Artificial Intelligence and Robotics Research 人工智能与机器人研究, 2025, 14(2), 341-349
Published Online March 2025 in Hans. https://www.hanspub.org/journal/airr
https://doi.org/10.12677/airr.2025.142034

Hans汉斯

基于机器学习的模糊测试系统

赵浩东，康晓凤，李　慧，陈　聪，袁杨坤，刘子豪

徐州工程学院信息工程学院(大数据学院)，江苏　徐州

收稿日期：2025年2月14日；录用日期：2025年3月13日；发布日期：2025年3月21日

摘　要

随着信息技术的发展和漏洞类型日益复杂化，如何更高效、简洁、有序地测试软件漏洞，辅助软件开发人员更好地开发软件成为网络安全领域的重要研究方向。本文提出了一种基于机器学习的模糊测试系统，涵盖自动模糊测试监视、待测程序管理、自主选择更优变异策略、自动生成程序测试报告等模块。系统采用了机器学习技术，对Havoc变异算法变异生成的种子进行进一步筛选变异，提高对程序漏洞检测的效率与覆盖率。实验表明，该系统能在模糊测试中生成多样、有效的变异种子，适用于复杂的程序漏洞检测。

关键词

漏洞检测，模糊测试，Havoc变异算法，网络安全

A Machine Learning-Based Fuzzing System

Haodong Zhao, Xiaofeng Kang, Hui Li, Cong Chen, Yangkun Yuan, Zihao Liu

College of Information Engineering (Big Data College), Xuzhou University of Technology, Xuzhou Jiangsu

Received: Feb. 14th, 2025; accepted: Mar. 13th, 2025; published: Mar. 21st, 2025

Abstract

With the advancement of information technology and the increasing complexity of vulnerability types, how to more efficiently, succinctly, and systematically test software vulnerabilities, thereby assisting software developers in improving their development processes, has become a significant research focus in the field of cybersecurity. This paper introduces a machine learning-based fuzzing system that includes modules for automatic fuzzing monitoring, management of the program under test, autonomous selection of optimal mutation strategies, and automatic generation of program testing reports. The system employs machine learning techniques and utilizes the Havoc mutation algorithm to mutate and expand seeds of input programs, enhancing the efficiency and coverage of program vulnerability detection. Experimental results show that this system can generate diverse

文章引用: 赵浩东, 康晓凤, 李慧, 陈聪, 袁杨坤, 刘子豪. 基于机器学习的模糊测试系统[J]. 人工智能与机器人研究, 2025, 14(2): 341-349. DOI: 10.12677/airr.2025.142034

and effective mutation seeds during fuzzing tests, making it suitable for detecting complex program vulnerabilities.

Keywords

Vulnerability Detection, Fuzzing, Havoc Mutation Algorithm, Cybersecurity

1. 引言

模糊测试是当今比较流行的漏洞挖掘技术之一，广泛应用于软件安全测试领域。然而，传统的模糊测试手段存在诸多局限性，例如测试数据多样性不足、测试用例接收率低、代码覆盖率低等问题，导致其在实际应用中难以高效发现复杂的软件漏洞[1]。近年来，随着机器学习技术的快速发展，研究者开始探索将机器学习引入模糊测试中，以解决传统方法的不足。机器学习技术能够对模糊测试过程中生成的海量测试样本和其他数据进行智能化处理和分析，从中学习测试用例的潜在特征，并挖掘更深层的结构信息和语义特征[2]。这些特征相较于传统基于经验的特征更加充分和有效，能够显著提升测试用例的质量和多样性。

其中 Rajpal [3]等人利用 LSTM 模型预测输入文件中不同位置与新代码覆盖率的相关性，从而指导变异过程，减少无效输入。She [4]等人则通过神经网络模型和增量学习技术，近似模拟程序分支行为，利用函数梯度引导变异过程，显著提高了测试效率。而本系统则利用 havoc 算法生成测试用例范围广的特点，结合有监督机器学习模型优化了种子变异筛选调度的过程，使测试效率进一步提高。

2. 技术特点

2.1. 机器学习在测试用例生成与变异中的应用特点介绍与分析

机器学习技术通过对模糊测试中海量的测试样本和其他数据进行处理、分析，来学习测试用例中的潜在特征，并用于指导种子生成过程[5]-[7]。接下来将于 3 个方面来详细分析机器学习在模糊测试中的应用。

(1) 各种软件的测试用例的生成

机器学习可以对不同软件进行针对性的测试用例生成，其中包括文件解析软件测试用例的生成，网络协议的测试用例生成，代码解析工具的测试用例生成。文件解析软件的测试用例设计：针对各类文件格式，通常需要借助专门的解析工具进行打开和操作。若这些解析工具存在缺陷或漏洞，可能会导致文件信息解析错误，进而影响文件的正常处理，严重时甚至可能引发安全风险。网络协议的测试用例设计：为了实现实时且可控的通信，许多协议在设计时并未采用加密机制，且不同协议的格式各异，需要借助特定的解析工具来处理协议报文。开发人员在实现和解析这些网络协议时，可能会引入逻辑错误，从而对网络系统的安全性和稳定性构成严重威胁。代码解析工具的测试用例生成：编译器和解释器是较为重要的计算机系统软件，它们仍然存在一些缺陷，可能导致意外的程序执行，甚至影响应用的安全性。

(2) 在测试用例变异中的作用

应用神经网络模型和增量学习技术来学习复杂的程序分支行为，并平滑近似程序分支，获得与真实

程序行为近似的程序模型。基于训练得到的近似程序，利用函数梯度来引导模糊测试输入用例的变异过程，以显著提高模糊测试的效率。

(3) 在种子调度与筛选中的应用

通过代码可达性和动态分析提取用于学习的特征，并从相同或相似程序上做出的种子调度决策中学习知识，同时基于这些知识评估每个选定的种子的模糊测试性能，从而确定哪些新种子将产生更好的模糊测试效果。

2.2. Havoc 种子变异策略特点分析

Havoc 变异策略旨在通过广泛的随机化变异来发现软件中的潜在漏洞，特别是在经过初步的确定性变异未能找到足够多的新路径后。这种策略虽然不如确定性策略那样有针对性，但它能够有效地覆盖那些不常见的代码路径，并且在长时间运行时能够提供持续的价值。

(1) 高度随机性：与早期确定性的变异策略(如位翻转、字节翻转等)不同，Havoc 采取了一种更加随机化的策略来对输入文件进行变异。这意味着它能够探索更广阔的输入空间，可能会检查到一些通过确定性策略难以达到的问题。

(2) 广泛变异：Havoc 实施多种变异操作，涵盖但不限于随机数据插入、数据块删除、数据块复制以及数据块位置交换等。通过这些变异手段，原始文件往往会经历显著的改动，生成与原文件差异极大的新文件，几乎达到“焕然一新”的效果。

(3) 组合变异：Havoc 不仅单独应用各种变异操作，还会将它们组合起来使用。例如，一个变异循环可能包含多次随机选择的变异操作，使得最终生成的测试用例更加多样化。

(4) 效率优化：尽管 Havoc 变异具有很大的随机性和破坏性，但为了提高效率，它也会利用之前收集的信息来指导变异过程。例如，如果某些变异操作倾向于产生更有价值的测试用例(即导致程序执行新路径或崩溃)，那么这些变异可能会被优先考虑。

(5) 拼接策略(Splice)：虽然严格意义上不属于 Havoc 的一部分，但是 splice 策略经常和 Havoc 一起提及。Splice 策略通过组合两个不同输入文件的部分内容来创建一个新的测试用例。这种方法旨在探索那些已知能够触发程序行为变化的输入之间的新边界条件，从而有助于发现潜在的漏洞或异常行为。

将 havoc 集成到此模糊测试系统有两种方法：

(1) 顺序方法

在主模糊测试策略变异阶段的后期进行 Havoc，原生 AFL/AFL++便采取这种方式进行集成。

(2) 并行方法

Havoc 和主模糊测试策略并行执行，如 QSYM 支持的三个线程，这三个线程分别执行 Havoc、AFL 确定性变异策略和符号执行。

2.3. Havoc 算法

位翻转：随机选择输入数据中的某些比特位并进行翻转(0 变 1 或 1 变 0)。

```
def bit_flip(data):
idx = random.randint(0, len(data) * 8 - 1)
return data[:idx//8] + bytes([data[idx//8] ^ (1 << (idx % 8))]) + data[idx//8 + 1:]
```

字节插入/删除：在输入数据中随机位置插入或删除一个或多个字节。

```
def byte_insert(data):
pos = random.randint(0, len(data))
return data[:pos] + bytes([random.randint(0, 255)]) + data[pos:]
```

数据块操作：从输入数据中随机选取一段子数据，对其进行复制或者删除操作。

随机融合：选取两个不同的输入样本，将它们各自的一部分截取并组合在一起，创造出一个新的输入样本。

数值调整：修改输入数据中的整数或浮点数值，例如加减固定值或乘除固定倍数。

随机替换：通过生成随机数据，对输入数据中的某些部分进行替换。

3. 系统设计与实现

3.1. Havoc 算法设计

Havoc 变异策略主要由变异次数和变异器叠加(mutator stacking)次数决定。如图 1 所示，Havoc 变异策略首先会基于实时种子信息(如种子执行时间、种子覆盖到的比特位图大小等)确定种子的变异次数(对应于 stage_max 变量值)，即模糊器需要执行多少次的 Havoc 操作(一次 Havoc 操作是指从一个特定的种子，经过一系列变异后得到一个测试用例的过程)；在确定了变异次数之后，Havoc 在每次 Havoc 操作之前确定一个变异器叠加次数(对应于 use_stacking 变量值)，并根据该次数按序随机应用多种变异方法。在变异器叠加阶段，Havoc 首先根据变异器叠加次数对一个种子应用多种变异操作。具体来说，Havoc 首先确定应用的变异器叠加大小，其值为 2 到 128 之间任一 2 的幂值；然后 Havoc 为每个叠加随机选择变异器并进行变异操作。最后，将所有堆叠的变异操作应用于种子以生成一个新的测试用例。

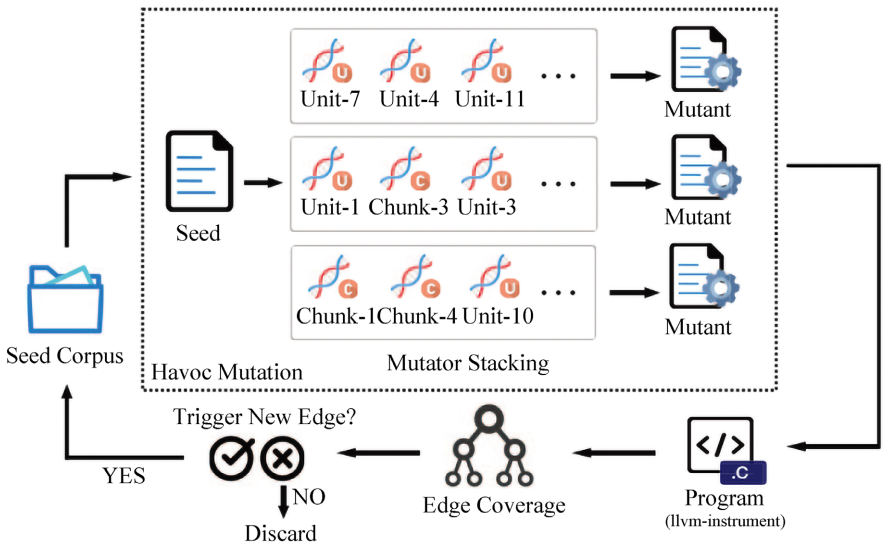

Figure 1. Havoc algorithm schematic diagram
图 1. Havoc 算法示意图

3.2. 机器学习在模糊测试中的设计

机器学习在模糊测试中的应用旨在通过智能化手段优化模糊测试过程，提高漏洞检测效率和准确性。具体步骤如下：

① 用户请求启动模糊测试

用户通过指定的界面或 API 向模糊测试框架提交待测应用程序及其配置参数，包括但不限于目标程序路径、输入类型等。这个过程类似于客户端请求上传文件到分布式存储系统。

② 模型分配变异策略并初始化种子队列

基于机器学习的模糊测试工具首先评估待测应用程序的特点，并根据这些信息选择合适的变异策略(如基于语法的变异、随机变异等)。同时，该工具会初始化一个种子队列，包含初始测试用例集合，用于后续变异操作。这一步骤与 NameNode 分配块大小及确定 DataNode 列表类似，其中模型扮演了“决策者”的角色，负责制定测试计划。

③ 执行变异和测试

模糊测试框架按照选定的策略对种子队列中的测试用例进行变异，并将生成的新测试用例应用于目标程序中。此过程中，机器学习模型可能还会参与到实时监控和调整变异策略中，以适应不同阶段的测试需求，类似于数据流从客户端流向 DataNode 的过程。

④ 反馈机制确认测试结果

当一次变异后的测试用例被执行后，系统会收集关于崩溃、内存泄漏、性能瓶颈等异常情况的信息，并通过反馈机制更新机器学习模型的知识库。如果发现新的漏洞或者增加了代码覆盖率，相应的测试用例会被标记为成功，并通知用户。这一步骤类似于 DataNode 确认接收数据块并向 NameNode 报告状态。

⑤ 更新模型和策略

基于上一阶段收集到的数据和反馈，机器学习模型将自我更新，调整其内部参数和预测规则，以便在未来更准确地指导模糊测试流程。此外，任何新发现的有效变异策略也会被纳入考虑范围，进一步丰富种子队列。这与 NameNode 在接收到所有数据块的成功确认后更新元数据相呼应，确保整个模糊测试过程能够持续改进和发展。

3.3. 测试结果图表显示流程设计

(1) 测试结果收集与处理

本部分主要负责从模糊测试工具中收集测试结果，并通过图表形式展示给用户。首先，通过调用模糊测试系统的 API 或命令行接口获取目标程序在不同测试阶段的性能指标和崩溃信息。函数 run_afl_test(target)封装了执行模糊测试的完整流程，包括初始化测试环境、运行模糊测试系统实例以及收集测试数据等步骤。使用 subprocess.Popen 模块进行调用，并将测试结果存储于本地文件系统中。

程序内嵌了一套完整的校验机制，用于检查模糊测试系统路径的有效性及测试执行的状态。当检测到配置错误或执行异常时，系统能够实时输出详细的错误日志，帮助用户迅速定位问题所在。此外，为了提高数据处理效率，我们引入了缓存机制来暂存中间结果，减少重复计算带来的资源消耗。

(2) 数据解析与图表生成

在收集到足够的测试数据后，下一步是对这些数据进行解析并转换为适合图表展示的格式。利用 Python 中的 pandas 库对原始测试数据进行清洗和整理，提取出关键指标，如发现的漏洞数量、每次迭代发现的新路径数、崩溃发生的频率等。这些经过处理的数据被导入至 ECharts 或 Plotly 等图表库中，以生成直观的可视化图表。

例如，通过调用 generate_crash_trend_chart(data)函数，可以创建一个展示随时间变化的崩溃趋势图。该函数接收经过预处理的数据作为输入，自动计算每日新增崩溃次数，并以折线图的形式展示出来。类似地，其他类型的图表，如饼图表示不同类型漏洞的比例分布，柱状图展示每个测试阶段发现的漏洞数量等，也可以根据实际需求灵活生成。

(3) 图表集成与展示

最后一步是将生成的图表集成到系统的前端界面中。前端采用 Vue.js 框架结合 Element-UI 组件库构建，提供了友好的用户体验。如图 2 所示，通过前后端交互接口(RESTful API)，前端页面能够动态加载由后端生成的图表数据，并实时更新显示内容。用户可以通过点击不同的导航选项，在同一页面上切换

查看各种图表，如崩溃趋势图、漏洞类型分布图等，从而全面了解测试结果。

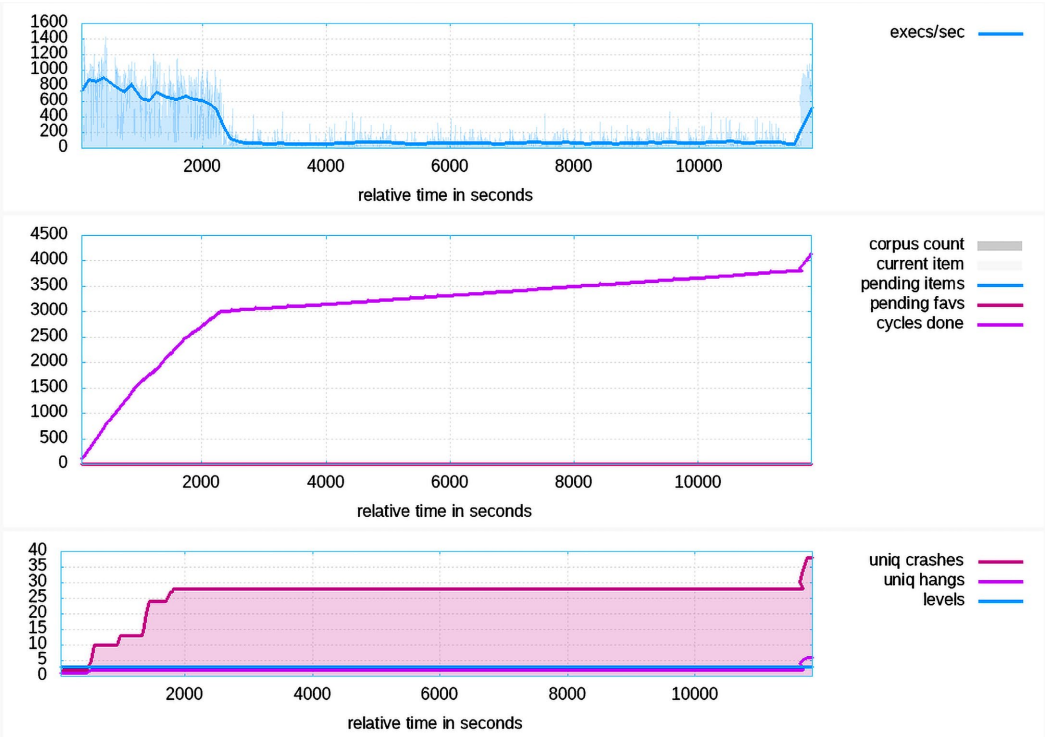

Figure 2. Chart generation results
图 2. 图表生成结果

3.4. 测试监视界面流程设计

(1) 测试过程监控数据收集

本部分主要负责实时收集模糊测试过程中产生的各类监控数据，并为用户提供一个直观的监视界面。首先，通过调用模糊测试提供的监控接口或直接读取其生成的日志文件，获取目标程序在不同测试阶段的关键性能指标(KPIs)，如每秒执行的测试案例数量(execs/sec)、发现的新路径数量、崩溃发生的频率等。函数 collect_monitor_data(target)封装了这一过程，包括初始化监控环境、设置日志文件路径以及定时采集数据等步骤。模糊测试的数据采集使用 subprocess.Popen 模块来调用命令行工具，并将结果存储于本地数据库中以便后续处理。

系统内部集成了一套完善的验证机制，用于保障监控数据的精确性和完整性。举例来说，在每个数据采集周期完成后，系统会自动进行数据校验，检查是否存在数据缺失或异常情况。一旦发现任何异常，系统会迅速生成错误日志并尝试重新获取数据，以确保监控界面始终呈现最新且可信的信息。

(2) 数据处理与格式转换

在收集到实时监控数据后，接下来需要对这些原始数据进行预处理和格式转换，以便于前端界面展示。利用 Python 中的 pandas 库对原始监控数据进行清洗和整理，提取出关键指标，并计算一些衍生指标，如平均执行速度、累计发现的漏洞数等。此外，为了提高数据处理效率，我们引入了批处理机制，即每隔一定时间间隔批量处理一次新收集的数据，减少频繁的数据处理带来的资源消耗。

经过预处理后的数据会被转换成 JSON 格式，这是一种轻量级的数据交换格式，非常适合前后端通信。每个监控指标都被组织成易于解析的数据结构，便于前端页面动态加载和更新。

(3) 监视界面构建与交互

最后一步是将处理好的监控数据集成到系统的前端监视界面上。前端采用 Vue.js 框架结合 Element-UI 组件库构建，提供了丰富的可视化组件和友好的用户交互体验。通过 RESTful API，前端页面能够实时请求由后端生成的监控数据，并动态更新显示内容。如图 3 所示。

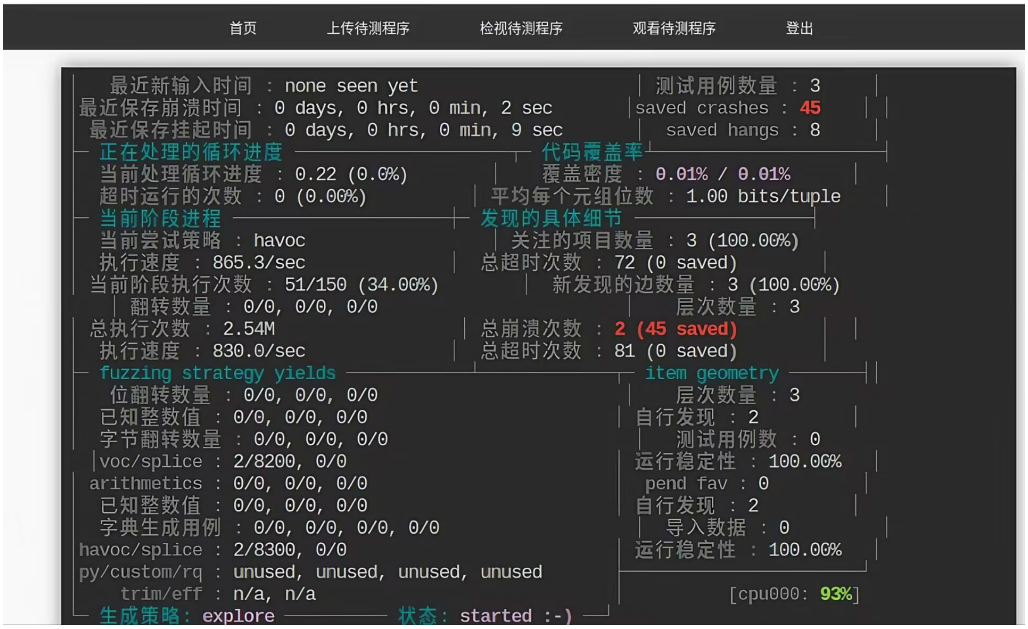

Figure 3. Test program status display interface
图 3. 测试程序状态显示界面

3.5. 性能比较

为了展示基于机器学习的模糊测试系统相较于传统 AFL (American Fuzzy Lop)的优势，我们对两者进行了性能对比测试。测试目标为同一开源软件项目(LibTIFF)，测试时间为 2 小时，测试环境保持一致。以下是具体的测试结果对比：

(1) 代码覆盖率

基于机器学习的模糊测试系统在测试期间达到了 78%的代码覆盖率，而传统 AFL 的代码覆盖率为 62%。这表明机器学习系统能够更有效地探索程序的执行路径，覆盖更多的代码分支。

(2) 发现的 Crash 数量

在测试过程中，基于机器学习的系统共发现了 42 个 Crash，而传统 AFL 仅发现了 25 个。机器学习系统通过智能化的变异策略和反馈机制，能够更高效地生成触发异常的测试用例，从而发现更多的潜在漏洞，性能比较如图 4 所示。

(3) 有效测试用例生成率

机器学习系统生成的测试用例中，能够触发新路径或异常的用例占比为 15%，而传统 AFL 的这一比例仅为 8%。这说明机器学习系统生成的测试用例质量更高，减少了无效用例的生成，提升了测试效率。

(4) 边缘覆盖率提升

在边缘覆盖率(Edge Coverage)方面，机器学习系统提升了 12%，而传统 AFL 仅提升了 7%。这表明机器学习系统能够更精准地指导变异过程，探索程序的边缘路径，从而提高漏洞挖掘的深度，性能比较如图 5 所示。

综上所述，基于机器学习的模糊测试系统在代码覆盖率、Crash 发现数量、有效测试用例生成率以及边缘覆盖率等方面均显著优于传统 AFL。这些结果表明，机器学习技术的引入能够显著提升模糊测试的效率和效果，为软件漏洞挖掘提供了更强大的工具支持。

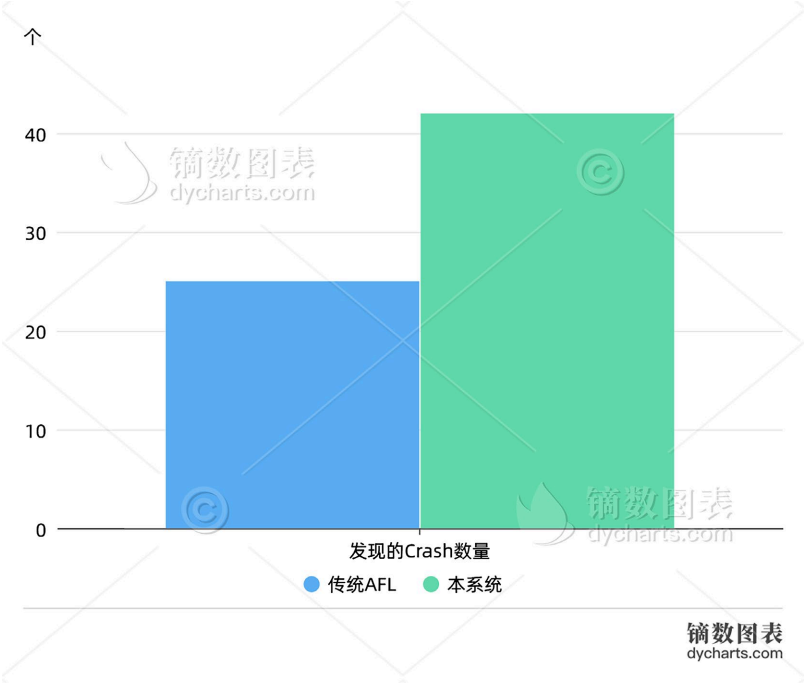

Figure 4. Performance comparison of discovered Crashes
图 4. 发现的 Crash 数量性能比较

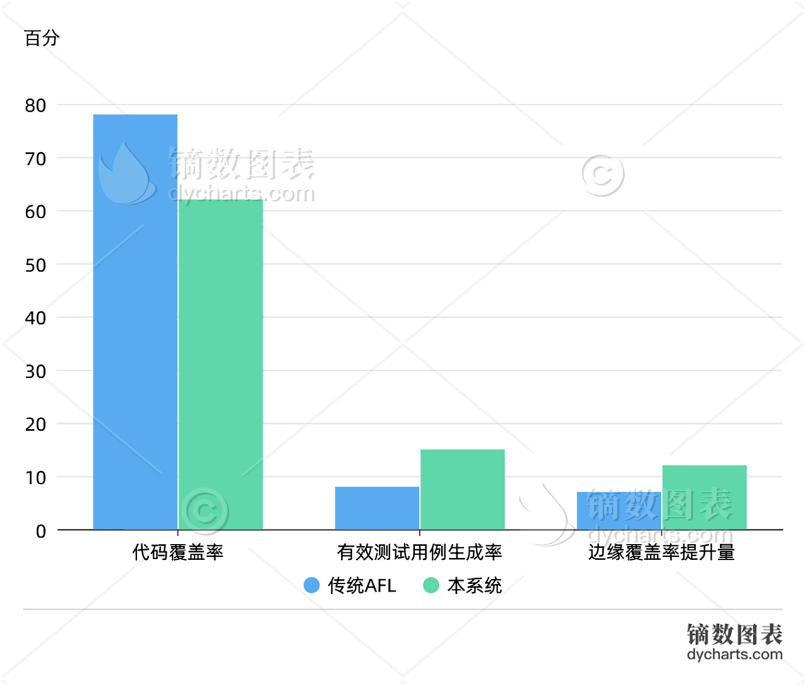

Figure 5. Performance comparison of code coverage
图 5. 代码覆盖率等性能比较

4. 结论

本文设计并实现了一个基于机器学习的模糊测试系统，旨在提高软件漏洞检测的效率和准确性。通过结合先进的 Havoc 变异策略与智能化算法，该系统能够更深入地探索程序的不同执行路径，有效地发现潜在的安全漏洞。系统的核心优势在于其智能化的变异策略分配机制，利用机器学习模型根据待测应用程序的特点选择最优变异策略，并通过反馈机制不断优化测试过程。此外，系统还进行了数据集构建、深层次信息提取以及跨项目应用中迁移学习的研究，为解决数据不平衡问题、挖掘复杂的多点触发型漏洞提供了新思路。

实验结果表明，基于机器学习的模糊测试系统能够在较短时间内覆盖更多的代码路径，显著提高了漏洞检测率，有效减少了误报情况[8] [9]。特别是机器学习的应用使得系统能够适应多样化的编程语言和软件架构，增强了系统的通用性和灵活性。本系统提供了一种高效、智能且易于扩展的软件安全测试解决方案，具有广泛的应用前景。它不仅有助于提高现有软件的安全性，也为未来的软件开发和维护工作提供了强有力的支持，对于保障大规模软件环境下的安全性具有重要意义。

基金项目

本文是江苏省大学生创新创业计划项目(xcx2024180)，徐州工程学院大学生创新创业项目(xcx2024190)的阶段性成果之一。

参考文献

[1] 张雄, 李舟军. 模糊测试技术研究综述[J]. 计算机科学, 2016, 43(5): 1-8+26.

[2] 王鹃, 张冲, 龚家新, 等. 基于机器学习的模糊测试研究综述[J]. 信息网络安全, 2023, 23(8): 1-16.

[3] Rajpal, M., Blum, W. and Singh, R. Not All Bytes Are Equal: Neural Byte Sieve for Fuzzing. https://arxiv.org/abs/1711.04596

[4] She, D.D., Pei, K.X., Epstein, D., *et al*. (2019) Neuzz: Efficient Fuzzing with Neural Program Smoothing. *IEEE Symposium on Security and Privacy* (*SP*), New York, 19-23 May 2019, 803-817. https://doi.org/10.1109/SP.2019.00052

[5] 任泽众, 郑晗, 张嘉元, 等. 模糊测试技术综述[J]. 计算机研究与发展, 2021, 58(5): 944-963.

[6] 陈晓东, 刘洋. 模糊测试中的机器学习应用综述[J]. 软件学报, 2021, 32(5): 1234-1250.

[7] 李明, 王伟. 基于深度学习的模糊测试技术研究进展[J]. 计算机工程与应用, 2022, 58(3): 1-10.

[8] 赵亮, 王鹏. 基于强化学习的模糊测试优化方法[J]. 计算机研究与发展, 2022, 59(4): 789-800.

[9] 邹权臣, 张涛, 吴润浦, 等. 从自动化到智能化: 软件漏洞挖掘技术进展[J]. 清华大学学报(自然科学版), 2018, 58(12): 1079-1094.