Introduction

SCIENCE AND TECHNOLOGY

Output is class label,Like function

Source code classification. This task aims to classify code fragments by their functionalities, which is usef
for program understanding and maintenance

- 2 Aspects
  - StructTexVec
  - 6 downsteam tasks
    - *Code Comment Generation*
    - *Code Authorship Identification*
    - *Code Clone Detection*
    - Source Code Classification
    - Logging Statement Prediction
    - Software Defect Prediction

Evaluation

Table 3 Evaluation results on the test set of six downstream tasks. The second last row shows the percentage of the best result produced by each approach on 22 datasets and the last row is the weighted averaged percentage of best results on six downstream tasks (i.e., each task's contribution to the percentage is weighted by its number of datasets)

Downstream tasks	Evaluation metrics	Dataset	None	Non-contextual embeddings					Contextual embeddings	
				Word2vec	GloVe	fastText	code2vec	StrucTexVec	CodeBERT	CuBERT
Code comment generation	BLEU	GitHub	14.9	15.4	15.9	14.6	15.3	16.0	**16.7**	16.1
Code authorship identification	Accuracy	Google Code Jam	87.5	80.2	87.5	77.1	85.4	86.5	87.0	**89.1**
Code clone detection	F1	BCB	92.7	**93.8**	**93.8**	**93.8**	93.5	93.6	93.5	93.6
		OJClone	85.1	86.8	81.4	78.0	**89.7**	88.1	85.9	81.6
		Avg.	88.9	90.3	87.6	85.9	**91.6**	90.9	89.7	87.6
Source code classification	Accuracy	OJ dataset	88.5	87.0	89.2	77.7	**91.2**	89.1	79.8	75.8
Logging statement prediction	Balance Accuracy	Airavata	**95.6**	94.3	94.2	93.1	94.8	94.5	**93.8**	93.4
		Camel	76.6	77.8	77.5	76.4	77.4	**79.2**	77.1	75.0
		CloudStack	85.9	86.0	85.5	84.7	86.9	**87.3**	86.0	86.7
		Directory-Server	82.9	84.1	85.6	84.7	84.0	86.6	**88.0**	81.9
		Hadoop	76.7	73.6	71.5	71.7	72.3	71.0	75.4	**77.6**
		Avg.	83.6	83.2	82.8	82.1	83.1	83.7	**84.1**	82.9
Software defect prediction	F1	Ant 1.5–>1.6	28.0	35.5	36.0	32.9	47.6	34.2	36.4	**54.8**
		Ant 1.6–>1.7	33.1	44.9	45.1	39.6	48.4	43.4	51.9	**52.9**
		Camel 1.2–>1.4	23.3	43.3	45.5	43.8	43.2	**46.8**	45.6	44.2
		Camel 1.4–>1.6	26.3	47.0	49.8	46.0	50.0	50.2	**51.2**	50.3
		jEdit 3.2–>4.0	32.7	52.0	56.2	55.9	56.6	59.5	**61.5**	59.4
		jEdit 4.0–>4.1	40.6	60.5	60.1	59.7	57.9	**64.7**	62.4	59.9

models that do not use pre-trained code embeddings (i.e., None).

三篇论文

SCIENCE AND TECHNOLOGY

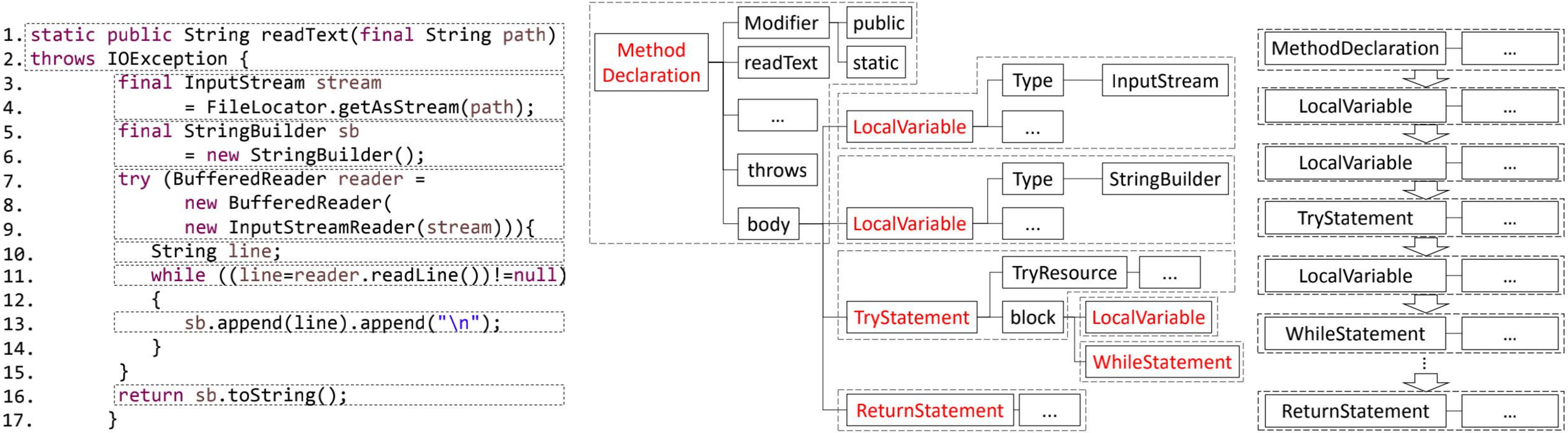
《A Novel Neural Source Code Representation based on Abstract Syntax Tree》	2019	ICSE
《DeepBugs:A Learning Approach to Name-Based Bug Detection》	2018	OOPSLA
《VulDeePecker: A Deep Learning-Based System for Vulnerability Detection》	2018	NSDD

GRADUATION DEFENSE

A Novel Neural Source Code Representation based on Abstract Syntax Tree

2019 ICSE

RNN GRU



(a) Code fragment and statements (b) AST and statement trees (c) Statement naturalness

Fig. 1. An example of AST Statement nodes (marked in red)

A Novel Neural Source Code Representation based on Abstract Syntax Tree

2019 ICSE

ASTNN

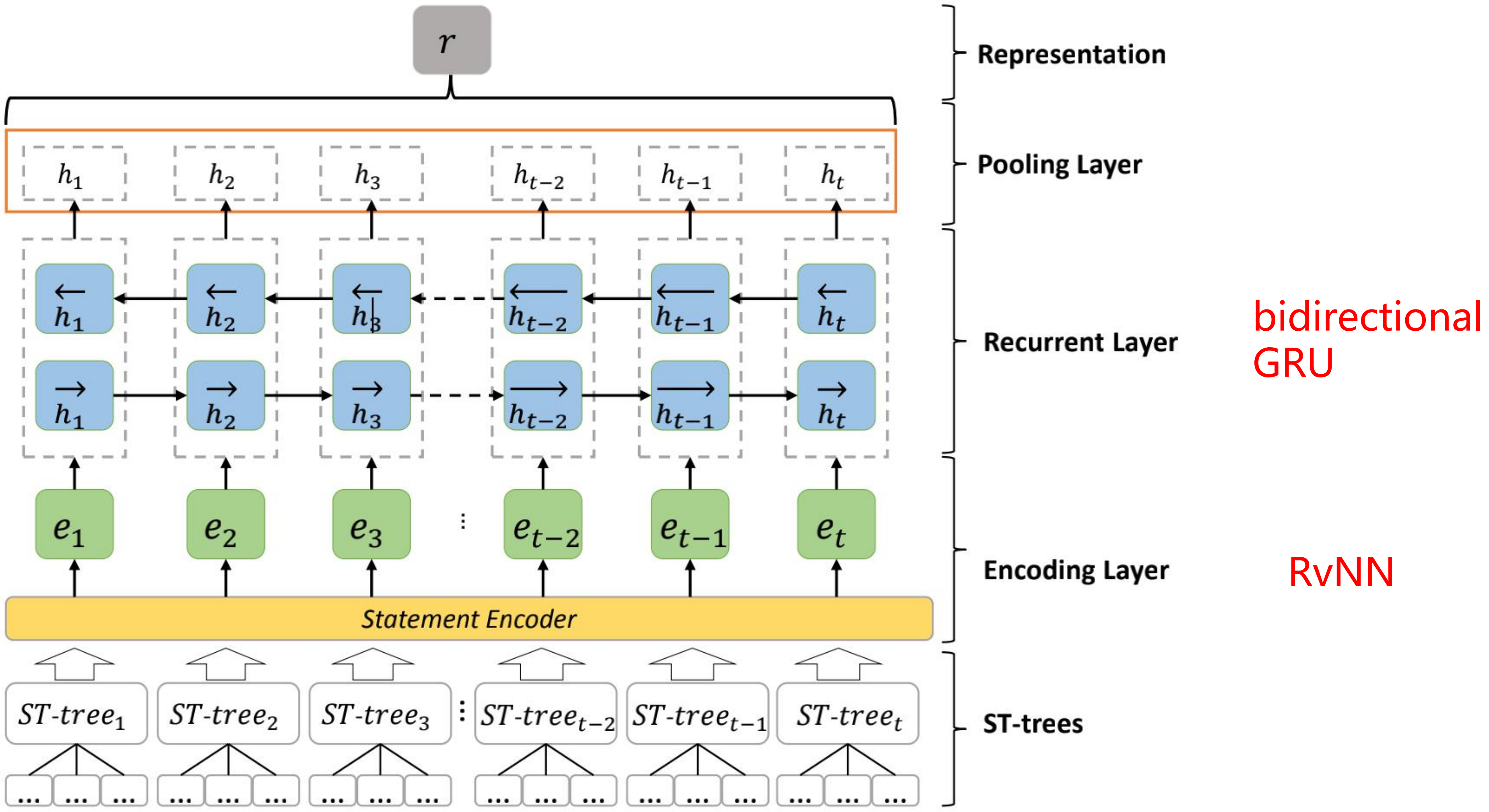


Fig. 2. The architecture of AST-based Neural Network

A Novel Neural Source Code Representation based on Abstract Syntax Tree

2019 ICSE

TABLE II
COMPARED APPROACHES FOR CODE CLASSIFICATION

Groups	Methods	Test Accuracy(%)
SVMs	SVM+TF-IDF	79.4
	SVM+N-gram	84.7
	SVM+LDA	47.9
Neural models	TextCNN	88.7
	LSTM	88.0
	TBCNN	94.0
	LSCNN	90.9
	PDG+HOPE	4.2
	PDG+GGNN	79.6
Our approach	ASTNN	**98.2**

TABLE IV
CODE CLONE DETECTION MODELS ON OJCLONE

Metric	RAE+	CDLH	PDG+HOPE	PDG+GGNN	ASTNN
P	52.5	47	76.2	77.3	**98.9**
R	68.3	73	7.0	43.6	**92.7**
F1	59.4	57	12.9	55.8	**95.5**

DeepBugs:A Learning Approach to Name-Based Bug Detection

2018 OOPLSA

Table 1. Examples of name-related bugs detected by DeepBugs.

ID	Buggy code	Description

1	`browserSingleton.startPoller(100,`
	`  function(delay, fn) {`
	`    setTimeout(delay,fn);`
	`});`
		The `setTimeout` function expects two arguments: a callback function and the number of milliseconds after which to invoke the callback. The code accidentally passes these arguments in the inverse order.

2	`for (j = 0; j < param.replace; j++) {`
	`  if (param.replace[j].from === paramVal)`
	`    paramVal = param.replace[j].to;`
	`}`
		The header of the for-loop compares the index variable `j` to the array `param.replace`. Instead, the code should compare `j` to `param.replace.length`.

3	`for(var i = 0; i<this.NR_OF_MULTIDELAYS; i++){`
	`  // Invert the signal of every even multiDelay`
	`  outputSamples = mixSampleBuffers(outputSamples,`
	`      this.multiDelays[i].process(filteredSamples),`
	`      2%i==0, this.NR_OF_MULTIDELAYS);`
	`    /*^^^^^^^*/`
	`}`
		The highlighted expression `2%i==0` is supposed to alternate between `true` and `false` while traversing the loop. However, the code accidentally swapped the operands and should instead be `i%2==0`.

len vs length

length vs count

DeepBugs:A Learning Approach to Name-Based Bug Detection

2018 OOPLSA

DeepBug

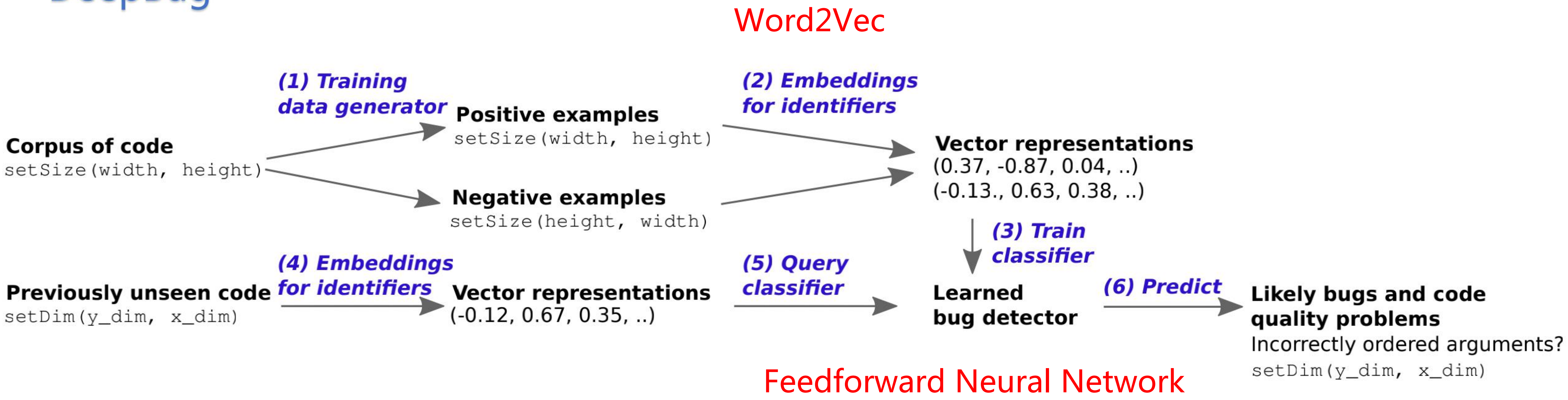


Fig. 1. Overview of our approach.

DeepBugs:A Learning Approach to Name-Based Bug Detection

2018 OOPLSA

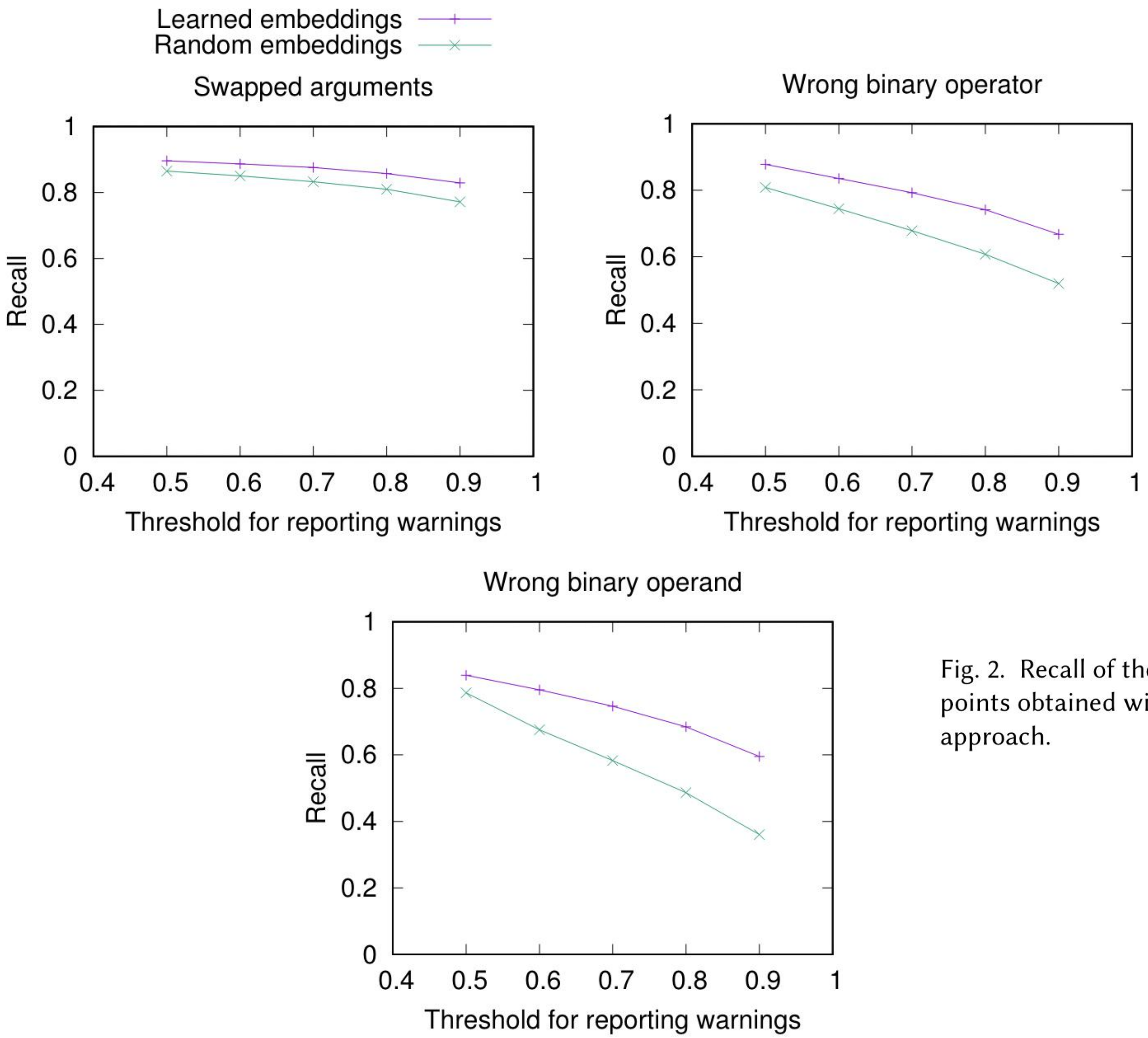


Fig. 2. Recall of the bug detectors with different thresholds t for reporting warnings. Each plot contains data points obtained with $t \in \{0.5, ..., 0.9\}$. The data labeled "Learned embedding" corresponds to the DeepBugs approach.

VulDeePecker:
A Deep Learning-Based System for Vulnerability Detection

2018 NSDD

2 Drawbacks

Intense Manual Labor

Existing solutions for vulnerability detection rely on human experts to define features.

High False Negative Rates

Existing solutions often miss many vulnerabilities or incur high false negative rates.

VulDeePecker:
A Deep Learning-Based System for Vulnerability Detection

2018 NSDD

Guiding Principles

A. How to represent software programs?

B. What is an appropriate granularity?

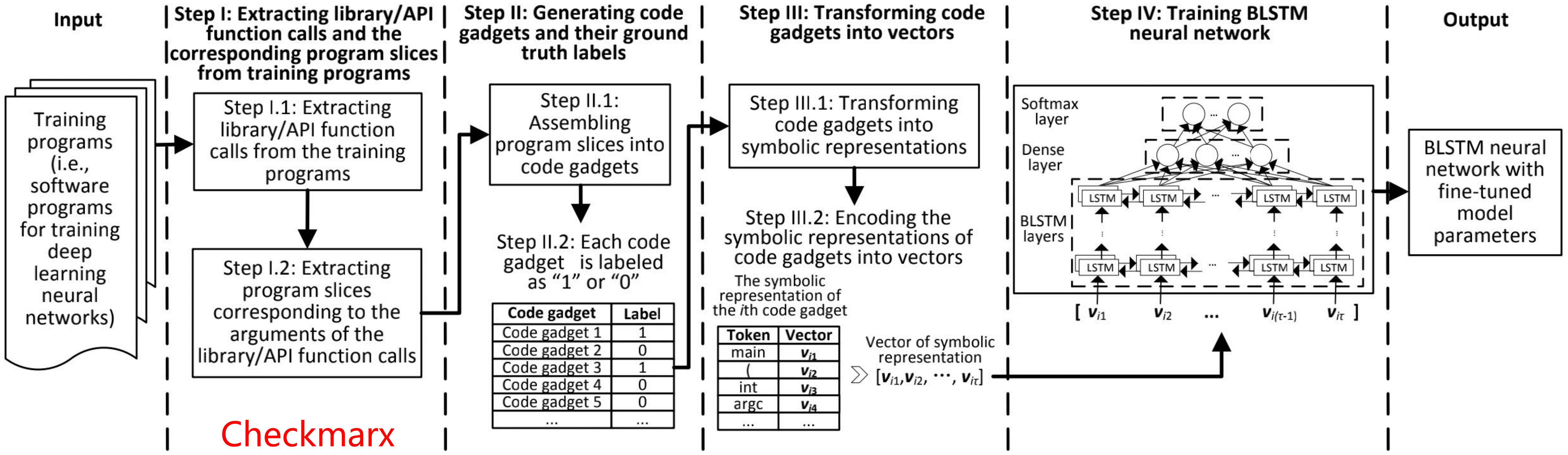
C. How to select neural networks?

Code Gadget

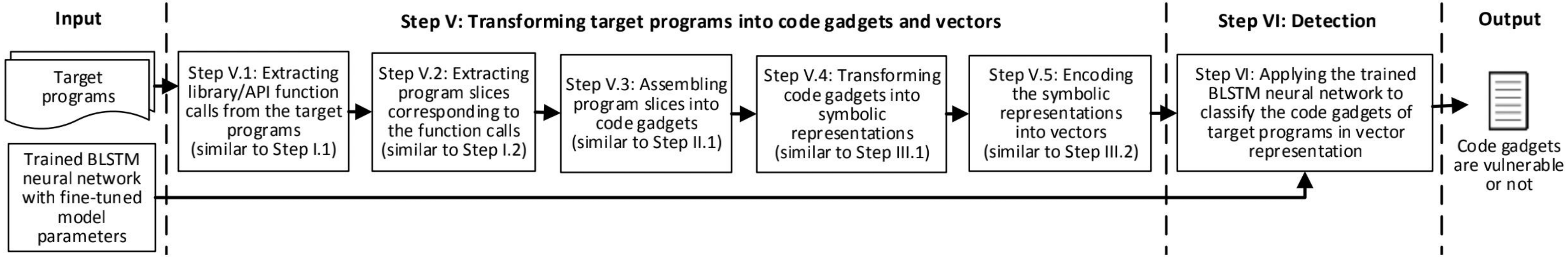
a number of program statements (i.e., lines of code)
which are semantically related to each other in terms of data dependency or control dependency.

BLSTM

VulDeePecker: A Deep Learning-Based System for Vulnerability Detection



(a) Learning phase



(b) Detection phase