三篇论文

SCIENCE AND TECHNOLOGY

《Are Mutation Scores Correlated with Real Fault Detection?》	2018	ICSE
《Automated Vulnerability Detection in Source Code Using Deep Representation Learning》	2018	ICMLA
《Big Code != Big Vocabulary- Open-Vocabulary Models for Source Code》	2020	ICSE

GRADUATION DEFENSE

Are Mutation Scores Correlated with Real Fault Detection?

What is the relation between mutants and real faults?

2018 ICSE

Table 1: Summary of studies investigating the relationship between mutants and real faults.

Author(s) [Reference]	Year	Largest Subject	Language	Considered Test Size	No Faults	Summary of Scientific Findings
Daran & Thévenod-Fosse [10]	96	1,000	C	×	12	Mutants result in failures and program data states that are **similar** to those produced by real faults.
Frankl *et al.* [14]	'97	78	Fortran, Pascal	✓	9	**Fault detection** probability is increasing at higher mutation score levels. The increase is non-linear.
Andrews *et al.* [3]	'05	5,000	C	✓	38	Mutants detection ratios are **representative** of fault detection ratios
Andrews *et al.* [4]	'06	5,000	C	×	38	Mutants detection ratios are **representative** of fault detection ratios
Papadakis & Malevris [37]	'10	5,000	C	×	38	1^{st} order mutation has higher **fault detection** than 2^{nd} order and mutant sampling. There are significantly less equivalent 2nd order mutants than 1^{st} order ones.
Namin & Kakarla [28]	'11	5,000	C	✓	38	There is a weak **correlation** between mutant detection ratios and real fault detection ratios
Just *et al.* [24]	'14	96,000	Java	×	357	There is a strong **correlation** between mutant detection ratios and real fault detection ratios
Shin *et al.* [40]	'17	96,000	Java	×	352	Distinguishing mutation adequacy criterion has higher **fault detection** probability than strong mutation adequacy criterion
Ramler *et al.* [38]	'17	60,000	Java	×	2	Mutation testing helps improving the test suites of a safety-critical industrial software system by increasing their **fault detection** potential.
Chekam *et al.* [8]	'17	83,100	C	✓	61	Mutation testing provides valuable guidance for improving test suites and revealing real faults. There is a strong connection between mutation score increase and **fault detection** at higher score levels.
This paper	'18	96,000	C & Java	✓	420	There is a weak **correlation** between mutation score and real fault detection. Despite the weak correlations, fault detection is significantly improved at the highest score levels.

Are Mutation Scores Correlated with Real Fault Detection?

Randoop、EvoSuite

2 0 1 8 I C S E

1. Create a large number of test suites

2. Measure the criteria score such as coverage or mutation score (if test suites are size controlled) of the test suites or their size (if test suites are score controlled).

3. Measure the fault detection capability of the test suites, by measuring either number or ratio of detected faults.

4. Determine the test effectiveness based on one of the two following methods:

   1) fault detection ratios and criteria scores

   2) fault detection ratios at predefined score

Are Mutation Scores Correlated with Real Fault Detection?

2018 ICSE

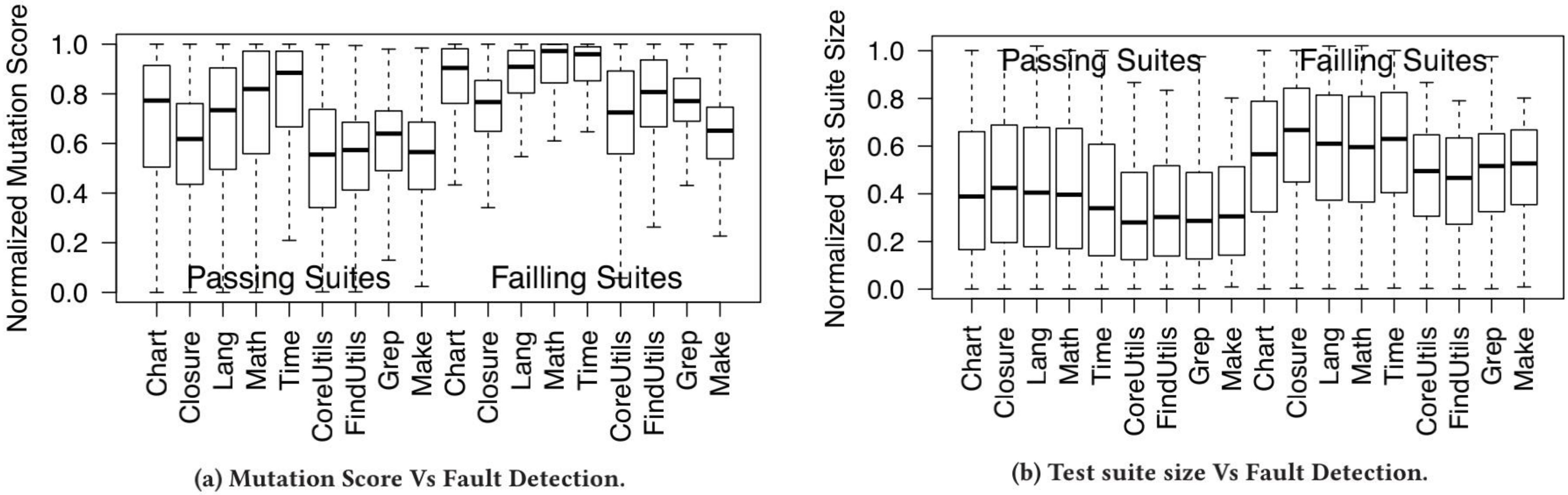


Figure 1: Mutation Score and Test suite size of the Passing and Failing Test Suites. Failing Test Suites have higher mutation scores and suite sizes than the Passing ones.

Are Mutation Scores Correlated with Real Fault Detection?

2018 ICSE

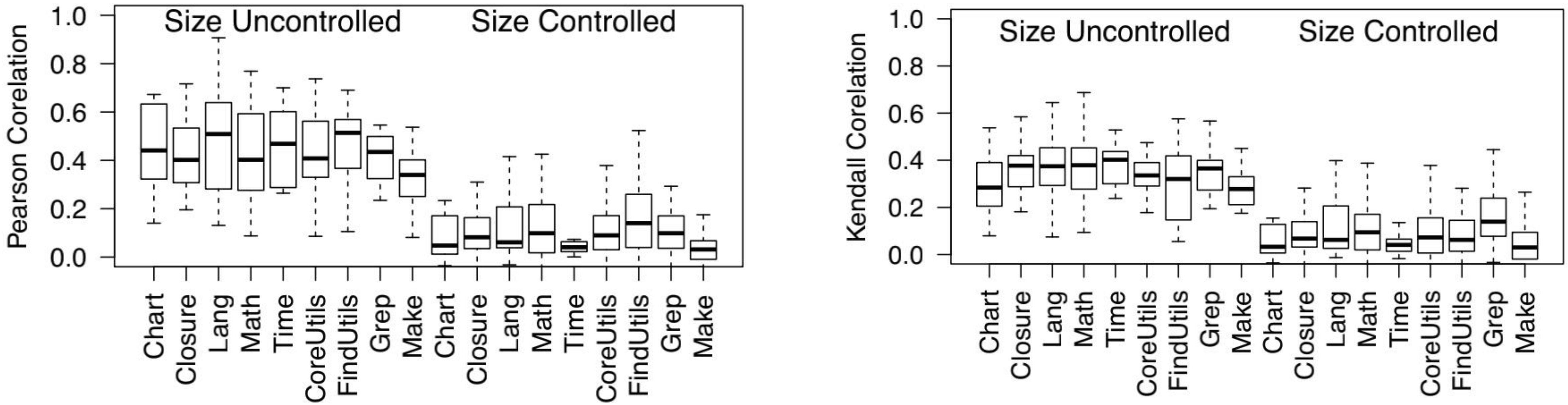
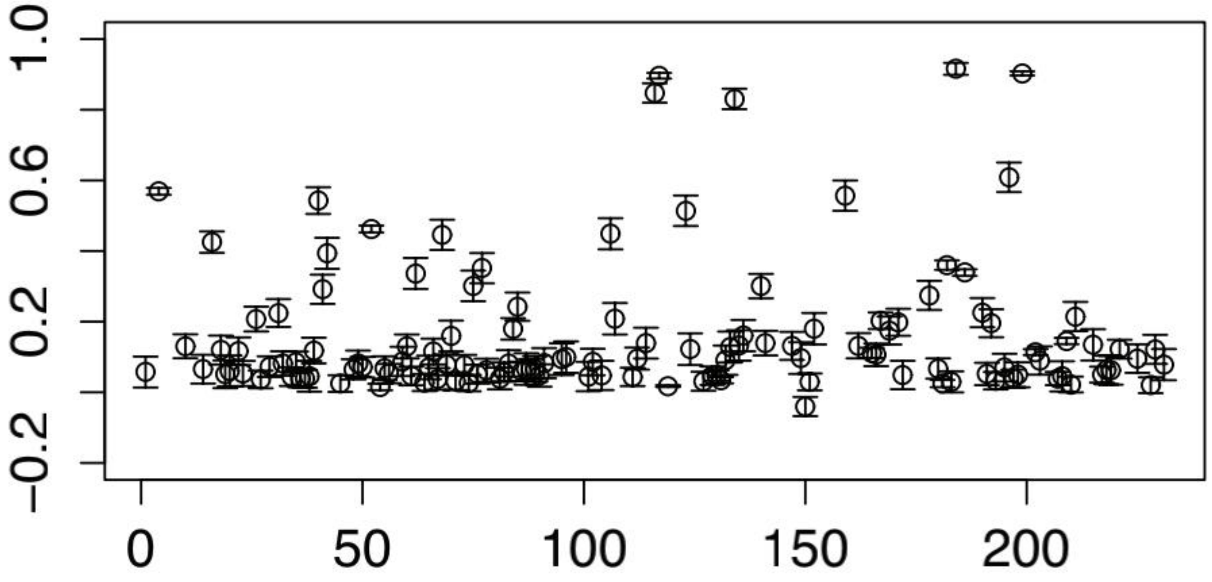


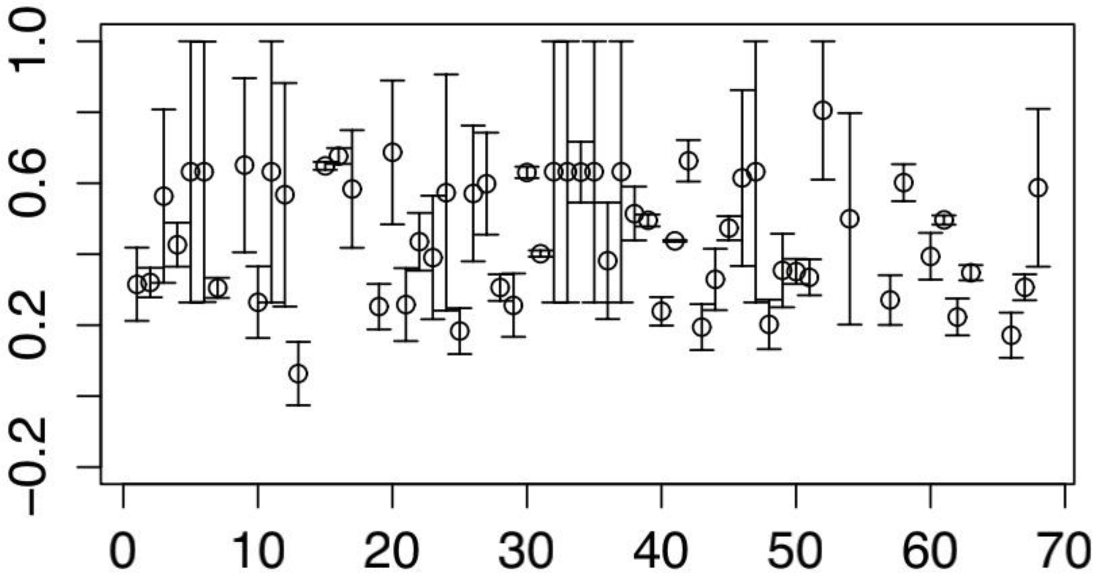
Figure 3: Correlation between mutation score and fault detection. Correlations are relatively strong when test suite size is uncontrolled but drop significantly when test suite size is controlled.

Are Mutation Scores Correlated with Real Fault Detection?

2018 ICSE



(a) Defects4J



(b) CoreBench

Figure 4: Improvement on the fault detection probabilities (intervals) with test suite size controlled. The values represent the difference on the fault detection probabilities between the test suites with the highest mutation score and randomly selected ones (top 10% of test suites Vs randomly selected). We observe that mutants lead to significant fault detection improvements.

Automated Vulnerability Detection in Source Code Using Deep Representation Learning

2018 ICMLA

Compiled a vast ***dataset*** of millions of open-source functions and labeled it with carefully-selected findings from three different static analyzers that indicate potential exploits.

Developed a fast and scalable ***vulnerability detection tool*** based on deep feature representation learning that directly interprets lexed source code.

Automated Vulnerability Detection in Source Code Using Deep Representation Learning

2018 ICMLA

	labeled code snippet	not labeled code	not labeled code
	SATE IV	GitHub	Debian
Total	121,353	9,706,269	3,046,758
Passing curation	11,896	782,493	491,873
‘Not vulnerable’	6,503 (55%)	730,160 (93%)	461,795 (94%)
‘Vulnerable’	5,393 (45%)	52,333 (7%)	30,078 (6%)

TABLE I: Total number of functions obtained from each data source, the number of valid functions remaining after removing duplicates and applying cuts, and the number of functions without and with detected vulnerabilities.

Automated Vulnerability Detection in Source Code Using Deep Representation Learning

2018 ICMLA

Lable

- Static analysis　　Clang, Cppcheck , Flawfinder
- Dynamic analysis —— take too much effort
- Commit-message/bug-report tagging—— challenging, providing low-quality labels

Automated Vulnerability Detection in Source Code Using Deep Representation Learning

2018 ICMLA

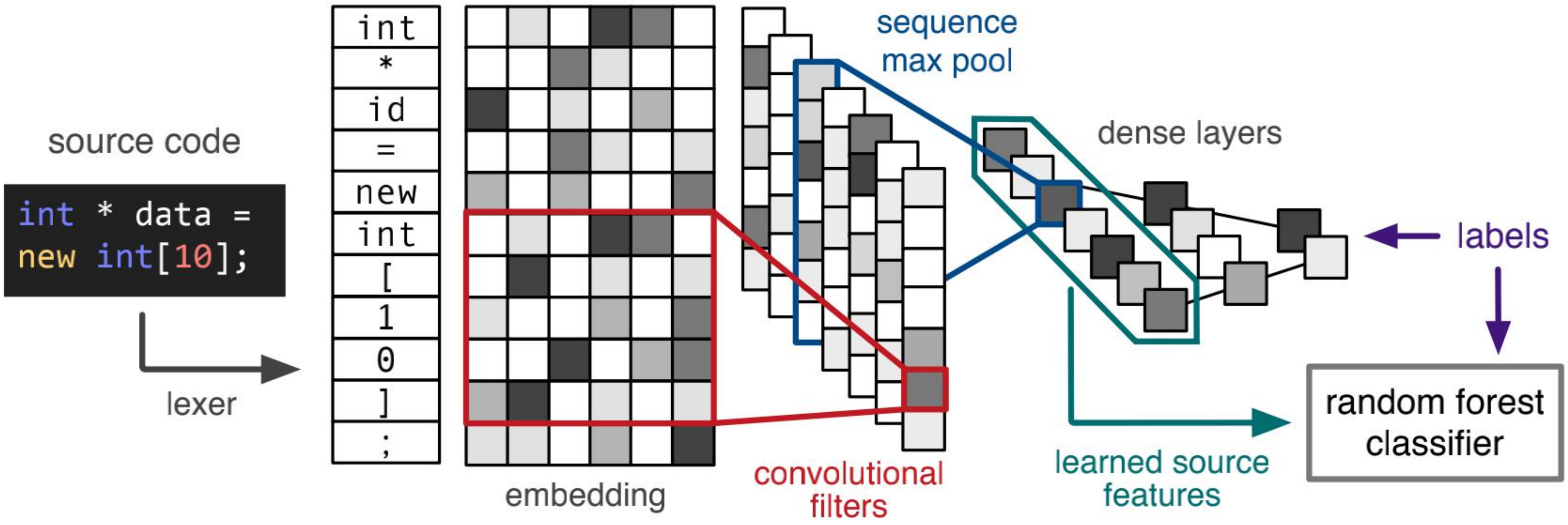


Fig. 1: Illustration of our convolutional neural representation-learning approach to source code classification. Input source code is lexed into a token sequence of variable length ℓ, embedded into a $\ell \times k$ representation, filtered by n convolutions of size $m \times k$, and maxpooled along the sequence length to a feature vector of fixed size n. The embedding and convolutional filters are learned by weighted cross entropy loss from fully-connected classification layers. The learned n-dimensional feature vector is used as input to a random forest classifier, which improves performance compared to the neural network classifier alone.

Automated Vulnerability Detection in Source Code Using Deep Representation Learning

2018 ICMLA

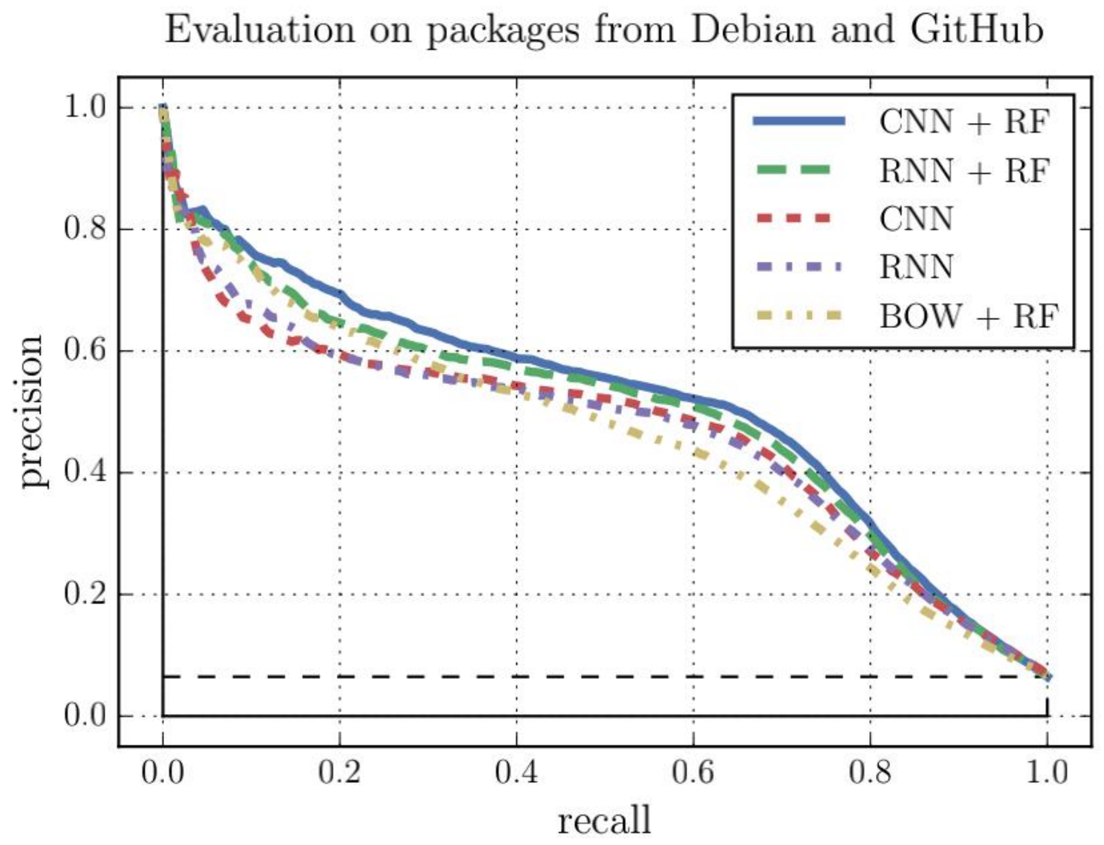


Fig. 2: Precision versus recall of different ML approaches using our lexer representation on Debian and Github test data. Vulnerable functions make up 6.5% of the test data.

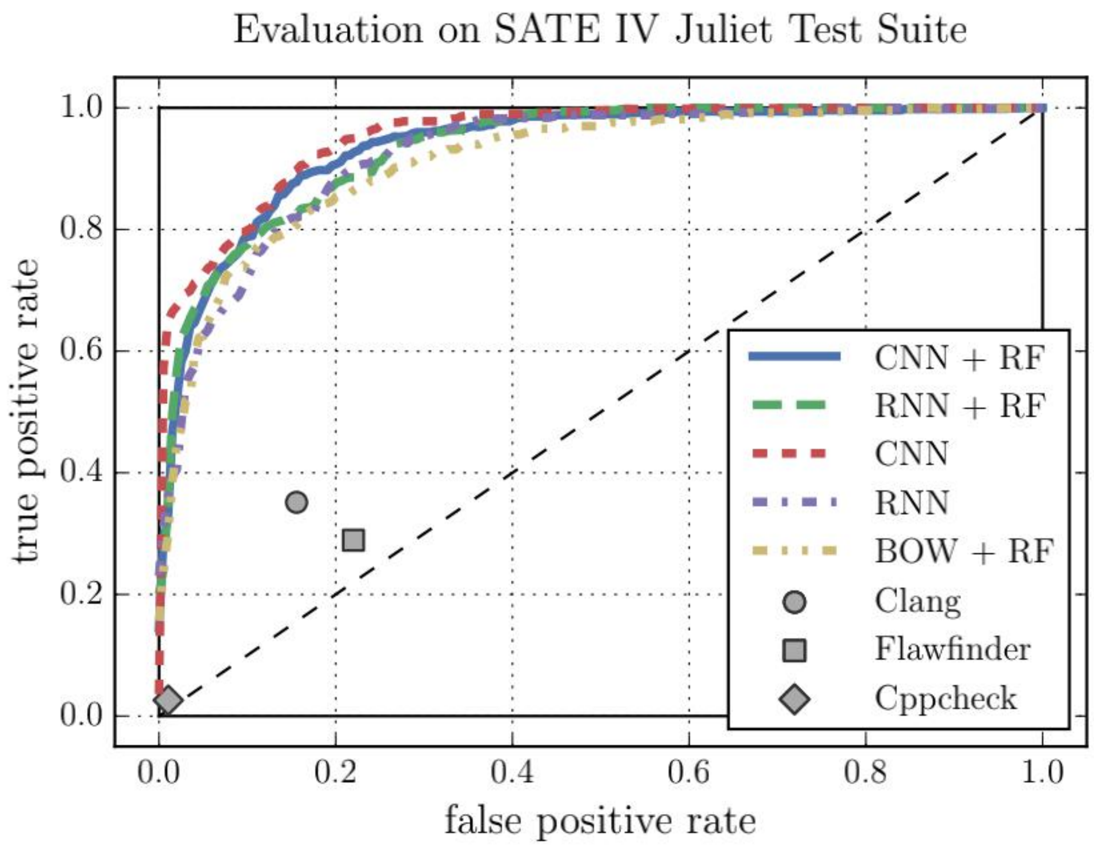


Fig. 3: SATE IV test data ROC, with true vulnerability labels, compared to the three static analyzers we considered. Vulnerable functions make up 43% of the test data.

Big Code != Big Vocabulary- Open-Vocabulary Models for Source Code

2020 ICSE

OOV (Out of Vocabulary)

If identifiers were not observed in the training set, many classes of models cannot predict them, which is known as the out-of-vocabulary (OOV) problem.

Big Code != Big Vocabulary- Open-Vocabulary Models for Source Code

2020 ICSE

LM (Language Model)

A language model (LM) estimates the probabilities of sequences of words based on a training corpus.

NLM (Natural Language Model)

- *The current state-of-the-art in NLP.*
- *NLMs represent words in a continuous vector space, allowing the model to infer relationships between words,even if they do not appear in a specific context during training.*

Difficulties with Large Vocabularies

- Scalability
- OOV
- Rare Words

Big Code != Big Vocabulary- Open-Vocabulary Models for Source Code

Modeling Vocabulary

2020 ICSE

1 **Filtering the vocabulary**

English、White space、Comments、String

Full token vocabularies range in the millions, and hence do not scale. OOV and frequency issues are extremely important.

2 **Word Splitting**

Word splitting is effective, but the vocabulary is still large (a million words). OOV and frequency issues are still important.

Big Code != Big Vocabulary- Open-Vocabulary Models for Source Code

Modeling Vocabulary

2020 ICSE

3 **Subword splitting**

Number、Spiral、Other approach

While these strategies are effective, they do not go far enough; vocabulary stays in the hundreds of thousands range. There are still OOV issues for unseen data; most words are uncommon.

4 **Subword splitting with BPE**

BPE is an algorithm originally designed for data compression, in which bytes that are not used in the data replace the most frequently occurring byte pairs or sequences .

(S1, S2)
S1S2

BPE shrinks source code vocabulary very effectively. Moreover, most of the vocabulary is frequent, improving embeddings.

Big Code != Big Vocabulary- Open-Vocabulary Models for Source Code

020 ICSE

Table 2: Performance of the various models (bold: best, underlined: second best).

MODEL	Java							Java Identifiers			C				Python			
	Static		Dynamic		Maintenance		Bugs	Dynamic			Static		Dynamic		Static		Dynamic	
	Ent	MRR	Ent	MRR	Ent	MRR	% Ent ↓	R@1	R@10	MRR	Ent	MRR	Ent	MRR	Ent	MRR	Ent	MRR
Small Train																		
n-gram	6.25	53.16	5.54	56.21	5.30	58.32	1.81	17.24	34.66	22.26	6.51	55.20	4.14	57.34	5.30	43.63	4.81	47.39
Nested	-	-	3.65	66.66	2.94	71.43	-	37.46	56.85	43.87	-	-	3.61	62.25	-	-	4.05	54.02
Cache	-	-	3.43	69.09	3.32	70.23	-	40.13	59.52	46.57	-	-	2.19	75.09	-	-	3.22	62.27
Nested Cache	-	-	2.57	74.55	2.23	77.04	-	49.93	70.09	56.81	-	-	2.01	76.77	-	-	2.89	65.97
Closed NLM	**4.30**	62.28	3.07	71.01	-	-	1.81	30.96	49.93	37.20	4.51	60.45	3.20	72.66	3.96	**81.73**	3.34	84.02
Heuristic NLM	4.46	53.95	3.34	64.05	-	-	1.04	39.54	58.37	45.28	4.82	52.30	3.67	61.43	4.29	65.42	3.56	71.35
BPE NLM (512)	4.77	63.75	2.54	77.02	**1.60**	**78.69**	3.26	45.49	67.37	52.66	4.32	62.78	1.71	76.92	3.91	81.66	2.72	86.28
BPE NLM (512) + cache	-	-	-	77.42	-	-	-	50.49	68.16	56.30	-	-	-	-	-	-	-	-
BPE NLM (2048)	4.77	**64.27**	**2.08**	77.30	-	-	**3.60**	48.22	69.79	55.37	**4.22**	**64.50**	**1.59**	**78.27**	**3.66**	81.71	**2.69**	**86.67**
BPE NLM (2048) + cache	-	-	-	**78.29**	-	-	-	**52.44**	**70.12**	**58.30**	-	-	-	-	-	-	-	-
Large Train																		
Nested Cache	-	-	2.49	75.02	2.17	77.38	-	52.20	72.37	59.09	-	-	1.67	**84.33**	-	-	**1.45**	71.22
BPE NLM (512)	3.15	70.84	1.72	79.94	**1.04**	**81.16**	4.92	51.41	74.13	59.03	3.11	70.94	1.56	77.59	3.04	84.31	2.14	87.06
BPE NLM (512) + cache	-	-	-	80.29	-	-	-	55.68	**74.30**	61.94	-	-	-	-	-	-	-	-
BPE NLM (2048)	**2.40**	**75.81**	**1.23**	82.41	-	-	**5.98**	57.54	72.18	62.91	**2.38**	**80.17**	**1.36**	83.24	**2.09**	**86.17**	1.90	**87.59**
BPE NLM (2048) + cache	-	-	-	**83.27**	-	-	-	**60.74**	73.76	**65.49**	-	-	-	-	-	-	-	-

Code Completion Scenarios

RQ1. Performance of Models

南京航空航天大学

Nanjing University of Aeronautics and Astronautics

2022

THANK YOU

THNAK YOU

GRADUATION DEFENSE