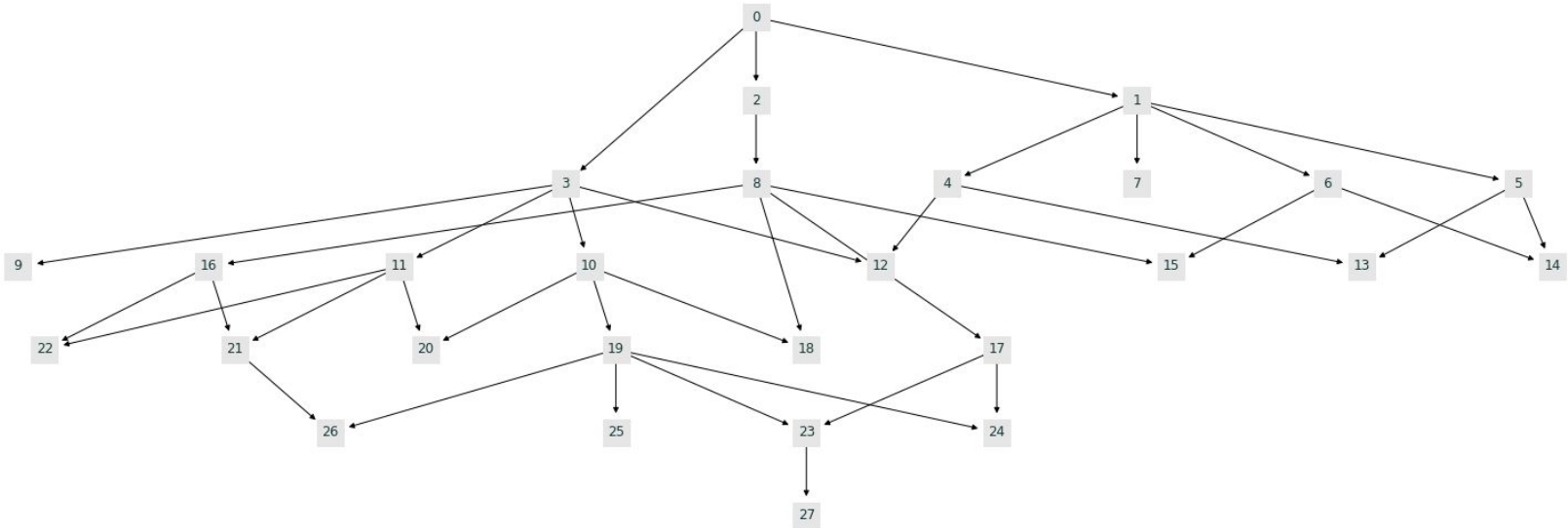
Program 2


program
local_variable_declaration
ERROR
block
modifiers
integral_type
variable_declarator
;
formal_parameters
{
ERROR
return_statement
}
public
int
add_one
(
formal_parameter
)
update_expression
return
number
;
integral_type
number
number
++
int

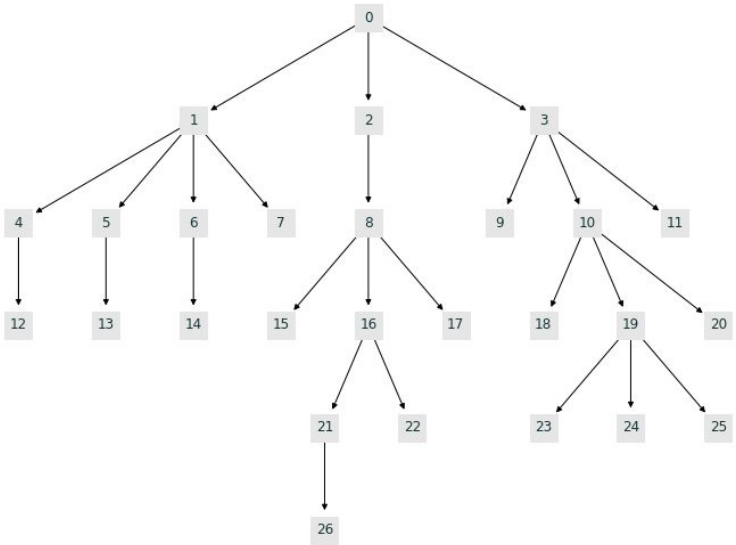
compose: considering location of text too

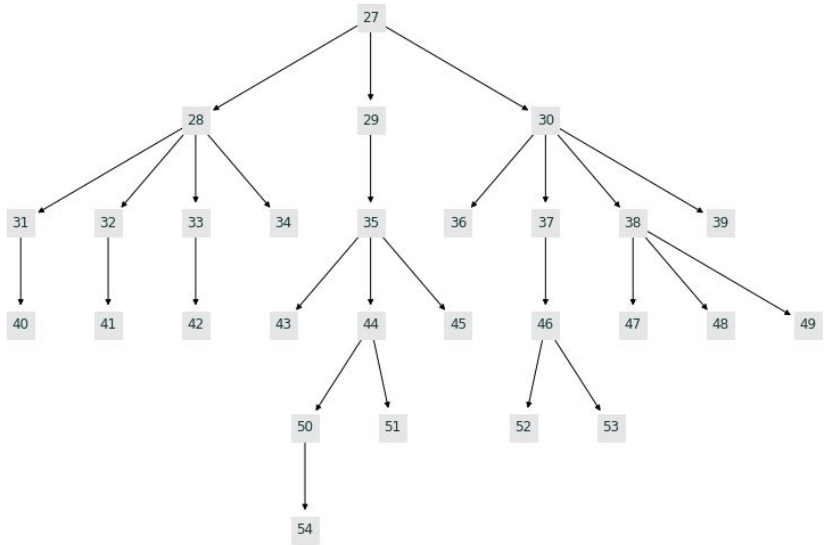


compose: based on ast structure only



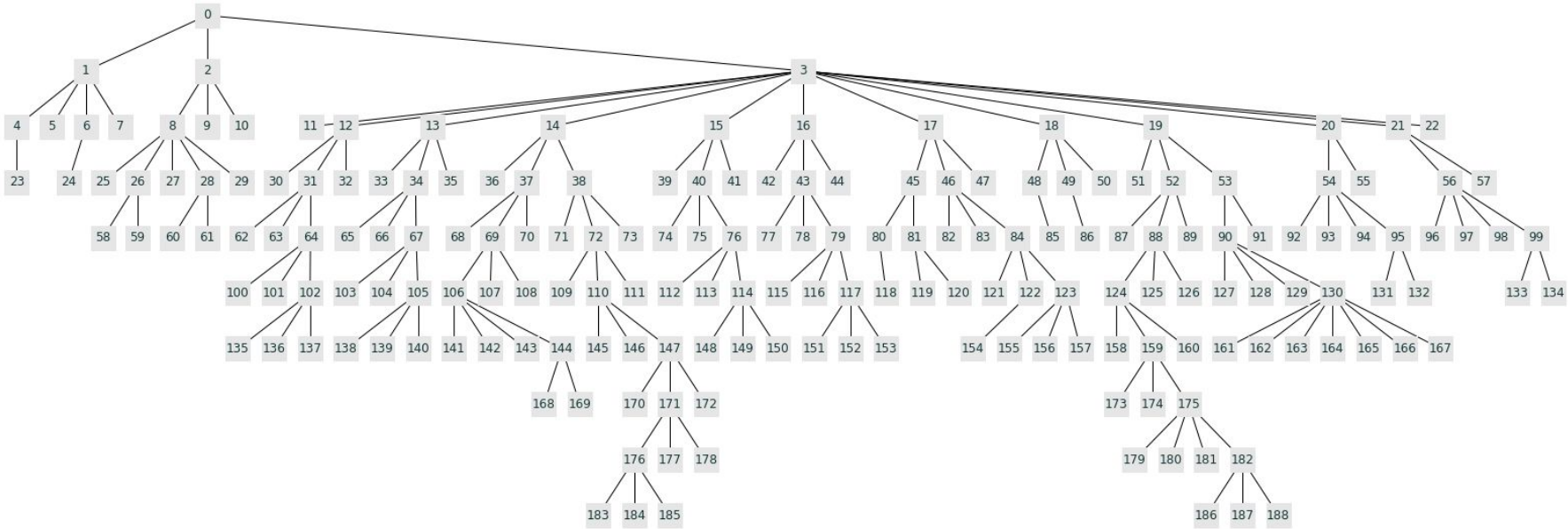
Disjoint Ast



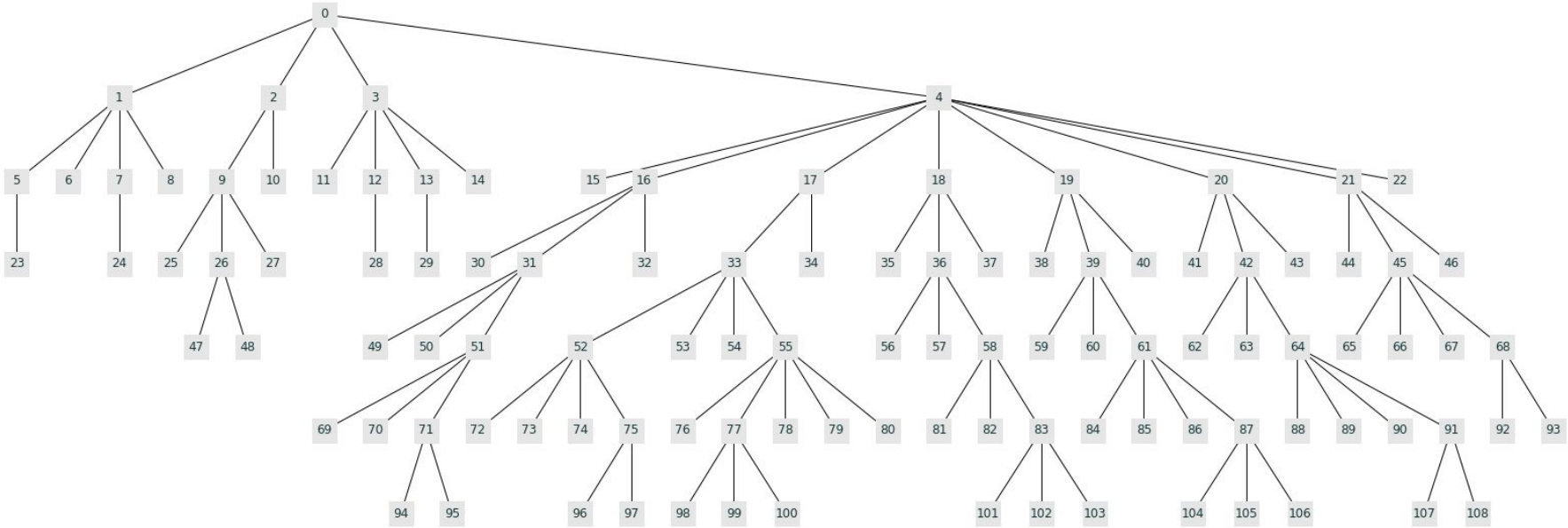


Example 2 :
Real World Programs

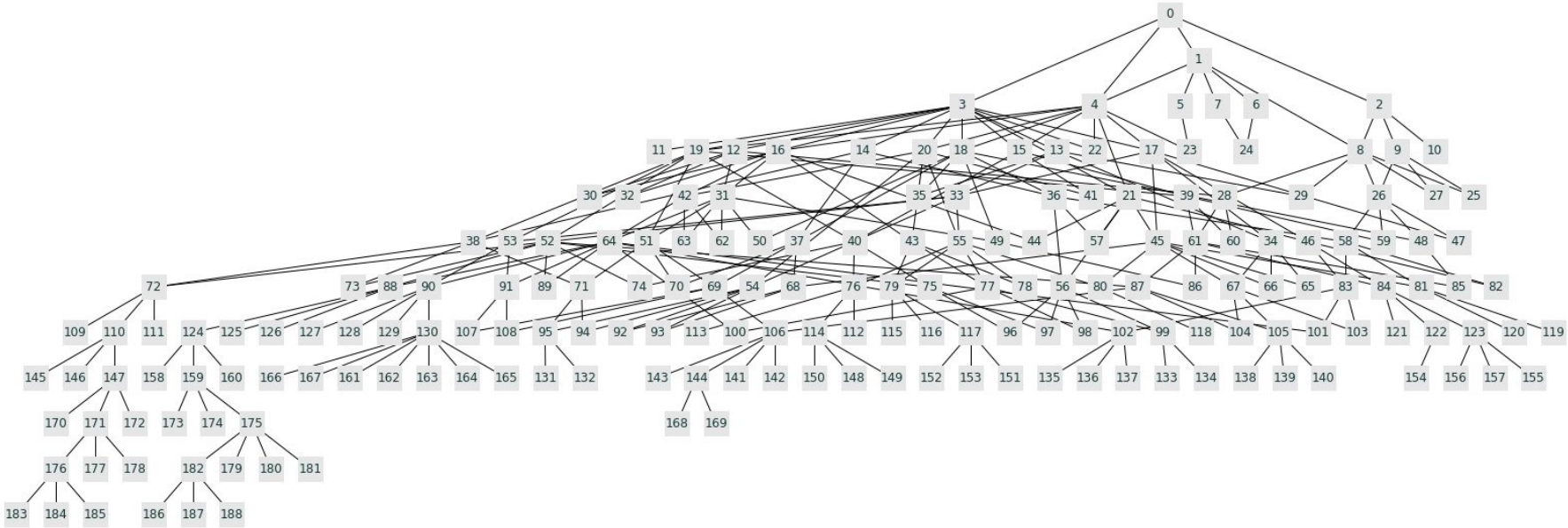
Program 1


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188

Program 2


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108

Overlapping structure



Model	Embedding	Accuracy
RNN	Normal	0.83
RNN	Code2Vec	0.86
GraphConv - TopKPooling*	Code2Vec	0.57
GraphConv (AST merged by tokes and their location)	Code2Vec	0.84
GraphConv (AST merged by their structure)	FastText	0.84
GraphConv (Disjoint AST)	FastText	0.87
Siamese of GraphConv	**FastText**	**0.90**

*Only instance where GraphConv without TopKPooling, otherwise TopKPooling is used with GraphConv everywhere else

CodRep'18

Registered participants:

1. JetBrains Research, HSE
2. Microsoft Research
3. The University of Edinburgh
4. Inria
5. Siemens Technology and Services Private Limited
6. source{d}
7. Universidad Central "Marta Abreu" de Las Villas
8. IPT Sao Paulo
9. Singapore Management University
10. Ericsson & Rise
11. Otto-von-Guericke University Magdeburg
12. KAIST, South Korea
13. University of Wisconsin--Madison & Microsoft Research

References

- https://miltos.allamanis.com/files/slides/2019ml4se.pdf
- https://miltos.allamanis.com/files/slides/2019fosdem.pdf
- http://xuwang.tech/paper/astnn_icse2019.pdf
- https://arxiv.org/pdf/1711.00740.pdf
- https://seim-conf.org/media/materials/2018/proceedings/SEIM-2018_Short_Papers.pdf#page=48
- https://arxiv.org/pdf/1906.08094.pdf
- https://arxiv.org/pdf/1904.12787.pdf
- https://arxiv.org/pdf/2004.03519v1.pdf
- https://github.com/rusty1s/pytorch_geometric
- https://github.com/zhangj111/astnn
- https://github.com/JetBrains-Research/astminer/
- https://github.com/c2nes/javalang
- https://github.com/tree-sitter/py-tree-sitter/
- https://github.com/rusty1s/pytorch_geometric
- https://github.com/dmlc/dgl
- https://github.com/dmlc/dgl/tree/master/examples/pytorch/tree_lstm
- https://github.com/microsoft/tf2-gnn/
- https://github.com/deepmind/graph_nets
- https://colab.research.google.com/github/deepmind/deepmind_research/blob/master/graph_matching_networks/graph_matching_networks.ipynb1

End

Siamese RNN: Results

- Embedding Dimension: *384*
- LSTM Units: *384*
- One FC Layer of *512*
- Trained for 25 epoches, with ADAM
- With Code2Vec:
  Acc: 0.86 | Recall: 0.83 | Precision: 0.85
- Without Code2Vec:
  Acc: 0.83 | Recall: 0.81 | Precision: 0.83

CodeRep

Input:

```
public class test{

  int a = 1 ;

  int b = 0.1; //error

}
```

Output:

2

Task 2: CodeRep (if time permits)

- CodeRep Challenge: https://github.com/KTH/codrep-2019/ (2020 version will be released soon)

- Description: Given a code file, find the line number in which the bug appear.

- Translated to a graph problem of **Node Selection or Classification.**

Model	Test Accuracy
GraphConv - TopKPooling + Code2vec Embedding	0.56
Siamese + GraphConv + Code2Vec : Correct	0.57
Siamese RNN without Code2Vec	0.83
GraphConv +Code2Vec	0.84
GraphConv + Ast merged by strucute + FastText	0.85
Siamese RNN with Code2Vec	0.86
Siamese + GraphConv + FastText	0.89