DeepGenFuzz:基于深度学习的高效PDF应用程序模糊测试用例生成框架

刘家豪, 江贺

引用本文

刘家豪, 江贺. DeepGenFuzz:基于深度学习的高效PDF应用程序模糊测试用例生成框架[J]. 计算机科学, 2024, 51(12): 53-62.

LIU Jiahao, JIANG He. DeepGenFuzz:An Efficient PDF Application Fuzzing Test Case Generation Framework Based on Deep Learning [J]. Computer Science, 2024, 51(12): 53-62.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231100179

DeepGenFuzz:基于深度学习的高效 PDF 应用程序模糊测试用例生成框架

刘家豪　江　贺
大连理工大学软件学院　辽宁 大连 116600
(1369347184@qq.com)

摘　要　PDF 文件是一种被广泛应用的重要文档格式。由于 PDF 文件的复杂性,PDF 相关的应用程序中存在的缺陷可能会导致严重后果,例如遭遇恶意攻击、信息错误呈现等。因此,针对 PDF 相关应用程序的测试成为当前研究的热点问题。目前最有效的方法是基于语法的模糊测试。然而,基于语法的模糊测试往往需要花费大量手工工作对复杂的语法规则进行总结和编写,严重阻碍了测试用例高效地自动化生成。深度学习技术为突破这一障碍提供了可行路径,但目前的方法生成的测试用例普遍质量较低,查找 bug 能力较差。进一步对其进行改进需要应对 3 个主要挑战,即数据集的筛选、测试用例覆盖率提升和测试用例大小增加两者间的平衡、测试用例的高效变异。因此,提出了一个基于深度学习的高效 PDF 应用程序模糊测试用例生成框架 DeepGenFuzz,利用 CNN,Seq2Seq 和 Transformer 等模型,通过数据筛选、对象生成、对象附加、高效变异等步骤生成高质量 PDF 测试用例。在 MuPDF 等 PDF 应用程序上的评估表明,DeepGenFuzz 生成的测试用例平均代码覆盖率明显高于 Learn&Fuzz 和 IUST-DeepFuzz 等目前最先进的工具,最高可达 8.12%～61.03%;bug 查找能力也远远优于 Learn&Fuzz 和 IUST-DeepFuzz 等最先进的工具,目前已经报告了在 7 个最流行的 PDF 应用程序中发现的 31 个未曾被报告的 bug,其中 25 个已经得到确认或修复,涵盖了所有被测程序。

关键词:PDF 应用程序;深度学习;模糊测试;测试用例;代码覆盖率
中图分类号　TP311

DeepGenFuzz:An Efficient PDF Application Fuzzing Test Case Generation Framework Based on Deep Learning

LIU Jiahao and JIANG He
School of Software,Dalian University of Technology, Dalian,Liaoning 116600,China

Abstract　PDF file is a widely used and important document format. Due to the complexity of PDF files,defects in PDF-related applications can lead to serious consequences such as malicious attacks and incorrect information rendering. Therefore,testing PDF-related applications has become a hot research topic. The most effective method currently is grammar-based fuzz testing,but it often requires a significant amount of manual work to summarize and write complex grammar rules,which seriously hinders the efficient automation of test case generation. Deep learning techniques provide a feasible solution to this challenge. However,the quality of test cases generated by current methods is generally low,and the ability to find bugs is poor. To further improve this,three main challenges need to be addressed:data set filtering,balancing test case coverage improvement and test case size increase,and efficient mutation of test cases. Therefore,this paper proposes a deep learning-based efficient PDF application fuzz test case generation framework called DeepGenFuzz. It utilizes models such as CNN,Seq2Seq,and Transformer to generate high-quality PDF test cases through steps including data filtering,object generation,object appending,and efficient mutation. Evaluations on PDF applications like MuPDF show that DeepGenFuzz generates test cases with significantly higher average code coverage compared to state-of-the-art tools like Learn&Fuzz and IUST-DeepFuzz,reaching up to 8.12% ～ 61.03%. Its bug-finding capabilities are also far superior to those of Learn&Fuzz and IUST-DeepFuzz. Currently,31 previously unreported bugs have been discovered in the seven most popular PDF applications,among which 25 have been confirmed or fixed,covering all tested programs.

Keywords　PDF application,Deep learning,Fuzz testing,Test cases, Code coverage

1　引言

PDF 是一种由 Adobe 公司开发的电子文档格式,它可以在不同的操作系统和计算机上以相同的方式呈现。其由于跨平台性和可靠性,被广泛应用于商业、学术、出版、设计和存档等诸多领域。

然而,也正是由于 PDF 文件的广泛应用性和复杂性,用于读取和处理 PDF 文件的应用程序中存在的缺陷和漏洞

到稿日期:2023-11-27　返修日期:2024-04-28
通信作者:江贺(jianghe@dlut.edu.cn)

可能会导致严重的后果[1]。例如 Adobe Reader——最流行的 PDF 查看器之一，也是 2012 年期间 Windows 计算机上第三大常见的计算机入侵源[2]，攻击者可以通过 Adobe Reader 中存在的漏洞，破坏计算机或盗取数据。此外，Kuchta 等[3]报告了 Chrome 和 Mozilla 支持论坛上的数百个关于 PDF 文件无法正确呈现信息的投诉。

虽然近年来研究人员提出了一些有效的软件测试方法，但是专门针对 PDF 应用程序的测试方法较少。到目前为止，模糊测试是最常应用于 PDF 应用程序的测试方法[4-17]。具体来说，模糊测试方法大致可以分为 4 种类型，分别是：白盒模糊测试[4-5]、黑盒模糊测试[6-7]、灰盒模糊测试[8-13]，以及基于语法的模糊测试[14-17]。

白盒模糊测试是基于对被测试软件的内部结构和逻辑的了解进行的测试，要求测试人员对被测软件内部结构和逻辑足够熟悉。白盒模糊测试虽然可以深入挖掘系统的内部漏洞，但仍然可能由于测试用例设计的问题而遗漏一些逻辑错误，导致测试覆盖不全面[18-19]。黑盒模糊测试是一种基于输入输出的测试方法，几乎不需要对测试软件有任何了解，但会生成很多冗余的测试用例，且代码路径覆盖有限[19-20]。灰盒模糊测试是介于白盒和黑盒测试之间的一种测试方法，一定程度上综合了白盒和黑盒测试技术的优势，但仍然缺乏对系统内部结构的完全可见性，可能无法覆盖所有的代码路径，且对系统变动敏感度较高。

基于语法的模糊测试是一种基于输入语法规则的测试方法，它利用目标程序的语法规则或文件格式定义生成有效的、符合语法要求的输入数据，并对生成的输入进行变异和修改。相比于黑盒模糊测试、白盒模糊测试和灰盒模糊测试，基于语法的模糊测试是目前对需要高度结构化输入（如 PDF 文件）的程序进行模糊测试的最有效技术，它在生成输入时更具约束和准确性，能够更有针对性地探索潜在的问题，同时减少无效的测试用例数量。因此，对 PDF 应用程序测试来说，基于语法的模糊测试是最优选择。

然而，相比于黑盒模糊测试、白盒模糊测试和灰盒模糊测试的自动化，基于语法的模糊测试往往需要人工总结输入格式的语法规则，对于具有复杂结构的输入格式，例如 PDF 文件格式，想要构建其语法，需要深入了解其规范。尤其是 PDF 格式具有长达上千页，包含数千条语法规则的规范文档[21]，通过手工去构建 PDF 所有必需的语法规则的成本几乎是不可承受的，目前尚未出现通过手工总结 PDF 语法规则的生成器[22]。

因此，本文研究了如何利用深度学习技术来突破手工构建 PDF 文件语法规则的障碍。目前在这方面已经有了一些工作。Learn&Fuzz[15]是通过深度学习模型学习 PDF 文件语法规则的第一次尝试。然而，PDF 文件格式包含文本数据和二进制数据，Learn&Fuzz 中只学习了文本数据。Jitsunari 等[16]从使用深度学习模型自动化生成二进制数据的角度改进了 Learn&Fuzz。IUST-DeepFuzz[17]将存储在文件中的文本数据与二进制数据区分开来，以缓解 Learn&Fuzz 生成的测试用例多样性低的问题。但目前这些方法生成的测试用例普遍质量低下，查找 bug 能力差。要想做进一步的改进，需要应对 3 方面挑战。首先，用于深度学习模型训练的数据集大多来自公共网络或公开数据集，但是这样的数据集中种子的执行效果与随机选择种子的执行效果几乎相同[23]。因此本文需要解决数据集筛选的问题，即如何在数据集中筛选出对于后续模型训练和 bug 查找来说质量更高的数据。其次，现有工作中的 PDF 对象附加方法和 PDF 文件更新方法往往仅关注生成测试用例的覆盖率提升，却忽略了测试用例大小增加对模糊测试的负面影响[22]。因此，本文要应对第二个挑战，即在测试用例覆盖率提升和大小增加之间维持平衡。最后，目前的测试用例变异方法大都是随机变异，但这样的变异方法效率较低，许多变异文件的覆盖率反而下降，且效果不稳定。因此，本文要面对的第三个挑战是对测试用例的有效变异。

针对上述挑战，本文提出了一个基于深度学习技术的高效 PDF 应用程序模糊测试用例生成框架，称为 DeepGen-Fuzz，用于生成 PDF 测试用例，进而查找 PDF 应用程序中的 bug。首先，DeepGenFuzz 通过一个 CNN 分类器从庞杂的原始数据集中筛选出那些更有可能覆盖程序中错误代码的 PDF 文件，组成后续深度学习模型的训练集。其次，从训练集中提取 PDF 对象的文本数据和二进制数据，并分别使用 Seq2Seq 预测模型进行训练，然后使用新的采样方法生成 PDF 对象的文本数据和二进制数据，将它们组合成完整的 PDF 对象。接着，提出了能够在测试用例覆盖率增加和大小之间维持最佳平衡的 PDF 对象附加方法和 PDF 更新方法，将生成的 PDF 对象附加到正常的 PDF 文件中。最后，通过 Transformer 模型，对生成的 PDF 文件进行高效的变异，进一步提高测试用例的覆盖率。

为了评估 DeepGenFuzz 的有效性，本文在一个开源 PDF 应用程序 MuPDF 的查看器上进行了实验，并设计了多个对比实验，将 DeepGenFuzz 与目前最先进的工作进行比较。结果显示，DeepGenFuzz 生成的测试用例平均代码覆盖率明显优于 Learn&Fuzz 和 IUST-DeepFuzz 等最先进的工具，最高可达 8.12%～61.03%；且查找 bug 能力最高，本文已经报告了在 7 个最流行的 PDF 应用程序中发现的 31 个未曾被报告的 bug，其中 25 个已经得到确认或修复。

综上所述，本文的主要贡献如下：

1）提出了一个基于深度学习技术的高效 PDF 应用程序模糊测试用例生成框架 DeepGenFuzz，通过数据筛选、模型训练、对象生成、对象附加、有效变异等多个步骤，实现了高质量 PDF 测试用例的生成。

2）设计了一系列实验，验证了 DeepGenFuzz 相比目前最先进的工具，在代码覆盖率提升和 bug 查找能力上有显著提高。

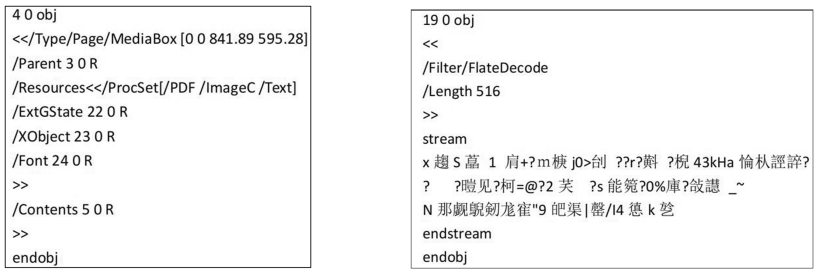
3）在 7 个最流行的 PDF 应用程序上进行了大量测试工作。本文报告了 31 个未曾被发现的 bug，其中 25 个已经得到开发者确认或修复。

本文第 2 章介绍了研究背景；第 3 章具体介绍了本文方法；第 4 章描述了评估结果；最后总结了研究并对未来的工作进行了展望。

2 背景

一般情况下，PDF 文件主要由 4 部分组成，分别是文件头、文件体、交叉引用表和文件尾。文件头指明遵从的 PDF 规范的版本；文件体是 PDF 文件的主要部分，由一系列的对象组成；交叉引用表用于描述 PDF 文件中对象的位置和编号，可以快速定位和访问 PDF 文件中的对象；文件尾声明了交叉引用表的地址，指明文件体的根对象。

对象是组成 PDF 文件结构的基本单元，构成了 PDF 文件的主体部分，所有对象都遵循相同的基本结构和格式，定义和组合对象的规则构成了 PDF 格式规范的大部分内容。正因为对象具有这几个特点，使用深度学习模型学习 PDF 语法结构便具备了可能性。对象根据包含的数据的类型可以分为两类：纯文本对象和包含二进制数据的对象。如图 1(a)所示，纯文本对象存储的是纯文本数据，一般用于表示文本内容、字体信息、页面结构等。包含二进制数据的对象存储的是文本数据和二进制数据，如图 1(b)所示，其中二进制数据由 stream 和 endstream 分隔，一般用于表示图像、音频、视频等媒体文件，或者其他无法用纯文本表示的数据，不可直接查看和编辑。



(a) (b)

图 1 不同类型的 PDF 对象

Fig. 1 Different types of PDF objects

3 DeepGenFuzz

本章首先介绍了 DeepGenFuzz 的框架；然后具体介绍了应对 3 个挑战的方法，即数据集的筛选、PDF 对象的生成与附加、PDF 的有效变异。

3.1 DeepGenFuzz 的概述

DeepGenFuzz 的框架如图 2 所示。DeepGenFuzz 由 5 个主要步骤组成。1)从公共网络、公开数据集以及被测程序的论坛或 bug 报告网站中收集大量的 PDF 文件，接着提取其中小部分数据进行模糊测试，并利用模糊测试结果训练 CNN 分类器，然后利用 CNN 分类器筛选剩余数据中更有可能覆盖被测程序错误代码的数据组成训练集。2)提取训练集数据中的文本数据和二进制数据，分别训练 Seq2Seq 预测模型。3)从训练集中随机提取文本数据和二进制数据的起始序列输入训练好的 Seq2Seq 模型，并采用新提出的采样方法，生成完整的文本数据和二进制数据组合成完整的 PDF 对象。4)使用本文新提出的 PDF 对象附加方法和 PDF 文件更新方法将生成的 PDF 对象附加到完整的 PDF 文件中。5)从数据集中随机抽取数据，并进行随机变异，提取变异后覆盖率最高的文件作为目标序列，变异前的文件作为输入序列，训练一个实现有效变异的 Transformer 模型。模型训练完后，将第 4)步得到的 PDF 文件输入模型中进行有效变异来进一步提升代码覆盖率。

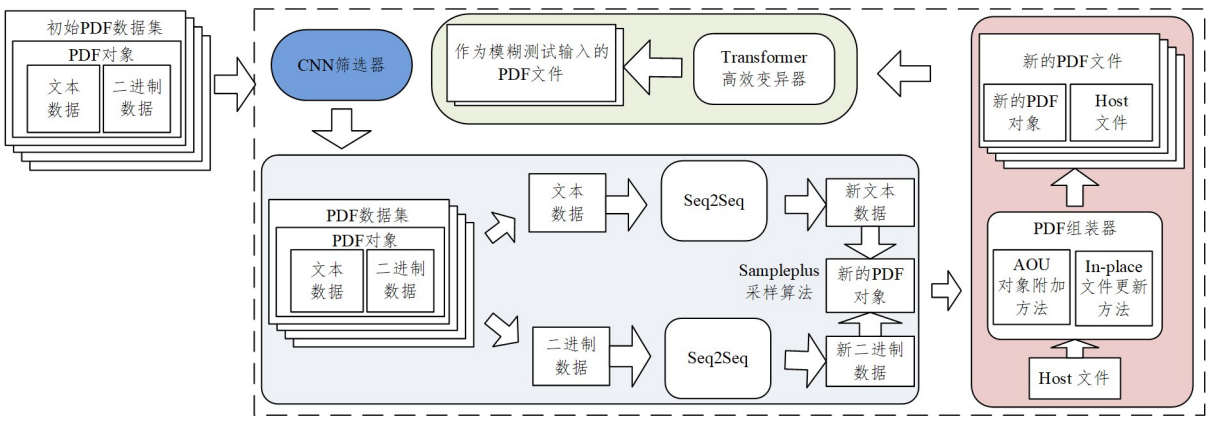


图 2 DeepGenFuzz 框架流程图

Fig. 2 Flowchart of DeepGenFuzz framework

3.2 数据集的筛选

一些研究已经证实了在软件的 bug 分布中存在着“Pareto principle”[24-26]，即 80% 的 bug 通常只分布在 20% 的源文件中，这表明 bug 往往呈现出集中分布的趋势。进一步可以得出推论：在错误代码的附近往往可能存在其他错误。因为当代码执行过程中出现错误时，常会导致程序的执行流程发生变化，从而引发其他错误。另外，即便错误被修改，修改或添加的代码也有可能引入新的错误[27-28]。因此，本文希望能从数据集中筛选出更有可能引发程序崩溃的文件，即便对应的错误已经被发现或修改。

具体来说，首先从原始数据集中随机提取一小部分种子，使用这些种子对被测程序进行反复的模糊测试，提取其中触发崩溃的种子和未触发崩溃的种子作为输入，训练一个二分类的 CNN 分类器，继而使用分类器筛选原始数据集中剩余种子中能触发被测程序崩溃的文件，并使用这些文件作为后续模型的训练集。这个任务在一定程度上类似于图像分类中的二分类任务，图像分类任务中物体的特征是由几个像素的组合来表示的，CNN 模型可以很好地提取这种特征[29]。而考虑到本文的工作中造成程序崩溃的输入特性一般也可以通过 PDF 文件中的几个字节的组合来表示，本文选择使用 CNN 来构建种子分类器模型。另外，相比于更适合字节序列训练的 RNN 模型，CNN 擅长处理长数据，较大的 PDF 文件长度可达十几万个字节，而输入越长，RNN 模型遗忘前一特征的速度越快。因此，CNN 更适合作为种子分类器。

对于每个 PDF 文件输入，首先以二进制形式提取，并将其表示为向量 $\boldsymbol{x}=\langle x_1, x_2, \cdots, x_n \rangle$，而向量的长度 n 就是最大的 PDF 文件的长度，$x_i = byte_i\ (x_i \in \{0, \cdots, 255\})$，另外用 $x_i = 256$ 来填充长度小于 n 的部分。对引发被测程序崩溃的文件赋予标签 1，反之赋予标签 0。

模型训练完成后，将原始数据集中的剩余种子输入 CNN 模型进行分类。为了进一步降低假阳性，本文设置了一个阈值 k，即当一个 PDF 文件被判定为标签 1 的概率大于 k 而不是 0.5 时，才赋予它标签 1。具体的 k 值可以根据被测程序而定，k 值要在低假阳性和足够的训练集数量之间取得平衡。最后，提取被判定为标签 1 的文件，并将其作为后续过程中模型的训练集。

3.3 PDF 对象的生成与附加

在获取训练集后，本文提取其中的 PDF 对象，再采用 IUST-DeepFuzz[17] 中的方法，使用一个特殊序列〈/stream〉替换掉 PDF 对象中的二进制数据，而被替换的二进制数据另外保存，这样就实现了文本数据和二进制数据的分离。然后分别使用文本数据和二进制数据训练模型，接着从训练集中随机提取文本数据和二进制数据的起始序列输入训练好的模型中来生成完整的文本数据和二进制数据，最后使用生成的二进制数据替换掉文本数据中的〈/stream〉，这样就在保证文本数据和二进制数据都包含变异和异常结构的前提下生成了完整的 PDF 对象。

具体来说，在训练文本数据的模型时，本文将经过处理的文本数据连接成一个长序列，然后切割成固定长度 d 的序列作为模型的输入。接着使用序列的前 $d-1$ 个元素作为输入序列，后 $d-1$ 个元素作为目标序列，训练一个 Seq2Seq 模型作为预测模型。在模型训练完成后，再随机选择训练集中 PDF 对象的起始部分作为起始序列输入模型。对于不包含〈/stream〉的 PDF 对象，使用从对象开头到它的第 n 个字符作为起始序列，n 随机取值且小于对象的字符数。而对于包含〈/stream〉的 PDF 对象，使用从对象开头到〈/stream〉的对象作为起始序列。每次选择概率最大的字符作为预测结果，然后将预测结果加入到输入序列中，接着使用新的输入序列再次预测下一个字符，直到生成一个完整的 PDF 对象。然而，模型训练完成之后，概率关系已经确定，即输入一个起始序列，模型生成的常常是相同的 PDF 对象。所以 Learn&Fuzz 提出了 Sample 和 SampleSpace 这两种采样方法，以增加生成对象的多样性。但是在实际应用过程中，可以发现生成的对象的多样性依然较低。因此本文在这两种采样方法的基础上进行了改进，提出了新的采样方法 Sampleplus 和 SampleSpaceplus。Sampleplus 如算法 1 所示。SampleSpaceplus 与 Sampleplus 同理，只是改为使用 Samplespace 算法进行概率采样。通过实验可以发现，两种新的采样方法生成的对象的多样性得到了进一步的提高。

算法 1 Sampleplus 采样算法

```
输入：seq / * 随机抽取的 PDF 对象 * /
输出：完整的 PDF 对象
1. if "〈/stream〉" is in seq then
2.     seq←seq from start to "〈/stream〉"
3. else
4.     n←randint(0,len(seq))
5.     seq←seq from start to n
6. while ¬seq. endswith("endobj") do
7.     c,p(c)←sample(seq)/ * 使用 sample 对 seq 进行概率采样，c 为概率最大的元素，p(c)为对应概率 * /
8.     p_fuzz←random(0,1)
9.     if(p_fuzz > t_fuzz)∧(p(c) > p_t)then
10.    s←randint(1,4)
11.    if s==1 then
12.        c←argmin(seq) / * 若 s 为 1，用概率最小元素替换 c * /
13.    else if s==2 then
14.        c←randint(0,255)/ * 若 s 为 2，用随机元素替换 c * /
15.    else if s==3 then
16.        c←seq[-1]/ * 若 s 为 3，重复前一元素 * /
17.    else if s==4 then
18.        c←null/ * 若 s 为 4，删除 c 元素 * /
19.    end if
20.    seq←seq+c
21. end while
```

对于二进制数据，本文使用与文本数据相同的方式训练 Seq2Seq 预测模型。不同的是，相较于普遍长度在几百字符左右的文本数据，二进制数据的长度则可能达到几万甚至十几万个字符。过长的输入序列会严重影响模型预测下一个字符的效率，因此本文设置了一个输入序列长度阈值 h，即当输入序列长度小于等于 h 时，将整个输入序列输入模型预测下一个字符；而当输入序列长度大于 h 时，仅将输入序列的最后 h 个字符作为输入模型来预测下一个字符。另外，为了避免二进制数据的生成时间过长，本文设置了一个终止参数 max_len，当生成的二进制数据长度达到 max_len 而生成还未结束时，就强制终止生成过程，并在生成的二进制数据末尾添加固定的结束标志，从而构建一条完整的二进制数据。另外，在二进制数据的生成过程中同样要用到前面提到的采样方法。最后，若生成的文本数据中包含〈/stream〉，就随机选择模型生成的二进制数据替换掉〈/stream〉，这样就构成了完整的 PDF 对象。

通过上述过程，就生成了完整的 PDF 对象，然而本文需要测试的是以完整的 PDF 文件作为输入的程序，而不是单独

的 PDF 对象，所以需要进一步将生成的 PDF 对象附加到完整的 PDF 文件中。PDF 文件可以根据 PDF 参考指南[21]中的方法进行增量更新，具体来说，首先将生成的 PDF 对象附加到完整的格式良好的 PDF 文件末尾，然后添加一个交叉引用表部分，并插入一个新的尾部。Learn&Fuzz 中作者仅用新生成的 PDF 对象覆盖了 PDF 文件中原本对象中的最后一个。IUST-DeepFuzz 对此做了改进，采用了两种模式：单对象更新(Single Object Update, SOU)，在此模式下每个文件中只会更改对象 ID 最大的对象；多对象更新(Multiple Objects Update, MOU)，在此模式下计算文件中对象的总数，并且随机选择一定比例对象用新对象覆盖。然而，在实际实验中可以发现，Learn&Fuzz 和 SOU 模式的对象附加方法只附加了一个新的对象，使得新生成的 PDF 文件大小增加不大，但相应的代码覆盖率提升也相对较小；MOU 模型附加多个 PDF 对象，使得代码覆盖率提升较大，但生成的 PDF 文件大小也大大增加。

因此本文提出了一种新的将 PDF 对象附加到 PDF 文件中的模式 AOU(All Objects Update)，以保持覆盖率提升和文件大小增加之间的平衡。具体而言，若 PDF 文件中有 n 个对象，那么就生成 n 个 PDF 文件，第一个文件中原本的第一个对象被覆盖，第二个文件中第二个对象被覆盖，以此类推，第 n 个文件中第 n 个对象被覆盖。接着再获取这 n 个 PDF 文件的覆盖率，并对这 n 个文件中的 PDF 对象再次进行覆盖：原本第一个对象被覆盖的文件，就再覆盖其第二个对象；第二个对象被覆盖的文件，就再覆盖它的第三个对象；而第 n 个文件，则覆盖它的第一个对象。这样，就再次获得了 n 个 PDF 文件。依此类推，直到生成文件的覆盖率不再增加为止。这种对象附加方法，保证了 PDF 文件中每个对象都能获得更新，可以在一定程度上实现代码覆盖率提升和文件大小增加的平衡。

然而，即便是使用 AOU 模式来将 PDF 对象附加到 PDF 文件中，PDF 文件大小的增加也依然对模糊测试有着很大的负面影响。根据 PDF 参考指南中的说明，增量更新后，PDF 文件中被删除的对象依然在文件中保持不变，但它们的交叉引用项被标记为已删除。也就是说，PDF 文件中被覆盖的对象，不会再被读取，但依然保存在 PDF 文件中，这就导致 PDF 文件中存在大量冗余的无用数据。因此，本文提出了一种新的 PDF 文件更新方法 In-place Update，当更新 PDF 对象时，不再将其附加在 PDF 文件末尾，而是直接将要被覆盖的 PDF 对象删除，并将要更新的 PDF 对象插入到它的原位置，最后再修改交叉引用表。这样的更新方法，直接删除了无用的 PDF 对象，且不用附加新的交叉引用表和文件尾，使得更新文件的增长大大减小。

3.4 PDF 的有效变异

目前的工作大多采用随机变异方式对测试用例进行变异，例如 Learn&Fuzz 中使用了一种简单黑盒随机模糊算法 Random，它随机选择文件中的一个位置，然后用 0 到 255 之间的随机值替换字节值。然而，这种随机变异往往效率较低，且效果不稳定。因此，本文提出了一种新的有效变异方式 OptMutation。首先，在数据集中随机抽取 500 个 PDF 文件，然后使用不同模糊系数的 Random 算法对它们进行模糊，每个模糊系数重复 10 次，然后获取 10 次的平均行覆盖数，表示为 mutation。另外使用 500 个原始 PDF 文件的行覆盖数作为基线，表示为 baseline。参考已有工作中的相关数据和 Transformer 模型实际训练时间与效果，500 个文件是一个较为合适的文件数量。实验结果如图 3 所示。可以发现，当模糊系数为 10000，即文件中被变异的字节数为文件的长度除以 10000 时，获得的 PDF 文件的平均覆盖数最高。当模糊系数过小时，变异字节数过多，文件结构遭到过度破坏，导致覆盖率反而低于原始文件；当模糊系数过大时，变异字节数过少，起不到模糊效果，导致覆盖率提升较小或者覆盖率反而低于原始文件。

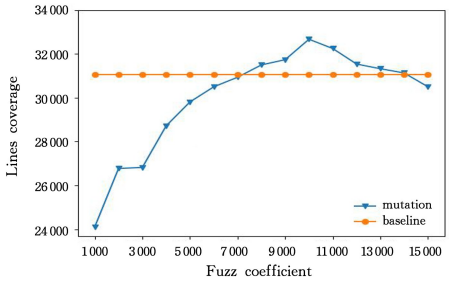


图 3 不同模糊系数生成的 PDF 文件的行覆盖数

Fig. 3 Lines coverage of PDF files generated by different fuzz coefficients

因此，本文采用模糊系数为 10000 的 Random 算法，对上述的 500 个 PDF 文件中的每一个 PDF 文件生成 100 个变体，分别获取其覆盖率，然后将覆盖率最高的变体提取出来。最后，使用 500 个未变异 PDF 文件作为源文件，500 个覆盖率最高的变体作为目标文件，训练 Transformer 模型，来学习有效的变异。具体来说，首先以二进制方式提取每个 PDF 文件的内容，每 x 个字节切割成一条序列，每个文件中最后不足 x 的部分，以一个特殊元素〈/p〉填充至 x。然后，分别将原文件提取的序列和变体文件提取的序列作为源序列和目标序列输入 Transformer 模型训练。在训练完成后，使用相同的方式提取需要变异的 PDF 文件的内容，并将其切分成长度为 x 的输入序列输入模型，模型每次选择概率最大的元素存入输出，直到组成输出序列。在生成所有输出序列后，就将生成的输出内容转回二进制格式，并拼接成完整的 PDF 文件，这样就获得了有效变异的 PDF 文件。然而，在实际应用过程中可以发现，模型输出的 PDF 文件中发生变异的字节数往往远远超出理论模糊字节数，导致变异之后的 PDF 文件覆盖率提升不明显或覆盖率反而下降。因此，本文设置了一个变异阈值 q，用于控制变异字节数，即当预测的概率最大的元素的概率超过 q 时，才将此元素加入输出序列，否则将输入序列中同一位置的元素加入输出序列。

4 评估

本文在 MuPDF 的查看器上进行实验来评估 DeepGenFuzz 的有效性。MuPDF 是最流行的开源 PDF 文档查看器和渲染器之一，它提供了一系列用于处理 PDF 文件的命令行工具。具体来说，本文实现 Learn&Fuzz[15] 和 IUST-Deep-

Fuzz[17]方法，并在 MuPDF 查看器上与本文提出的方法 DeepGenFuzz 进行有意义的比较。实验旨在回答以下 3 个研究问题。

RQ1：DeepGenFuzz 在提升测试用例质量及查找 PDF 应用程序中 bug 方面是否有效？

RQ2：DeepGenFuzz 中提出的各个策略是否都能有效地提高整体方法的质量？

RQ3：DeepGenFuzz 中每个策略的效率如何？

在这些 RQ 中，RQ1 评估 DeepGenFuzz 生成高质量 PDF 测试用例的能力和实际查找 PDF 应用程序 bug 的能力。RQ2 集中于 DeepGenFuzz 中每个策略的贡献。RQ3 主要研究 DeepGenFuzz 中每个策略引起的开销。

4.1 评估指标

本文使用两个主要指标来评估本文方法的有效性：代码覆盖率和发现 bug 数量。

代码覆盖率：本文中选择行覆盖率，它由 lcov 工具测量获得。行覆盖率是指测试用例执行期间覆盖的代码行数与总代码行数的比例，具体公式为：行覆盖率＝(程序执行代码行数/程序代码总行数) * 100%。它的优点是粒度细，可以帮助发现未被测试到的代码，从而提高测试的全面性。

发现 bug 数量：在被测程序中发现的 bug 数量是验证软件测试方法的最优标准。

此外，Learn&Fuzz 中还提出了另一个评估指标，即通过率。具体来说，对于每次测试执行，以编程方式检查被测程序的执行日志中是否存在解析错误消息。如果没有错误消息，则称此测试为通过，否则称其为失败。但是，这个指标主要用来评估学习的质量，而非测试用例对被测程序的覆盖能力和发现 bug 的能力。事实上，Learn&Fuzz 的作者也表示此指标对于模糊测试的目的来说并不重要，因此本文没有选择此指标来评估本文方法。

4.2 实验设置

DeepGenFuzz 主要依靠 Python3.9 实现，深度学习框架选择了 PyTorch。对于 CNN 筛选器，阈值 k 设置为 0.8；对文本数据的 Seq2Seq 生成模型，序列切分长度 d 设置为 20；对于二进制数据的 Seq2Seq 生成模型，序列切分长度 d 也设置为 20，输入序列长度阈值 h 设置为 200，终止参数 max_len 设置为 20000；对于有效变异 OptMutation 方法，序列切分长度 x 设置为 100，变异阈值 q 设置为 0.99999。对于模型具体设置，CNN 模型使用两个卷积层，第一个卷积层输入通道数为 1，输出通道数为 16，卷积核大小为 3；第二个卷积层输入通道数为 16，输出通道数为 32，卷积核大小为 3。每个卷积层后都有一个池化层和 ReLU 激活函数，池化层都采用最大池化，池化核大小为 2。Dropout 层丢弃概率为 0.5。最后是两个全连接层，最终输出特征数为 2；Seq2Seq 模型包含一个 embedding 层，维度为 128。编码器层、解码器层都使用 LSTM 构造，每个 LSTM 由 256 个隐藏状态构成。Dropout 层丢弃概率为 0.1；Transformer 模型的字嵌入层和位置嵌入层维度均为 256，FeedForward 层隐藏神经元个数为 1024，注意力机制中 Q，K，V 维度均设置为 64，编码器层和解码器层均为 6 层，多头注意力机制中的注意力头数为 8。模型的训练和实验都是在 Dell Inc(Intel Xeon Gold 6226R @ 2.90 GHz CPU，64 GB RAM)上运行的，该机器还具有 NVIDIA RTX 3080 GPU，系统为 Ubuntu20.04。

4.3 数据集和 host 文件

深度学习模型的训练需要足够的数据集。本文从公开网络和各种公开数据集以及被测程序的论坛或错误报告网站中共获取了超过 3000 个 PDF 文件，进一步使用 CNN 分类器从中筛选出了 493 个 PDF 文件，并从这些文件中提取了 65494 个 PDF 对象和 32921 条二进制数据作为 Seq2Seq 模型的训练集。另外还需要选择 host 文件，host 文件是用于附加生成的 PDF 对象的完整 PDF 文件。在 Learn&Fuzz 中，作者仅仅选择了数据集中最小的 3 个 PDF 文件作为 host 文件。而为了让实验的结果更具有普适性，本文选择了数据集中最大、最小和中位数大小的 3 个 PDF 文件作为 host 文件，分别表示为 host_max，host_min，host_med，它们的具体信息如表 1 所列。

表 1 host 文件详细信息

Table 1 Host file details

Hosts	File size/kB	Number of objects	Lines coverage
host_max	897	139	13345
host_med	131	27	13036
host_min	17	16	10764

4.4 对 RQ1 的调查

为了回答 RQ1，本文将 DeepGenFuzz 与目前最先进的方法 Learn&Fuzz 和 IUST-DeepFuzz 进行比较。对于 Learn&Fuzz 和 IUST-DeepFuzz 方法，采用相应工作[15,17]中发布的实现。为了公平比较，同样使用 DeepGenFuzz 的模型训练集作为 Learn&Fuzz 和 IUST-DeepFuzz 中模型的训练集。具体来说，使用 3 种方法分别在 host_max，host_med 和 host_min 上各生成 1000 个 PDF 文件，并测量了它们的覆盖率，结果如图 4 所示。

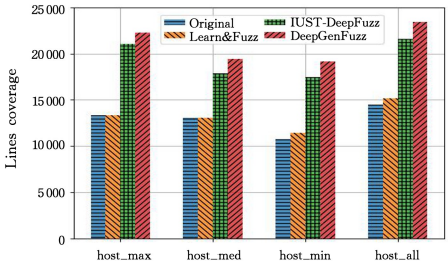


图 4 host 文件和 3 种方法生成 PDF 文件的行覆盖数

Fig. 4 Lines coverage of host files and PDF files generated by three methods

从图 4 中可以看到，相比于原始 host 文件，3 种方法生成的 PDF 文件都一定程度地提升了覆盖率。其中 Learn&Fuzz 效果最差，原因是对二进制数据的忽略和对象附加位置的单一，严重限制了 Learn&Fuzz 生成的 PDF 文件的覆盖率提升。IUST-DeepFuzz 效果优于 Learn&Fuzz，而 DeepGenFuzz 效果最好，在 3 个 host 文件上都取得了最高行覆盖的提升。相对而言，DeepGenFuzz 生成的 PDF 文件行覆盖数比 Learn&Fuzz 平均高出 61.03%，比 IUST-DeepFuzz 平均高出 8.12%。

表 2 中列出了 3 种方法发现的 bug 数量，其中‘—’表示方法未对对应程序进行测试。可以明显看出 DeepGenFuzz 在查找 bug 能力方面明显优于 Learn&Fuzz 和 IUST-DeepFuzz，Learn&Fuzz 仅在 Edge 浏览器的 PDF 解析器中发现了 1 个 bug[15]，IUST-DeepFuzz 没有发现任何 bug[17]，而 DeepGenFuzz 则在 Firefox，MuPDF，XPDF，Poppler，Ghostpdl，Podofo 和 QPDF 这 7 个最流行的 PDF 应用程序中发现了 31 个未曾被报告的 bug，其中 25 个已经得到确认或修复。图 5 中给出了发现的 bug 的类型的比例分布，可以看到发现的 bug 类型包括 7 种，其中 logic error 占比最大。虽然图 4 中的覆盖率结果可能会因为应用程序的不同而略有不同，但从表 2 和图 5 中可以看到，本文方法在测试的 7 个最流行的 PDF 应用程序中都发现了新的 bug 且 bug 包含多种类型，可以证明 DeepGenFuzz 的有效性广泛适用于不同的 PDF 应用程序。

表 2　3 种方法发现的 bug 数量

Table 2　Number of bugs discovered by three methods

Approach	Target program								
	Edge	Firefox	MuPDF	XPDF	Poppler	Ghostpdl	PoDoFo	QPDF	Total bugs
Learn&Fuzz	1	—	—	—	—	—	—	—	1
IUST-DeepFuzz	—	—	0	—	—	—	—	—	0
DeepGenFuzz	—	1	9	1	4	5	3	2	25

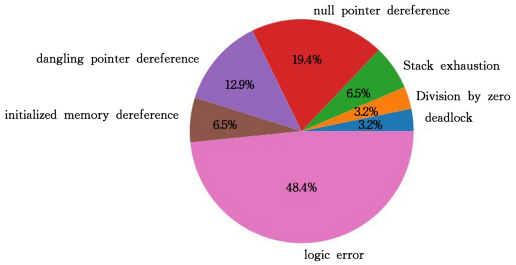


图 5　DeepGenFuzz 发现 bug 种类分布

Fig. 5　Type distribution of bugs discovered by DeepGenFuzz

接着，本文对发现的部分 bug 进行更加具体的分析。例如，DeepGenFuzz 在 Ghostpdl 和 Mupdf 中都发现了错误修复代码引入的新的错误，如 Bugzilla 平台上的 Bug-706865 和 Bug-707135，这证明了 3.2 节中思路的正确性。为了进一步验证 CNN 筛选器的有效性，本文构造了另一个版本的 DeepGenFuzz，即 DeepGenFuzz-rand，它在初始阶段不再使用 CNN 筛选器筛选原始数据集，而是直接使用相同概率随机抽取数据构建后续模型的训练集。本文使用 DeepGenFuzz-rand 生成的测试用例测试了相同版本的 Ghostpdl 和 Mupdf，每次测试持续时间 24 h，各重复 10 次，它们都没能发现 Bug-706865 和 Bug-707135。这证明了 CNN 筛选器的有效性。

另外，DeepGenFuzz 在 Xpdf 中发现了 CVE-2023-3436：若 PDF 对象流的“Length”字段本身位于另一个对象流中，则会导致 Xpdf 4.04 死锁。本文具体分析了引发 CVE-2023-3436 的测试用例，发现是 Sampleplus 算法通过采样将“Length”生成在了另一个对象流中，从而触发了这一错误，这证明了 Sampleplus 采样算法的有效性。

综上所述，在提高代码覆盖率能力上，DeepGenFuzz 生成的 PDF 测试用例的行覆盖率要明显高于目前最先进的方法 Learn&Fuzz 和 IUST-DeepFuzz，最高可达 8.12%～61.03%；在查找 bug 方面，DeepGenFuzz 找到的 bug 数量更是远远多于二者。这充分证明了 DeepGenFuzz 在提升测试用例质量及查找 PDF 应用程序中 bug 方面的有效性。另外，通过对发现的部分 bug 的分析，证明了 bug 的发现确实是依赖于 DeepGenFuzz 中的组件与算法，而非偶然性。

4.5　对 RQ2 的调查

为了更深入地了解 DeepGenFuzz 的工作原理，在本 RQ，将研究 DeepGenFuzz 的各个策略的有效性。

4.5.1　采样方法的有效性

在训练深度学习模型时，一个重要的参数是使用的 epoch 数。为了评估本文新提出的采样方法的有效性，本节报告了 Seq2Seq 模型训练 10，20，30，40，50 个 epoch 后获得的实验结果。首先，随机在 PDF 数据集中抽取 10 个 PDF 文件，从训练集中随机选择 100 个完整的 PDF 对象，使用 Learn&Fuzz 中的方法将这 10 个 PDF 文件和 100 个 PDF 对象进行组合，得到 1 000 个 PDF 文件作为基线，表示为 baseline。然后，从这 100 个 PDF 对象中提取起始序列输入训练好的模型，并分别采用 Learn&Fuzz 中提出的两种采样方法 Sample 和 SampleSpace 以及本文提出的两种采样方法 Sampleplus 和 SampleSpaceplus 分别生成 100 个 PDF 对象，再使用标准的 PDF 文件更新方法和 Fuzz&Learn 的对象附加方法，将 4 组 100 个 PDF 对象分别与 10 个 PDF 文件相互组合以获得 4 组 1 000 个新的 PDF 文件，分别表示为 Sample，SampleSpace，Sampleplus，SampleSpaceplus。最后分别获取这 5 组 PDF 文件的覆盖率，结果如图 6 所示。可以观察到以下结果：除了 10 epoch 和 50 epoch 时使用 SampleSpace 和 SampleSpaceplus 采样方式获得的 PDF 文件覆盖率低于基线覆盖率，其余都显著高于基线覆盖率。对于所有 epoch，使用 Sampleplus 和 SampleSpaceplus 采样方法获得的 PDF 文件覆盖率都等于或高于 Sample 和 SampleSpace 采样方法。所有采样方法，都在 20 epoch 到 30 epoch 获得了最高覆盖率。其中，Sampleplus 采样方法在模型训练 30 个 epoch 时获得的覆盖率最高。

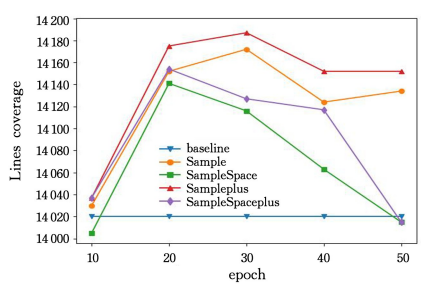


图 6　4 种采样方法生成的测试用例的行覆盖数

Fig. 6　Lines coverage of test cases generated by four sampling methods

可以得出结论：使用采样方法确实能有效地提高生成的PDF对象的多样性，进而提高PDF文件覆盖率。本文新提出的两种采样方法都明显优于之前的采样方法，其中Sample-plus采样方法效果最好。另外，并非训练的epoch越多越好，模型训练20到30个epoch时，能取得最好的训练效果，再继续训练时，训练效果反而开始下降。

4.5.2 对象附加方法和PDF更新方法的有效性

为了验证本文提出的对象附加方法AOU和PDF更新方法In-place Update的有效性，使用Seq2Seq模型生成1 000个PDF对象，然后分别使用Learn&Fuzz的对象附加方法、IUST-DeepFuzz中的两种对象附加方法SOU和MOU、本文提出的AOU对象附加方法及标准的PDF更新方法[21]将1000个PDF对象附加到每个host文件中，各生成1 000个PDF文件，分别表示为Learn&Fuzz，SOU，MOU和AOU。另外，同样使用AOU对象附加方法和In-place PDF文件更新方法将1 000个PDF对象附加到每个host文件中，各生成1 000个PDF文件，表示为AOU+In-place。

然后，本文获取了上述5组PDF文件相较于host文件的代码覆盖率提升情况和PDF文件平均大小增长情况，如表3和表4所列。从表中可以明显看出，Learn&Fuzz和SOU的代码覆盖率增长和文件大小平均增长都最低，当PDF文件中最后一个对象即是id最大的对象时，Learn&Fuzz对象附加方法实际就与SOU相同。AOU+In-place在host_min和host_med上获得了最大的代码覆盖率提升；AOU在这两个主机上也获得了第二大的覆盖率提升。此外，除了AOU在host_min上的平均文件大小增长略高于MOU外，AOU+In-place和AOU在host_min和host_med上的平均文件大小增长都远远小于MOU。而MOU在host_max上覆盖率的提升略高于AOU+In-place和AOU，但文件平均增长远远高于二者。

表3 对不同host文件使用不同对象附加方式和文件更新方式生成的PDF文件的代码覆盖增长比例

Table 3 Code coverage growth rate of PDF files generated using different object attachment methods and file update methods for different host files

(%)

Hosts	Learn&Fuzz	SOU	MOU	AOU	AOU+In-place
host_max	+4.04	+4.04	+57.74	+53.78	+55.29
host_med	+5.29	+5.29	+36.97	+41.42	+44.32
host_min	+2.46	+2.94	+61.98	+69.49	+78.28

表4 对不同host文件使用不同对象附加方式和文件更新方式生成的PDF文件的大小平均增长比例

Table 4 Average growth rate of PDF file size generated using different object attachment methods and file update methods for different host files

(%)

Hosts	Learn&Fuzz	SOU	MOU	AOU	AOU+In-place
host_max	+0.31	+0.31	+81.41	+50.18	+11.93
host_med	+0.16	+0.16	+57.27	+60.40	+25.19
host_min	+27.69	+27.69	+400.96	+258.54	+238.82

为了验证哪种方法能在覆盖率增长和文件大小增长之间取得最佳平衡关系，使用AFL++分别对使用AOU，MOU和AOU+In-place在host_max上生成的1 000个PDF文件进行24 h的模糊测试，结果如图7所示。可以观察到，尽管MOU生成的PDF文件的初始覆盖率较高，但经过24 h的模糊测试后，AOU代码覆盖率更高，而AOU+In-place的代码覆盖率最高，这表明AOU模式的覆盖率增长和文件大小增长关系比MOU模式更平衡，而In-place更新方式的覆盖率增长和文件大小增长关系比标准增量更新方式更平衡。

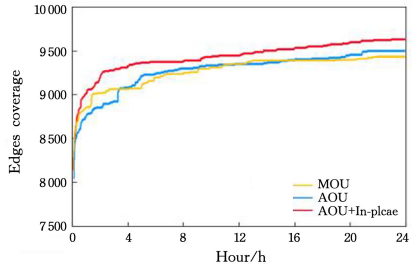


图7 AOU+In-place，AOU和MOU生成的PDF文件24 h模糊测试的代码覆盖增长情况

Fig. 7 Code coverage growth for 24-hour fuzzing test of PDF files generated by AOU+In-place, AOU and MOU

4.5.3 高效变异方法的有效性

为了验证本文提出的新的变异方法OptMutation的有效性，首先随机在数据集中抽取了100个PDF文件，采用模糊系数为10 000的Random算法，对这100个PDF文件，每个生成5个变体，生成5组100个随机变异PDF文件，表示为Random-1到Random-5。然后将这100个PDF文件输入已训练好的Transformer模型，变异阈值k设为0.999 99，得到100个经过有效变异的PDF文件，表示为OptMutation。图8给出了6组PDF文件的覆盖率，可以明显看出采用OptMutation变异获得的PDF文件覆盖率高于所有随机变异生成的PDF文件的覆盖率；另外，随机变异生成的PDF文件的覆盖率波动明显，说明随机变异对提高PDF文件覆盖率效果较差，且很不稳定。

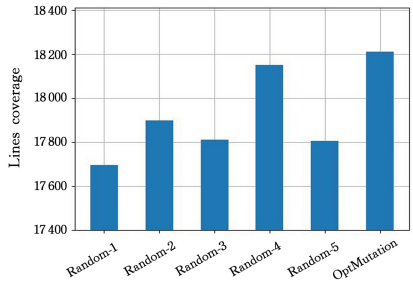


图8 Random和OptMutation变异方法生成PDF文件的边覆盖数

Fig. 8 Lines coverage of PDF files generated by Random and OptMutation

综上所述，本RQ中重点讨论了DeepGenFuzz中各项策略的有效性，通过实验，证明了DeepGenFuzz中各项策略都是有效的，都有助于提高整体方法的质量。

4.6 对RQ3的调查

在这个RQ中，研究了DeepGenFuzz的效率，主要体现为

DeepGenFuzz 的主要步骤所花费的时间。

图 9 显示了 5 个主要步骤的时间分布。具体来说，图中每个箱线图对应于图 2 中的 5 个主要步骤执行 1 000 个对象的时间，分别为筛选 1 000 个 PDF 文件、生成 1 000 条文本数据、生成 1 000 条二进制数据、对 1 000 个 PDF 文件附加新对象、对 1 000 个 PDF 文件进行有效变异。从图中可以看出，二进制数据生成和有效变异是耗时最长的两个步骤，平均值分别为 7 619 s 和 5 042 s。

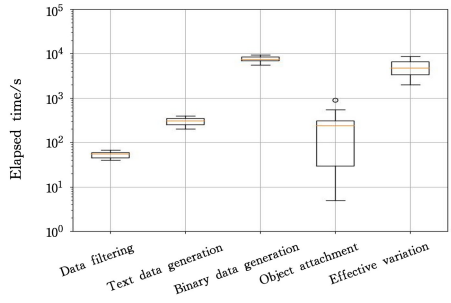


图 9 DeepGenFuzz5 个主要步骤消耗的时间

Fig. 9 Time consumed by five main steps of DeepGenFuzz

对于二进制数据生成来说，因为生成的二进制数据可以重复使用，所以这个时间是可以接受的。对有效变异来说，消耗的时间虽然较长，但考虑到其他步骤消耗时间较短，总体方法时间消耗仍低于目前最先进的方法 IUST-DeepFuzz[17] 的时间消耗，因此本文认为这也是可以接受的；并且通过对算法的进一步优化和硬件设备的提升，二进制数据生成和有效变异消耗的时间仍可以大幅降低。

基于语法的模糊测试查找对具有复杂结构化输入的应用程序的漏洞是十分有效的，但要生成这些复杂的结构化输入，往往需要手工总结编写复杂的语法规则。本文参考了最近的一些通过深度学习从大量数据集中自动学习复杂结构化输入的工作，然而它们存在一些普遍的缺陷，导致生成的测试用例质量差，且查找 bug 能力弱。

为了解决这些问题，本文提出了一个基于深度学习的高效 PDF 应用模糊测试用例生成框架 DeepGenFuzz，利用 CNN，Seq2Seq 和 Transformer 等模型，通过数据筛选、对象生成、对象附加、高效变异等步骤生成高质量 PDF 测试用例。通过这种改进的方法，可以自动化地生成更有效的复杂结构化输入，从而提高模糊测试的效果。评估表明，DeepGenFuzz 生成的测试用例平均代码覆盖明显高于 Learn&Fuzz 和 IUST-DeepFuzz 等最先进的工具，最高可达 8.12%～61.03%；bug 查找能力也远远优于 Learn&Fuzz 和 IUST-DeepFuzz 等最先进的工具，本文已经报告了在 7 个最流行的 PDF 应用程序中发现的 31 个未曾被报告的 bug，其中 25 个已经得到确认或修复，且涵盖了所有被测程序。

本文的工作仍有一些可以改进的方向。一方面，可以引入其他广泛应用于模糊测试的技术，如集成学习[30]、强化学习[31]等来进一步优化测试用例生成过程。另一方面，可以将本文的方法应用于 PDF 格式以外的具有复杂结构的输入格式。

参 考 文 献

[1] SMUTZ C，STAVROU A. Malicious PDF detection using metadata and structural features[C]// Proceedings of the 28th Annual Computer Security Applications Conference. 2012：239-248.

[2] VANIEA K E，RADER E，WASH R. Betrayed by updates：how negative experiences affect future security[C]// Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2014：2671-2674.

[3] KUCHTA T，LUTELLIER T，WONG E，et al. On the correctness of electronic documents：studying，finding，and localizing inconsistency bugs in PDF readers and files[J]. Empirical Software Engineering，2018，23：3187-3220.

[4] GANESH V，LEEK T，RINARD M. Taint-based directed whitebox fuzzing[C]// 2009 IEEE 31st International Conference on Software Engineering. IEEE，2009：474-484.

[5] PHAM V T，BÖHME M，ROYCHOUDHURY A. Model-based whitebox fuzzing for program binaries[C]// Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016：543-553.

[6] CHA S K，WOO M，BRUMLEY D. Program-adaptive mutational fuzzing[C]// 2015 IEEE Symposium on Security and Privacy. IEEE，2015：725-741.

[7] KOIKE Y，KATSURA H，YAKURA H，et al. SLOPT：Bandit Optimization Framework for Mutation-Based Fuzzing[C]// Proceedings of the 38th Annual Computer Security Applications Conference. 2022：519-533.

[8] SHE D，KRISHNA R，YAN L，et al. MTFuzz：fuzzing with a multi-task neural network[C]// Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2020：737-749.

[9] WU M，JIANG L，XIANG J，et al. Evaluating and improving neural program-smoothing-based fuzzing[C]// Proceedings of the 44th International Conference on Software Engineering. 2022：847-858.

[10] SHE D，PEI K，EPSTEIN D，et al. Neuzz：Efficient fuzzing with neural program smoothing[C]// 2019 IEEE Symposium on Security and Privacy(SP). IEEE，2019：803-817.

[11] BA J，DUCK G J，ROYCHOUDHURY A. Efficient Greybox Fuzzing to Detect Memory Errors[C]// Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. 2022：1-12.

[12] LI Y，XUE Y，CHEN H，et al. Cerebro：context-aware adaptive fuzzing for effective vulnerability detection[C]// Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019：533-544.

[13] PHAM V T. AFLSmart++：Smarter Greybox Fuzzing[C]// 2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing(SBFT). IEEE，2023：76-79.

[14] DINH S T, CHO H, MARTIN K, et al. Favocado: Fuzzing the Binding Code of JavaScript Engines Using Semantically Correct Test Cases[C]// NDSS. 2021.

[15] GODEFROID P, PELEG H, SINGH R. Learn&Fuzz: Machine learning for input fuzzing[C]// 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2017: 50-59.

[16] JITSUNARI Y, ARAHORI Y. Coverage-guided learning-assisted grammar-based fuzzing[C]// 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2019: 275-280.

[17] ZAKERI NASRABADI M, PARSA S, KALAEE A. Format-aware Learn&Fuzz: deep test data generation for efficient fuzzing [J]. Neural Computing and Applications, 2021, 33: 1497-1513.

[18] KHAN M E. Different approaches to white box testing technique for finding errors[J]. International Journal of Software Engineering and Its Applications, 2011, 5(3): 1-14.

[19] ACHARYA S, PANDYA V. Bridge between black box and white box-gray box testing technique[J]. International Journal of Electronics and Computer Science Engineering, 2012, 2(1): 175-185.

[20] KHAN M E, KHAN F. A comparative study of white box, black box and grey box testing techniques[J]. International Journal of Advanced Computer Science and Applications, 2012, 3(6): 12-25.

[21] Adobe Systems Incorporated. PDF Reference (Version 1.7) Nov. 2006 [OL]. https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.7old.pdf.

[22] PANG C, LIU H, WANG Y, et al. Generation-based fuzzing? Don't build a new generator, reuse! [J]. Computers & Security, 2023, 129: 103178.

[23] REBERT A, CHA S K, AVGERINOS T, et al. Optimizing seed selection for fuzzing[C]// 23rd USENIX Security Symposium (USENIX Security 14). 2014: 861-875.

[24] ANDERSSON C, RUNESON P. A replicated quantitative analysis of fault distributions in complex software systems[J]. IEEE Transactions on Software Engineering, 2007, 33(5): 273-286.

[25] FENTON N E, OHLSSON N. Quantitative analysis of faults and failures in a complex software system[J]. IEEE Transactions on Software engineering, 2000, 26(8): 797-814.

[26] ZHANG H. On the distribution of software faults[J]. IEEE Transactions on Software Engineering, 2008, 34(2): 301-302.

[27] MURPHY-HILL E, ZIMMERMANN T, BIRD C, et al. The design of bug fixes[C]// 2013 35th International Conference on Software Engineering (ICSE). IEEE, 2013: 332-341.

[28] WILLIAMS C, SPACCO J. Szz revisited: verifying when changes induce fixes[C]// Proceedings of the 2008 Workshop on Defects in Large Software Systems. 2008: 32-36.

[29] ZONG P, LV T, WANG D, et al. {FuzzGuard}: Filtering out unreachable inputs in directed grey-box fuzzing through deep learning[C]// 29th USENIX Security Symposium (USENIX Security 20). 2020: 2255-2269.

[30] DIETTERICH T G. Ensemble learning[J]. The Handbook of Brain Theory and Neural Networks, 2002, 2(1): 110-125.

[31] KAELBLING L P, LITTMAN M L, MOORE A W. Reinforcement learning: A survey[J]. Journal of Artificial Intelligence Research, 1996, 4: 237-285.

LIU Jiahao, born in 1999, postgraduate. His main research interests include software testing and deep learning.

JIANG He, born in 1980, professor, Ph.D supervisor, is a member of CCF (No. 08846D). His main research interests include system software and intelligent software engineering.

(责任编辑: 柯颖)