该配置下,模型效果如图 8 所示,横轴为迭代次数,纵轴为准确率.

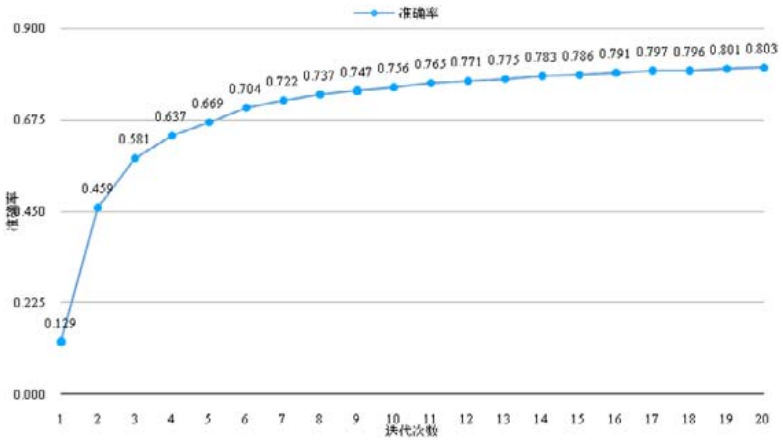


Fig.8 Accuracy of DL model (*HIDDEN_SIZE*=250,*NUM_LAYER*=2,*LR*=0.002)

图 8 深度学习模型准确率(*HIDDEN_SIZE*=250,*NUM_LAYER*=2,*LR*=0.002)

经过 20 次迭代后,模型准确率达到 80.3%.该参数组合在验证集上效果最优,因此最终用于缺陷预测的模型参数为:*HIDDEN_SIZE*=250,*NUM_LAYER*=2,*LR*=0.002,*NUM_EPOCH*=20.

4.3 代码缺陷检测实验

本节进行基于深度学习模型的 API 误用相关代码缺陷检测实验.进行该部分实验前,对实验中的测试标准进行定义.

- 定义 TP 为检测文件 API 误用缺陷位置正确的不重复缺陷报告数.
- 定义 FP 为检测文件 API 误用缺陷位置错误的不重复缺陷报告数.
- 定义 FN 为未进行报告的缺陷报告数.

1. 查准率(precision)定义为

$$P=\frac{TP}{TP+FP} \tag{1}$$

2. 召回率(recall)定义为

$$R=\frac{TP}{TP+FN} \tag{2}$$

3. 查准率和召回率的调和均值($F1$)定义为

$$F1=\frac{2TP}{2TP+FP+FN} \tag{3}$$

查准率和召回率往往体现的是实验结果的一个侧面,比如召回率较高的模型在查准率上不一定很高.因此在本文的实验中,采用二者的调和均值 $F1$ 进行 API 误用缺陷检测的实验评价.

4.3.1 实验数据

本次实验中,根据相关文献中 Java Cryptography API 误用的真实数据[5],选用了从 SourceForge 和 GitHub 上的优质项目(大于 10 星)中整理的关于 Java Cryptography APIs 的误用代码片段 8 个,其中包含 API 误用 14 个,将这 14 个 API 误用作为计算模型查准率、召回率以及 $F1$ 值的测试集.实验所用的测试集整理如表 1 所示,表格中标明了每个测试用例的代码来源、API 误用方法来源、误用引入缺陷的具体 API 以及 API 误用的说明,所有测试用例源码可通过代码来源在托管平台上下载.

Table 1 Information of test case

表 1 测试用例信息

编号	代码来源	API 误用方法	API 误用	API 误用说明
1	https://github.com/alibaba/druid/commit/e10f2849d046265bf17360ab4aa9eb60fd3ab8de	Decrypt (PublicKey,String)	javax.crypto.Cipher.init (int,java.security.Key)	An instance of Cipher is used twice (the init() method is called again), which is an invalid operation
2	https://github.com/alibaba/druid/commit/e10f2849d046265bf17360ab4aa9eb60fd3ab8de	Encrypt (byte[],String)	javax.crypto.Cipher.init (int,java.security.PrivateKey)	A call to Cipher.init() may throw an InvalidKeyException
3	https://github.com/android-rcs/rcsjta/commit/04d84799daa51ed7cc0ad270f0eea51ffaf7a53a#diff-bf160ca00204f2ae4c100aabe57a1dfd	getContributionId (String)	java.lang.String.getBytes()	Exports bytes for Mac.doFinal() without specifying the encoding
4	https://github.com/android-rcs/rcsjta/commit/04d84799daa51ed7cc0ad270f0eea51ffaf7a53a#diff-bf160ca00204f2ae4c100aabe57a1dfd	getContributionId (String)	javax.crypto.Mac.doFinal(byte[])	Exports bytes for Mac.doFinal() without specifying the encoding
5	http://sourceforge.net/p/adempiere/svn/1312/tree/trunk/looks/src/org/compiere/util/Secure.java?diff=5139a2ef34309d2ec1827857:1311	encrypt(String)	java.lang.String.getBytes()	A string is converted to bytes without specifying an explicit encoding. The bytes are then passed to Cipher.doFinal()
6	http://sourceforge.net/p/adempiere/svn/1312/tree/trunk/looks/src/org/compiere/util/Secure.java?diff=5139a2ef34309d2ec1827857:1311	Encrypt (String)	javax.crypto.Cipher.doFinal(byte[])	A string is converted to bytes without specifying an explicit encoding. The bytes are then passed to Cipher.doFinal()
7	http://sourceforge.net/p/adempiere/svn/1312/tree/trunk/looks/src/org/compiere/util/Secure.java?diff=5139a2ef34309d2ec1827857:1311	Decrypt (String)	java.lang.String.new(byte[])	An encrypted message is decrypted and then converted back to a string, without specifying an explicit encoding. The fix specifies the encoding “UTF-8”
8	http://sourceforge.net/p/battleforge/code/878/tree/trunk/de.battleforge/src/java/de/battleforge/util/BFProperties.java?diff=50ee84dee88f3d24b3d975fe:877	setProperty (BFProps, String, boolean)	java.lang.String.new(byte[])	Encoded data is converted into a String for storing, without explicitly specifying an encoding. The fix introduces base64 encoding
9	http://sourceforge.net/p/battleforge/code/878/tree/trunk/de.battleforge/src/java/de/battleforge/util/BFProperties.java?diff=50ee84dee88f3d24b3d975fe:877	setProperty (BFProps,String, boolean)	java.lang.String.getBytes()	Text is converted to bytes for encoding without an explicit encoding. The bytes are then passed to Cipher.doFinal()
10	http://sourceforge.net/p/battleforge/code/878/tree/trunk/de.battleforge/src/java/de/battleforge/util/BFProperties.java?diff=50ee84dee88f3d24b3d975fe:877	setProperty (BFProps,String, boolean)	javax.crypto.Cipher.doFinal(byte[])	Text is converted to bytes for encoding without an explicit encoding. The bytes are then passed to Cipher.doFinal()
11	http://sourceforge.net/p/battleforge/code/878/tree/trunk/de.battleforge/src/java/de/battleforge/util/BFProperties.java?diff=50ee84dee88f3d24b3d975fe:877	getProperty (BFProps)	java.lang.String.getBytes()	Encoded data is retrieved from a string (from storage) without explicitly specifying an encoding. The bytes are then passed to Cipher.doFinal()
12	http://sourceforge.net/p/battleforge/code/878/tree/trunk/de.battleforge/src/java/de/battleforge/util/BFProperties.java?diff=50ee84dee88f3d24b3d975fe:877	getProperty (BFProps)	javax.crypto.Cipher.doFinal(byte[])	Encoded data is retrieved from a string (from storage) without explicitly specifying an encoding. The bytes are then passed to Cipher.doFinal()

Table 1 Information of test case (Continued)

表 1 测试用例信息(续)

编号	代码来源	API 误用方法	API 误用	API 误用说明
13	http://sourceforge.net/p/battleforge/code/878/tree/trunk/de.battleforge/src/java/de/battleforge/util/BFProperties.java?diff=50ee84dee88f3d24b3d975fe:877	getProperty (BFProps)	java.lang.String.new(byte[])	Decoded data is converted to String without explicitly specifying an encoding
14	https://sourceforge.net/p/jmrtd/code/51/tree//passporthostapi/src/sos/mrtd/SecureMessagingWrapper.java?diff=5058d727fd48f84fd52d6740:50	readDO8E (DataInputStream, byte[])	EOS	DataOutputStream is left open

这些测试用例中包含 4 种类型的 API 误用(不互斥).

a. 使用了多余的 API 调用;

b. 使用了错误的 API 调用;

c. 遗漏了关键的 API 调用;

d. 忽略了对 API 调用中可能抛出的异常的处理.

其中,测试用例 1 使用了多余的 API 调用;测试用例 2 忽略了对 API 调用中可能抛出的异常的处理;测试用例 3、测试用例 5、测试用例 7~测试用例 9、测试用例 11、测试用例 13 使用了错误的 API 调用;测试用例 4、测试用例 6、测试用例 10、测试用例 12、测试用例 14 遗漏了关键的 API 调用.

4.3.2 实验设计

人工整理 API 误用测试用例,将每个 API 误用标注成标准化的格式“文件路径,方法,API 误用缺陷位置,误用 API”,用于计算查准率、召回率和 $F1$ 值.

在进行 API 误用缺陷检测时,DeepLSTM 将预测出 API 调用序列上某个位置上的 API 调用概率列表,为进行缺陷检测,定义 API 调用在 API 调用概率列表中可以接受的排名范围,在本文中称为可接受阈值(acceptable threshold).假设实验中的缺陷检测模型的可接受水平设置为 k,那么当 API 调用在 API 调用概率列表中的排名在 Top-1~Top-k 之间时(包括 Top-k),认为该 API 调用在本次 API 误用缺陷检测中处于 API 调用的可接受范围内;反之,认为该 API 调用属于 API 误用代码缺陷.

为验证模型的有效性,设置以下模型作为实验的对比模型.

(1) 基准模型(baseline).该模型假设 API 调用序列上的每个位置都可能是潜在的 API 误用,该假设覆盖到所有的 API 误用缺陷,召回率恒定为 1,在测试集上的查准率为 0.088 6,$F1$ 值为 0.163.

(2) N-gram 检测模型.Bugram[10]基于 N-gram 语言模型实现了对存在 API 误用的异常序列的检测.本文关注于对 API 调用序列上某一位置可能存在的 API 误用的检测,因此,本文实现 Bugram 应用的 N-gram 预测模型作为对比实验,使用与深度学习模型训练时相同的 API 调用序列,训练 N-gram 模型,基于前 N 个词元预测下一个词元.实验中取 N={3,4,5}分别对应 3-gram 模型、4-gram 模型和 5-gram 模型,并进行 Jelinek-Mercer 平滑处理[25].N-gram 检测模型应用 N-gram 预测模型对 API 调用序列上某个位置的 API 调用概率列表进行预测,同样的,应用可接受阈值(Top-k)进行缺陷检测.

在本实验中,比较不同模型的 API 误用缺陷检测效果,并探究可接受阈值的取值对代码缺陷检测模型的 $F1$ 值的影响.

4.3.3 实验结果分析

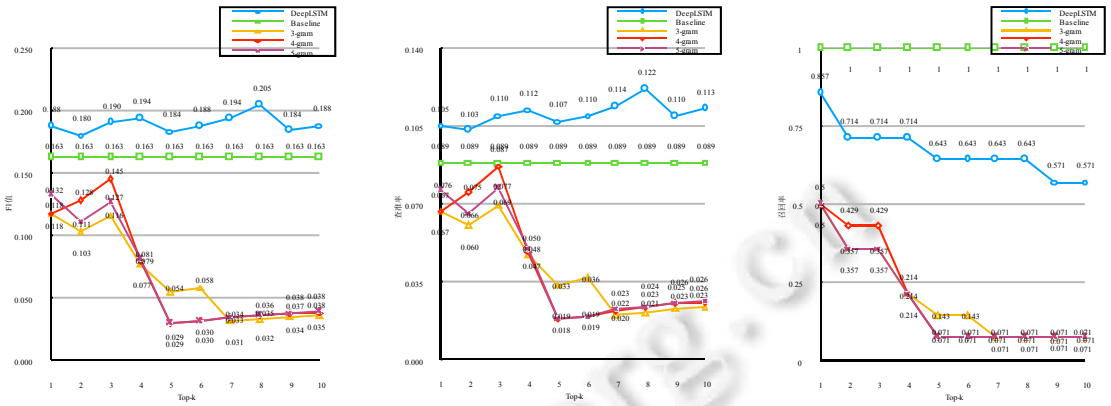
经测试,API 误用缺陷检测的实验结果如图 9 所示,图中折线表示不同的实验模型,横轴表示可接受阈值(top-k)的取值,图 9(a)是各模型的 $F1$ 值,图 9(b)是各模型的查准率,图 9(c)是各模型的召回率.

从结果中可以看出:

在一定范围内,DeepLSTM 的查准率均高于基准模型且召回率大于等于 50%,有一定的缺陷检测能力.本实

验中 4-gram 模型较 3-gram 模型和 5-gram 模型缺陷检测效果较好,但在本实验中,它们均不如基准模型(查准率最高约和基准持平).而相较而言,DeepLSTM 模型在本次实验所采用的模型中有一定的检测能力且效果最好.

当可接受阈值取到 Top-8 时,DeepLSTM 缺陷检测效果最好.



(a) *F*1 值 (b) 查准率 (c) 召回率

Fig.9 Result of API misuse bug detection experiment

图 9 API 误用缺陷检测实验结果

为了进一步探究 API 误用类型与模型能力的关系,实验中,将 DeepLSTM 模型对应每个可接受阈值(top-*k*)的具体 API 误用缺陷报告情况整理在表 2 中,表格中检测正确的 API 误用用“√”表示,未检出的 API 误用用“–”表示.

Table 2 API misuse bug detection reports statistics

表 2 API 误用缺陷检测报告情况统计

测试用例	Top-1	Top-2	Top-3	Top-4	Top-5	Top-6	Top-7	Top-8	Top-9	Top-10
1	√	–	–	–	–	–	–	–	–	–
2	√	√	√	√	√	√	√	√	–	–
3	√	√	√	√	√	√	√	√	√	√
4	√	–	–	–	–	–	–	–	–	–
5	√	√	√	√	√	√	√	√	√	√
6	√	√	√	√	√	√	√	√	√	√
7	√	√	√	√	√	√	√	√	√	√
8	√	√	√	√	√	√	√	√	√	√
9	√	√	√	√	√	√	√	√	√	√
10	√	√	√	√	–	–	–	–	–	–
11	√	√	√	√	√	√	√	√	√	√
12	–	–	–	–	–	–	–	–	–	–
13	√	√	√	√	√	√	√	√	√	√
14	–	–	–	–	–	–	–	–	–	–

基于整理后的API误用缺陷检测报告情况,按照测试集中API误用的误用类型对缺陷检测报告情况进行进一步的可视化统计分析,如图 10 所示.

纵轴表示了不同类型的 API 误用缺陷,横轴表示检测出以及未检出的该类型 API 误用数量在该类型 API 误用总数中的占比,用颜色深蓝色、浅绿色分别表示检测出和未检出的两种情况,图中有 10 个独立的子图来分别表示每个可接受阈值(top-*k*)对应的 API 误用缺陷检测报告情况.

分析该图,可以看出:

- 随着 *k* 取值的增大,在某些 API 误用的检测上,模型的效果保持基本不变(错误的 API 调用),一定程度说明了本实验中采用的缺陷检测方法对检测该类 API 误用的有效性:当一个 API 误用在 API 调用序列中

不应该出现(使用了错误的 API)时,则应用本实验中按位置预测并比较检测 API 误用是可行且有效的.

- 但是与之相反的,随着 k 取值的增大,在某些 API 误用的检测上,模型效果不断降低(遗漏 API 调用),这一定程度上也说明了本实验中采用的缺陷检测方法对检测该类 API 误用的局限性:当一个关键性 API 在 API 调用序列中遗漏,仅靠当前的 API 调用序列作为预测的前文,很难推断出下一位置是否遗漏了某个 API.例如,某个关键性 API 可能在当前位置更靠后几个调用中出现,这种情况对目前采用的仅对一个位置的 API 误用缺陷进行检测的方法就具有很大的挑战性.
- 随着 k 取值的增大,在某些 API 误用的检测上,模型效果有较大的突变(多余的 API 调用、忽略异常处理),这与本实验中采用的测试集有一定的关系——多余的 API 调用和忽略异常处理的测试用例都仅有 1 个.但是在 k 取到一定范围(top-8)之前,本实验中采用的缺陷检测方法对忽略异常处理类型的 API 误用缺陷检测效果还是不错的.
- 就每个独立子图而言,也可以验证以上几点结论:本实验采用的 API 误用缺陷检测方法,对检测错误的 API 调用类型的 API 误用效果较好,而在发现遗漏 API 调用类型的 API 误用上,能力稍有欠缺.

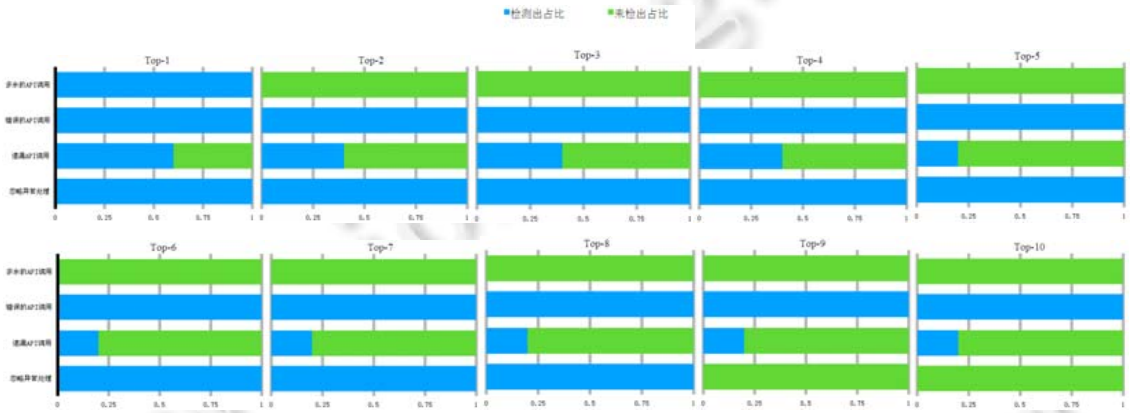


Fig.10 Analysis of relationship between acceptable thresholdand type of API misuse

图 10 可接受阈值与 API 误用类型关系分析

综上,本实验应用深度学习模型学习大量代码中的 API 使用规约,并应用在 API 误用缺陷检测上的方法.虽然由于大量开源代码中可能存在些许误用的 API 代码,影响到训练数据的质量,对模型的效果产生着一定的影响,但是本实验中,模型在以检测 Java 加密相关的 API 误用代码缺陷为例的实验中仍然起到了一定的效果,以可接受阈值为 8 时效果最佳($F1$ 值、准确率).在本实验中,模型对检测错误的 API 调用类型的 API 误用效果较好,而在发现遗漏 API 调用类型的 API 误用上能力稍有欠缺,这与每种 API 误用类型内在的规律特征密不可分.

5 结束语

本文将深度学习中的循环神经网络模型应用于 API 使用规约的学习及 API 误用缺陷的检测.将 API 调用组合、顺序及控制结构等方面的使用规约建模为 API 语法图和 API 调用序列.在大量的开源 Java 代码基础上,对代码进行静态分析,并构造大量 API 使用规约训练样本,对循环神经网络进行训练.在实验中通过尝试不同参数组合,选定并训练出较优的循环神经网络模型,并用于基于前文的 API 调用预测,通过预测结果与实际代码进行比较来发现潜在的 API 误用缺陷.随后,在以检测 Java 加密相关的 API 误用代码缺陷为例的实验中证明了该方法的有效性.实验表明,本方法检测错误的 API 调用类型的 API 误用时最为稳定有效.

本文提出的方法在一定程度上能自动检测 API 误用缺陷,并在某类 API 误用的检测中较为有效,但是还存在一些不足和可改进之处——在发现遗漏 API 调用类型的 API 误用上,能力稍有欠缺.在未来研究中,可考虑对多个位置的 API 调用进行预测比对,减少模型由于单个位置的预测带来的 API 是否漏用的不确定性.

References:

[1] Li Z, Wu JZ, Li MS. Study on key issues about API usage. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1716−1738 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5541.htm [doi: 10.13328/j.cnki.jos.005541]

[2] Zhou Y, Gu R, Chen T, Huang Z, Panichella S, Gall H. Analyzing APIs documentation and code to detect directive defects. In: Proc. of the 39th Int'l Conf. on Software Engineering. 2017. 27−37. [doi: 10.1109/ICSE.2017.11]

[3] Liu S, Bai G, Sun J, Dong JS. Towards using concurrent Java API correctly. In: Proc. of the 21st Int'l Conf. on Engineering of Complex Computer Systems. 2016. 219−222. [doi: 10.1109/ICECCS.2016.32]

[4] Sacramento P, Cabral B, Marques P. Unchecked exceptions: Can the programmer be trusted to document exceptions. In: Proc. of the 2nd Int'l Conf. on Innovative Views of NET Technologies. 2006.

[5] Amann S, Nadi S, Nguyen HA, *et al*. MUBench: A benchmark for API-misuse detectors. In: Proc. of the 13th Int'l Conf. on Mining Software Repositories. 2016. 464−467. [doi: http://dx.doi.org/10.1145/2901739.2903506]

[6] Gao Q, Zhang H, Wang J, *et al*. Fixing recurring crash bugs via analyzing q&a sites (T). In: Proc. of the 2015 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2015. 307−318. [doi: 10.1109/ASE.2015.81]

[7] Zhong H, Zhang L, Mei H. Mining invocation specifications for API libraries. Ruan Jian Xue Bao/Journal of Software, 2011,22(3): 408−416 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3931.htm [doi: 10.3724/SP.J.1001.2011.03931]

[8] Li Z, Zhou Y. PR-miner: Automatically extracting implicit programming rules and detecting violations in large software code. SIGSOFT Software Engineering Notes, 2005,30(5):306−315. [doi: 10.1145/1081706.1081755]

[9] Zhong H, Zhang L, Xie T, Mei H. Inferring resource specifications from natural language API documentation. In: Proc. of the 2009 IEEE/ACM Int'l Conf. on Automated Software Engineering. 2009. 307−318. [doi: 10.1109/ASE.2009.94]

[10] Wang S, Chollak D, Movshovitz-Attias D, Tan L. Bugram: Bug detection with *n*-gram language models. In: Proc. of the 2016 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. 2016. 708−719. [doi: 10.1145/2970276.2970341]

[11] https://en.wikipedia.org/wiki/Deep_learning

[12] Zheng ZY, Liang BW, Gu SY. TensorFlow: Googledeep Learning Framework, Put It into Practice. 2nd ed., Beijing: Publishing Houseof Electronic Industry, 2018 (in Chinese).

[13] Graves A, Jaitly N, Mohamed A. Hybrid speech recognition with deep bidirectional LSTM. In: Proc. of the 2013 IEEE Workshop on Automatic Speech Recognition and Understanding. 2013. 273−278. [doi: 10.1109/ASRU.2013.6707742]

[14] Luong MT, Pham H, Manning CD. Effective approaches to attention-based neural machine translation. In: Proc. of the Empirical Methods in Natural Language Processing. 2015. 1412−1421. [doi: 10.18653/v1/D15-1166]

[15] Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proc. of the Empirical Methods in Natural Language Processing. 2014. 1724−1734. [doi: 10.3115/v1/D14-1179]

[16] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997,9(8):1735−1780.

[17] Li H. Statistical Learning Method. Beijing: Tsinghua University Press, 2012 (in Chinese).

[18] Smith N, Bruggen DV, Tomassetti F. JavaParser: Visited analyse, transform and generate your Java code base. 2017. https://enterprise.leanpub.com/javaparservisited

[19] Grinberg M. Flask Web Development. Oreilly Vlg Gmbh & Co., 2014.

[20] Nadi S, Kr S, Mezini M, Bodden E. Jumping through hoops: Why do Java developers struggle with cryptography APIs? In: Proc. of the 38th Int'l Conf. on Software Engineering. 2016. 935−946. [doi: 10.1145/2884781.2884790]

[21] Egele M, Brumley D, Fratantonio Y, Kruegel C. An empirical study of cryptographic misuse in Android applications. In: Proc. of the Conf. on Computer and Communications Security. 2013. 73−84. [doi: 10.1145/2508859.2516693]

[22] Fahl S, Harbach M, Muders T, Smith M, Baumgärtner L, Freisleben B. Why eve and mallory love Android: An analysis of android SSL (in) security. In: Proc. of the Conf. on Computer and Communications Security. 2012. 50−61.

[23] Gousios G, Spinellis D. Mining software engineering data from GitHub. In: Proc. of the 2017 IEEE/ACM 39th Int'l Conf. on Software Engineering Companion. 2017. 501−502. [doi: 10.1109/ICSE-C.2017.164]

[24] Fowkes J, Sutton C. Parameter-free probabilistic API mining across GitHub. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. 2016. 254−265. [doi: 10.1145/2950290.2950319]

[25] Chen SF, Goodman J. An empirical study of smoothing techniques for language modeling. Computer Speech & Language, 1999, 13(4):359−394. [doi: 10.1006/csla.1999.0128]

附中文参考文献:

[1] 李正,吴敬征,李明树.API 使用的关键问题研究.软件学报,2018,29(6):1716−1738. http://www.jos.org.cn/1000-9825/5541.htm [doi: 10.13328/j.cnki.jos.005541]

[7] 钟浩,张路,梅宏.软件库调用规约挖掘.软件学报,2011,22(3):408−416. http://www.jos.org.cn/1000-9825/3931.htm [doi: 10.3724/SP.J.1001.2011.03931]

[12] 郑泽宇,梁博文,顾思宇.TensorFlow:实战 Google 深度学习框架.第 2 版,北京:电子工业出版社,2018.

[17] 李航.统计学习方法.北京:清华大学出版社,2012.

汪昕(1996－),女,福建泉州人,硕士生,CCF 学生会员,主要研究领域为智能化软件开发.

陈驰(1993－),男,博士生,主要研究领域为代码推荐.

赵逸凡(1995－),男,硕士生,CCF 专业会员,主要研究领域为智能化软件开发.

彭鑫(1979－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为智能化软件开发,移动计算,云计算.

赵文耘(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,智慧城市.