Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

大模型协同的软件模糊测试技术

博士研究生 张浩然

2026年03月29日

问题回溯

- 总结反思
  - 讲解语气过于平淡
  - 算法选取较为简单

- 相关内容
  - 2026.03.01 徐菊彬《协议模糊测试方法》
  - 2025.06.29 张浩然《软件灰盒定向模糊测试技术》
  - 2025.06.22 杨语航《大模型指导的内核模糊测试》
  - 2024.05.18 徐菊彬《大模型指导的协议模糊测试》
  - 2024.09.03 张浩然《大模型赋能的模糊测试用例生成技术》
  - 2024.06.09 谢宁　《基于变异的模糊测试》
  - 2024.05.26 邵思源《面向网络应用程序的模糊测试》

内容提要

- 预期收获
- 题目内涵解析
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - ProphetFuzz
  - G^2FUZZ
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 了解大模型在软件模糊测试中的应用方法基本概念
  - 理解大模型对软件模糊测试性能提升的思路和应用场景
  - 掌握大模型在软件模糊测试中的使用方式

内涵解析与研究目标 大模型协同的软件模糊测试技术

- 研究目标
  - 在软件功能日渐复杂、全面的背景下，利用大语言模型的理解、生成能力，并保留传统灰盒模糊测试效率，引导测试更快速地触发潜在漏洞
- 题目内涵解析
  - 模糊测试(Fuzz)：通过向测试对象提供大量畸形测试用例作为输入，并监视其错误响应，以揭示潜在的异常缺陷及安全漏洞
  - 软件灰盒模糊测试：以二进制文件测试对象，利用覆盖率反馈信息引导测试
  - 大模型协同模糊测试：将大模型应用于传统模糊测试，提升自动化水平和测试效果

研究背景 大模型协同的软件模糊测试技术

- 研究背景
  - 软件日益复杂化：随着功能需求的增多，软件日益复杂，如何确定测试对象
  - 通用模糊测试专注于触发尽可能多的代码块，在软件功能复杂、分支较多时，无法高效的测试易出现问题的区域
  - 大语言模型的发展：大模型具有优越的理解、生成能力和agent模式，但受制于响应速度和API成本等问题，相悖于fuzz强调的大量、高效测试，如何平衡成本与效率，并取得好的测试结果
- 研究意义
  - 通过不同协同模式，提升模糊测试的自动化水平和覆盖效率，增强漏洞发现能力

研究历史与现状 大模型协同的软件模糊测试技术

2023 Deng等人提出TitanFuzz模糊测试方法，利用大语言模型自动生成和修改程序输入以测试深度学习库。方法首先生成程序的初始种子，再应用进化算法逐步产生新的代码片段

2024 Zhang等人提出LLAMAFuzz方法，利用预训练知识生成新的有效输入，并通过成对变异种子进一步微调模型以有效学习结构化格式和变异策略

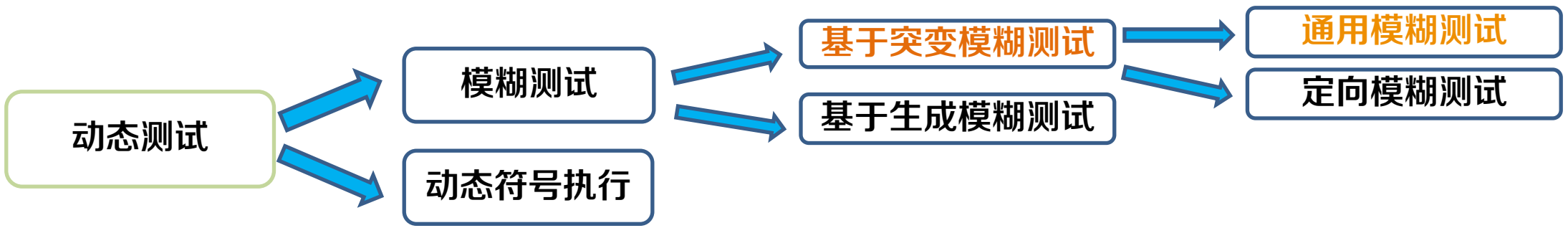
2024 Wang等人为优化模糊测试对象选取，提出了ProphetFuzz，通过分析待测目标的选项关系并定位高风险测试目标，提升洞触发数量

2024 Zhang等人针对模糊测成本过高问题，提出G2FUZZ通过大语言模型生成测试用例生成脚本，仅在fuzz遇到覆盖率瓶颈时使用大模型优化脚本，提升漏洞数量

2024 Eom等人提出CovRL，通过融合了覆盖率引导的大语言模型，通过将覆盖反馈直接融入到LLM，显著提升了针对JavaScript引擎的模糊测试性能

2025 Asmita等人利用大语言模型生成目标特定的初始种子，大幅度提高了崩溃数量，同时通过崩溃数据分析在不进行传统模糊测试的情况下识别最新BusyBox版本中的崩溃

2025 Xu等人提出CKGFuzzer方法，利用大语言的代码知识图谱驱动，通过分析程序间关系构建图谱，针对特定API场景提升漏洞触发率



知识基础 AFL

- 基本概念
  - AFL（American Fuzzy Lop）是一种流行的模糊测试工具，用于自动化发现软件漏洞
- 运行方式
  - 通过生成大量随机输入数据并将其提供给目标程序运行，检测程序在异常输入下的行为
- 特点
  - 基于覆盖率的反馈：使用代码覆盖率作为反馈机制，以最大化覆盖新的代码路径
  - 易用性：AFL 具有简单的命令行界面，易于集成和使用

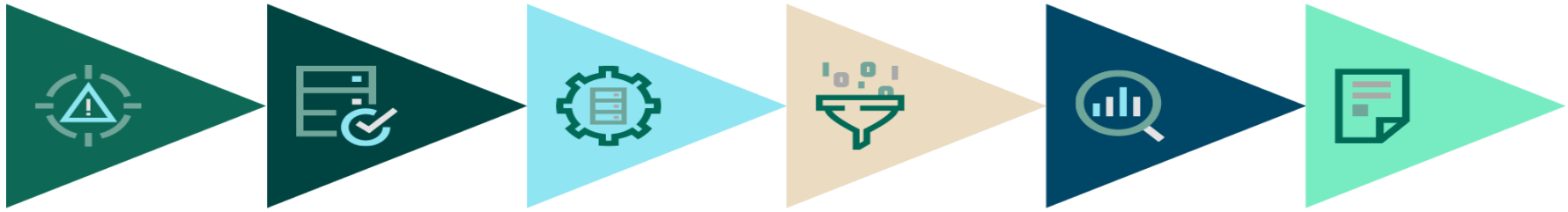
Algorithm 1: AFL Algorithm.

Input: Seed input: *seeds*, Instrumented target program: P
Output: Seed queue: Q, Crash set: C

```
1  Q ← seeds
2  C ← ∅
3  while TRUE do
4      s ← ChooseNext(Q)
5      e ← AssignEnergy(s)
6      for op in deterministicOps do
7          for i in Range(0, s.length) do
8              s' ← Mutation(s, op, i)
9              status ← Run_Target(P, s')
10             Save_Update(status)
11     for i in Range(0, e) do
12         op ← Random(non−deterministicOps)
13         s' ← Mutation(op)
14         status ← Run_Target(P, s')
15         Save_Update(status)
16 Procedure Save_Update(status)
17 if is_Crash(status) then
18     C ← C ∪ s'
19     return
20 if is_NewCoverage(status) then
21     Q ← Q ∪ s'
22     return
```

知识基础 模糊测试

- 模糊测试
  - 生成大量随机输入数据，提供给目标程序运行，检测程序在异常输入下的行为
  - 但是在目标程序代码量大、功能复杂时效率较低下
- 常见研究对象
  - 输入构造机制：高质量种子生成、变异算法设计、生成模型
  - 空间搜索策略：定向模糊测试、能量分配算法、路径优先级策略
  - 反馈机制设计：覆盖率信息设计、距离引导信息设计、语义反馈信号
  - 程序推理机制：混合符号执行(Concolic)、污点分析
  - 执行效率优化：低成本执行、插桩开销优化



研究现状

- 大模型协同的软件模糊测试技术
  - 通过适当的调度方法，使模糊测试更智能化和自动化，寻找程序崩溃
  - 适用场景
    - 补丁验证、敏感功能测试
    - 漏洞复现与验证
    - 探索程序深层次漏洞
  - 面临问题
    - 目标选择方面：测试对象一般人为设定，缺乏多样性和自动化
    - 反馈机制方面：大模型往往不直接接收测试的反馈信息，仅作为生成器使用
    - 执行效率方面：大模型的生成效率较低，影响模糊测试的效率

ProphetFuzz: Fully Automated Prediction and Fuzzing of High-Risk Option Combinations with Only Documentation via Large Language Model

关键词：全自动；选项组合；仅文档；大模型

ProphetFuzz TIPO

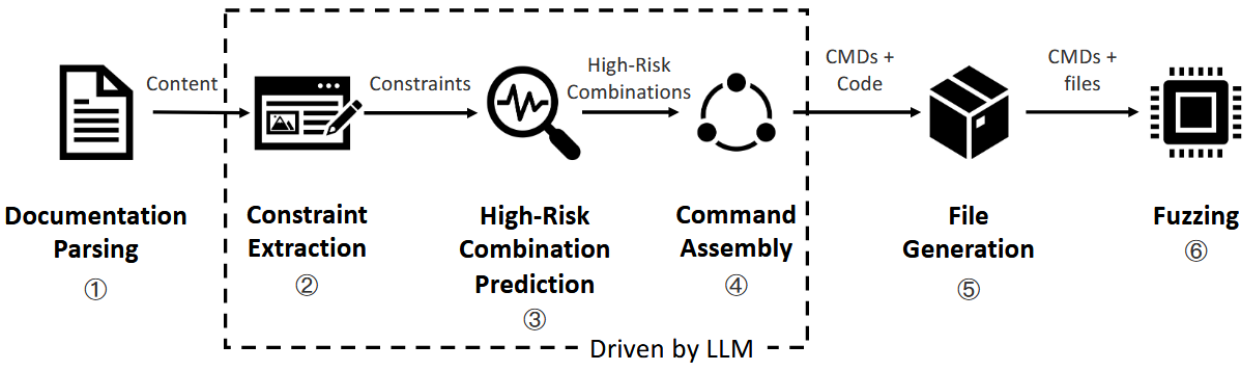
T	目标	针对待测程序的不同命令行选项，自动筛选高风险组合，寻找漏洞
I	输入	待测程序文档（程序名、程序描述、选项及其说明-help/man）
P	处理	1. 提取选项约束，通过双向推理去除其中错误组合 2. 使用小样本学习已有漏洞，预测高风险组合， 3. 组装可执行命令，并自动生成配套文件代码（生成复杂输入） 4. 执行文件生成代码并驱动模糊测试
O	输出	崩溃信息、引起崩溃的输入

P	问题	程序选项及其组合数量激增，如何高效定位高风险组合
C	条件	待测软件的文档拥有较高质量
D	难点	如何减少模型幻觉带来的错误组合
L	水平	Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security(CCS '24)

ProphetFuzz 算法原理

- 核心创新：1个预测范式，2个推理增强机制
  - 基于文档预测高风险选项组合
    - 使用大语言模型直接，从选项描述中预测更可能触发深层内存破坏漏洞的选项组合
  - 使用双向推理校验生成结果
    - 对冲突/依赖的测试选项进行验证与反证，过滤大模型幻觉导致的错误选项组合
    - 避免无效组合进入后续预测，减少因错误约束带来的提前退出和目标误筛
  - 少样本知识补全
    - 从少量历史漏洞中自动生成高风险组合示例，减少大模型知识缺口
- 具体实现
  - 预设提示词模板
  - 生成Python程序执行，获得测试输入

高效定向，减少无用

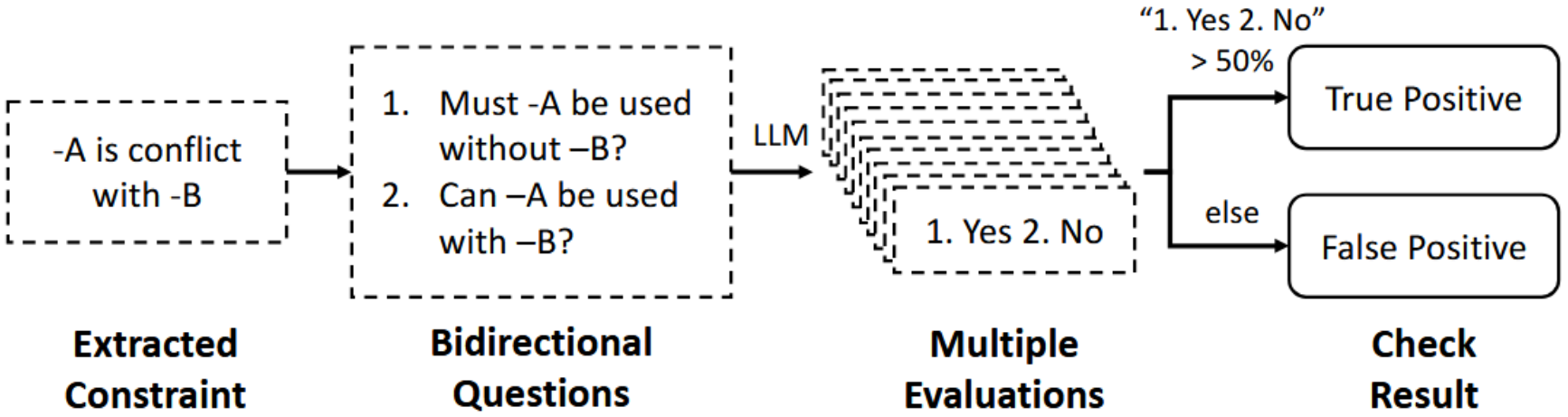


ProphetFuzz 算法原理—选项组合筛选

- 根据待测程序的文档抽取可能的选项组合，如何减少模型幻觉产生的错误
  - 冲突组合：两个选项不能同时使用
  - 依赖组合：两个选项必须同时使用
  - 大模型错判冲突组合为高风险
  - 直接测试需要生成复杂的内容

Constraint	Type	Question
Conflict	Verification	Must -A be used without -B?
Conflict	Counterexample	Can -A be used with -B?
Dependency	Verification	Must -A be used with -B?
Dependency	Counterexample	Can -A be used without -B?

- 双向推理：针对候选约束对，多次询问模型正反问题，获得稳定的输出
  - 设计4种问题，对一对约束能否组合，从正反两个方面多次提问



ProphetFuzz 算法原理—高风险组合预测

- 使用大模型预测可能产生崩溃的选项组合：思维链(Chain-of-Thought, CoT)
  - 预测组合：理解程序-分析选项-记住约束-引导模型假设性思考
  - 跨程序进行少样本学习，引导大模型分析已有的高风险组合模式
  - 提升思考表现：Let's take a deep breath and think step by step

Prompt for Prediction

Here is the document of [Program name],
```
{"name": [Program name], "description": [Program description], "options": [Option descriptions], "requirement": [Constraints]}
```

Instructions:
1. Understand the core functionality of the program from the "name" and "description" fields.
2. Analyze individual options and their respective roles from the "options" field.
3. Remember the constraints in "requirements" field and use them to guide subsequent steps.
4. Hypothetically analyze all combinations of any options that, when used together, could lead to deep memory corruption vulnerabilities while functioning correctly.
5. Add more options in the combination to make it easier to trigger the vulnerability.
6. Provide the final results in JSON format.

Let's take a deep breath and think step by step. Please show your thoughts in each step.

Examples:
[Put Examples here]

Prompt for Example Generation

Here is the document of [Program name],
```
{"name": [Program name], "description": [Program description], "options": [Option descriptions], "requirement": [Constraints]}
```

Buffer vulnerabilities have occurred in the following option combinations,
[Put combinations here]

Instructions:
1. Understand the core functionality of the program from the "name" and "description" fields.
2. Analyze all the listed options in the "options" field and their respective roles.
3. Remember the constraints in "requirements" field and use them to guide subsequent steps.
4. Hypothetically analyze why these option combinations are susceptible to buffer vulnerabilities and summarize what kind of combinations of any options that, when used together, could lead to deep memory corruption vulnerabilities while functioning correctly.
5. Examine whether any options in the combination, while not directly causing the vulnerability, might facilitate its trigger.
6. Provide the final results in JSON format.

Let's take a deep breath and think step by step. Please show your thoughts in each step.

ProphetFuzz 实验设置

- 对比方法
  - CarpetFuzz（SOTA）
- 被测程序
  - 40个软件库中的52个软件
  - 26种输入格式，包括视频、音频、文本、二进制等
- 评价指标
  - 边覆盖率(afl-showmap)、uniq-crash数量
- 硬件环境
  - Intel Xeon(R) CPU E5-2630 v3 @ 2.40GHz、128G内存（弱于实验室GPU服务器）
- 测试时间72小时，重复5次
- 所有测试总计耗时10.44 CPU年

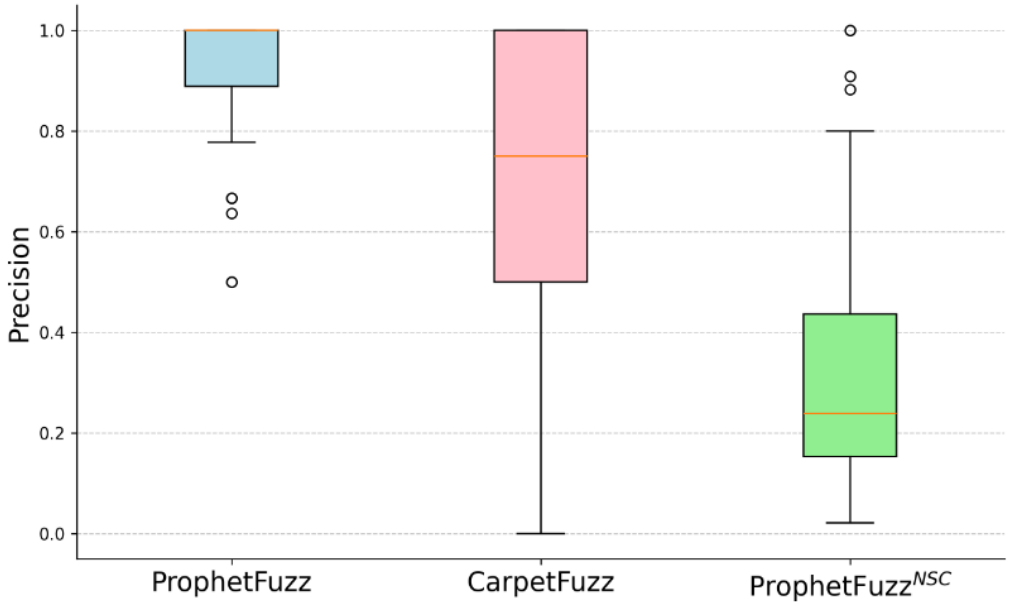
ProphetFuzz 对比实验

- **RQ1: ProphetFuzz发现漏洞性能**
  - **ProphetFuzz预测1748种高风险组合，组装7614条命令，仅为对比方法的20%**
  - **发现364个独立漏洞，较对比方法提升33%**
  - **更少的命令量实现了更高的漏洞发现效率，且覆盖率几乎无变化**

Program	#Cmds		#Uniq Vuls		#Excl Vuls		Vul Com Ratio		#Edge Cov.		
	$data_P$	$data_C$	$data_P$	$data_C$	$data_P$	$data_C$	$data_P$	$data_C$	$data_P$	$data_C$	r
avocnv	497	3414	4	**12**	2	**10**	**5.45%**	2.81%	**48951**	35006	139.84%
bison	221	484	**13**	8	**5**	0	**8.82%**	0.21%	6229	**6305**	98.79%
c++filt	25	78	**96**	63	**60**	27	4.00%	**5.13%**	3268	**3372**	96.92%
cflow	162	632	**17**	**17**	**6**	**6**	**14.29%**	0.32%	1453	**1700**	85.47%
cjpeg	53	555	0	0	0	0	0.00%	0.00%	775	**782**	99.10%
cmark	52	150	0	0	0	0	0.00%	0.00%	7000	**7737**	90.47%
vim	243	1031	0	**2**	0	**2**	0.00%	**0.19%**	10630	**49166**	21.62%
xmlcatalog	66	116	0	0	0	0	0.00%	0.00%	**6129**	5864	104.52%
xmllint	109	664	2	**17**	1	**16**	2.22%	**14.31%**	**12609**	10946	115.19%
xmlwf	67	329	0	0	0	0	0.00%	0.00%	**3473**	3104	111.89%
yara	74	455	0	**3**	0	**3**	0.00%	**0.22%**	2588	**3021**	85.67%
Count	7614	38984	**364**	274	**224**	134	**12.30%**	1.50%	393250	**394999**	99.56%

ProphetFuzz 对比实验

- RQ2: ProphetFuzz发现约束组合的精确度
  - ProphetFuzz获得633条约束，准确率为100%
  - CarpetFuzz获得447条约束，准确率75%
  - ProphetFuzz不进行双向检查时，获得6682条约束，但准确率仅为23.4%
  - 双向推理是ProphetFuzz约束提取可用性的关键，
  - 通过语义识别文档中未提到的冲突



ProphetFuzz 消融实验

- RQ3&4: ProphetFuzz各个组件对性能的影响
  - 实验设置
    - ProphetFuzz_NSC-去掉双向检查
    - ProphetFuzz_ZS-去掉少样本学习
    - OBLLM-基线大模型(GPT-4 Turbo)，且无提示
  - 缺少双向检查，6个程序无约束，仅发现84个漏洞
  - 缺少历史知识，仅发现153个漏洞
  - 双向检查：正确推理约束组合
  - 少样本学习：准确地找到危险组合
  - 错误约束影响大模型思维链的推理

测试目标更有效

Program	ProphetFuzz	Prophet^{NSC}	Prophet^{ZS}	OBLLM
cflow	34	18	21	25
cmark	0	-*	0	0
djpeg	0	-*	0	0
dwarfdump	7	1	1	4
eu-elfclassify	0	0	0	0
exiv2	5	0	4	3
ffmpeg	17	3	9	7
gif2png	26	-*	27	16
gm	0	1	0	0
img2sixel	24	2	13	2
jasper	0	-*	1	0
jpegoptim	10	6	8	18
jpegtran	1	1	2	0
lrzip	0	0	0	0
mutool	6	0	0	4
nasm	2	1	0	1
objdump	13	8	8	13
openssl-ec	0	0	0	0
openssl-rsa	0	0	0	0
pdftohtml	28	-*	27	30
pdftopng	27	17	15	6
pngfix	0	-*	0	0
size	0	0	0	0
tiffcrop	19	26	13	16
xmllint	2	0	4	0
Count	221	84	153	145

* Fail to predict.

ProphetFuzz 消融实验

- RQ5: ProphetFuzz各个组件对性能的影响
  - 实验设置
    - ProphetFuzz_NV：用数据集默认的参数值
    - ProphetFuzz_NS：用数据集自带初始输入
  - 生成参数值值
    - 多发现34.65%漏洞，coverage提升8.21%
    - 参数值帮助fuzz探索更多路径
  - 生成初始输入
    - 多发现17.24%漏洞，coverage仅提升0.26%
    - 生成更符合漏洞语义的测试输入
  - 部分数据集的初始值经过人为设计，反而更优

Program	ProphetFuzz		ProphetFuzz^{NV}		ProphetFuzz^{NS}	
	#vuls	#cov	#vuls	#cov	#vuls	#cov
cflow	**17**	**1453**	10	1367	12	1445
cmark	0	7000	0	**7739**	0	7001
djpeg	0	**438**	0	400	0	416
dwarfdump	2	7767	**4**	6958	1	**7808**
eu-elfclassify	0	**213**	0	**213**	0	203
exiv2	**2**	**8496**	1	8128	1	7671
ffmpeg	**4**	71198	1	65551	2	**71667**
gif2png	**10**	**441**	**10**	427	**10**	440
gm	0	**10172**	0	9415	0	8727
img2sixel	5	2207	3	2136	**7**	**2440**
jasper	0	**3416**	**1**	3182	0	2161
jpegoptim	**4**	**244**	2	202	**4**	242
jpegtran	0	**5773**	0	5662	0	5672
lrzip	0	4069	0	3941	0	**4706**
mutool	**1**	13118	0	9601	**1**	**16556**
nasm	**17**	7399	2	6205	11	**8331**
objdump	**9**	11392	7	**11902**	6	11050
openssl-ec	0	**9474**	0	9150	0	8420
openssl-rsa	0	**8034**	0	7949	0	7529
pdftohtml	**3**	5635	**3**	5611	0	**6224**
pdftopng	**12**	4531	11	3653	5	**4895**
pngfix	0	**1058**	0	988	0	922
size	0	**3708**	0	3602	0	2842
tiffcrop	65	6283	48	6038	**67**	**6555**
xmllint	2	**12609**	**8**	11764	1	11681
Count	136		101		116	

ProphetFuzz 实战效果

- RQ6: ProphetFuzz发现真实bug能力测试
  - 共发现140个漏洞
  - 分布在15个程序中，覆盖11种漏洞类型
  - 93个已被开发者确认
  - 26个已修复
  - 21个已获得CVE编号

Program	Vulnerability Type	Count	#Fixed	#CVEs
avconv	assertion failure	1	0	0
avconv	floating point exception	1	0	0
avconv	global-buffer-overflow	1	0	0
avconv	heap-buffer-overflow	21	0	0
avconv	segmentation violation	6	0	0
avconv	use-after-free	1	0	0
bison	segmentation violation	10	0+	0
c++filt	stack-buffer-overflow	12	0+	0
cflow	global-buffer-overflow	1	0	0
cflow	use-after-free	2	0	0
dwarfdump	use-after-free	1	1	1
editcap	bad free	2	2	1
editcap	heap-buffer-overflow	1	1	1
editcap	segmentation violation	1	1	0*
ffmpeg	floating point exception	2	1+	1
ffmpeg	segmentation violation	3	2	2
img2sixel	floating point exception	2	0	0
img2sixel	heap-buffer-overflow	1	0	0
img2sixel	segmentation violation	1	0	0
img2sixel	use-after-free	1	0	0
jasper	assertion failure	1	1	1
jpegtran	segmentation violation	1	1	0*
mupdf	negative-size-param	1	1	1
mupdf	segmentation violation	1	1	1
nasm	segmentation violation	4	0	0
nasm	use-after-free	3	0	0
objdump	heap-buffer-overflow	2	1	1
pspp	assertion failure	29	8+	6
pspp	bus on unknown address	1	0+	0
pspp	double-free	1	0+	0
pspp	heap-buffer-overflow	3	0+	0
pspp	segmentation violation	13	4+	4
pspp	stack-buffer-overflow	5	0+	0
pspp	use-after-free	3	0+	0
xpdf	stack-buffer-overflow	1	1	1
Count		140	26	21

+ Have been confirmed by the developers.
* Applying for CVE.

Low-Cost and Comprehensive Non-textual Input Fuzzing with LLM-Synthesized Input Generators

关键词：低成本；非文本格式输入；全方面；大模型协同

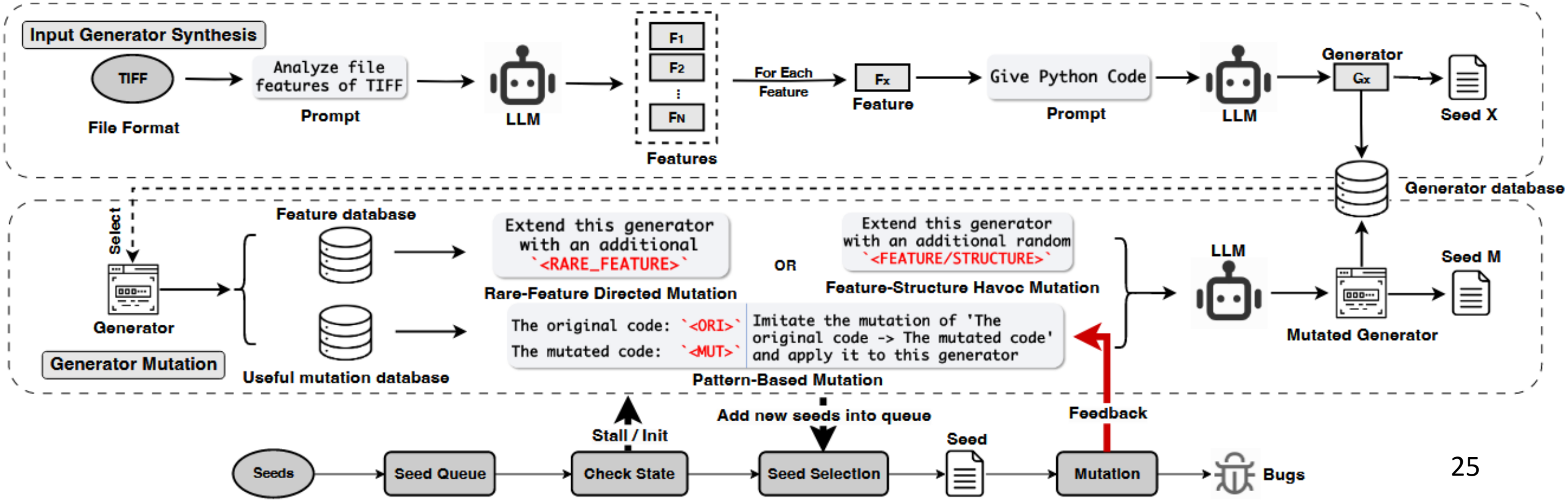
G²FUZZ TIPO

T	目标	针对复杂非文本格式输入的程序，自动生成并迭代输入生成脚本，寻找漏洞
I	输入	待测程序、目标输入格式名（如TIFF / PDF / MP4）
P	处理	1. 自动分析待测程序的输入格式特征 2. 使用LLM自动生成对应的输入生成脚本 3. 执行脚本生成复杂输入，并驱动AFL++模糊测试 4. 根据fuzzing反馈持续更新脚本，生成更复杂、更多样的输入
O	输出	崩溃信息（是/否）、引起崩溃的输入

P	问题	LLM生成慢、贵、吞吐低；传统fuzz不懂复杂格式语法
C	条件	大语言模型对待测输入格式有一定的了解
D	难点	如何减少大模型的使用次数并合理使用测试反馈
L	水平	34th USENIX Security Symposium (USENIX Security 25)

G²FUZZ 算法原理

- 核心创新：1个生成策略，1个反馈驱动方式
  - 面向非文本格式特征的生成策略：分析目标格式多种功能，生成输入生成脚本
  - 反馈驱动的生成器协同模糊测试：根据fuzzing反馈持续更新脚本，生成复杂输入
- 收集反馈、持续迭代



G²FUZZ 算法原理

- 基于LLM生成输入生成器：分析目标程序输入的所有格式特征(features)
  - 使用LLM提示词，分析目标输入格式的格式特征，获取fuzz测试的不同目标
    - TIFF：无损压缩（LZW、ZIP…）、颜色深度（RGB、灰度…）、…
    - ELF：加载方式（PIE、non-PIE）、链接方式（动态链接、静态链接）、…
  - 针对每一条特征，针对性的生成一个Python脚本，用于生成模糊测试输入
  - 执行n个生成器，获得模糊测试种子

```
What features can '<TARGET>' files have? Output the
information in the following format:
1. <feature 1>: <feature description>
2. <feature 2>: <feature description>
.....
N. <feature N>: <feature description>
```

```
Generate '<TARGET>' files containing the following features
using Python without any input files, and save the
generated files into './tmp/'.
```
<TARGET_FEATURES>
```
Please use Markdown syntax to represent code blocks. Please
ensure that there is only one code block. You don't need to
tell me which libraries need to be installed.
```

Algorithm 1: Generator Procedure

Input: The target file format, *target_format*.
Output: A set of (generator, feature description), *G*.

```
prompt ← construct_prompt(target_format)          // Based on Fig. 2
features ← LLM(prompt)
for f in features do
    g ← generator_generation(target_format, target_feature)
                                                  // Based on Alg. 2
    if g ≠ none then
        seeds ← run(g)
        add_to_queue(seeds)
        G ← (g, f)
```

G²FUZZ 算法原理

- 反馈驱动的输入变异模块：根据模糊测试反馈，迭代生成器，获得更多样输入
  - 稀有特征定向变异：init阶段，查漏补缺
    - 自检提示词，使LLM输出未覆盖到的格式特征
  - 复杂特征探索变异：stall阶段，跳出局部
    - 复杂变异提示词，探索更多区域
  - 基于模式变异：stall阶段，扩展高价值输入
    - 前两种变异中，导致覆盖率增加的<org, mul>
    - 迁移提示词，将模式应用到其他生成器
  - 仅在初始化和fuzz卡住时调用LLM，成本低

稀有特征定向变异

```
Analyzed features:
```
1. <feature 1>: <feature description>
2. ... : ...
```
Apart from the above features, what other features can '<TARGET>' files have? Output the information in the following format:
1. <feature 1>: <feature description>
2. <feature 2>: <feature description>
...
N. <feature N>: <feature description>
```

稀有特征定向变异

```
<TARGET_GENERATOR>
```
The code above is used to generate <FROMAT> files. Now, we need to extend this code to generate a new <FROMAT> file that includes an additional `<NEW_FEATURE>` feature besides the existing features. The description of the `<NEW_FEATURE>` feature is as follows:
```
<FEATURE_DES>
```
```

复杂特征探索变异

```
<TARGET_GENERATOR>
```
Based on the above code, provide me with a more complex code that can generate <FROMAT> files with additional more complex file <features/structures>.
```

基于模式变异

The original code: ```<ORI>```
The mutated code: ```<MUT>```
Imitate the mutation of 'The original generator -> The mutated generator' above and apply it to the following target code:
```
<TARGET_CODE>
```

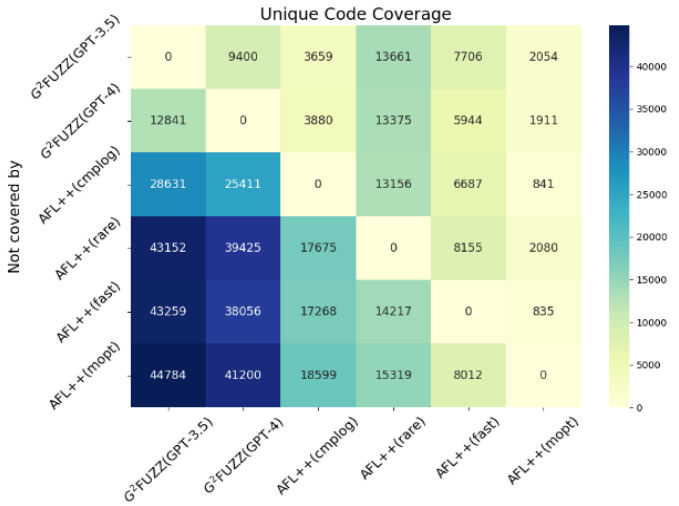
G²FUZZ 实验设计

- 对比方法
  - AFL++(cmplog)：使用REDQUEEN变异器
    - 利用比较信息迭代输入
  - AFL++(mopt)：使用MOPT变异器
    - 自适应优化变异策略
  - AFL++(fast)：使用AFLFast种子调度器
  - AFL++(rare)：优先考虑更罕见的种子输入
  - FormatFuzzer、WEIZZ、AFLSmart：结构感知类型fuzz
- 测试数据集
  - 10个测试程序，包含20种输入格式
- 评价指标
  - 边覆盖率、漏洞发现数量

UNIFUZZ	FuzzBench	MAGMA
gdk-pixbuf-pixdata	bloaty_fuzz_target	libpng_read_fuzzer
jhead	freetype2-2017	read_rgba_fuzzer
mp3gain	harfbuzz-1.3.2	tiffcp
ffmpeg	lcms-2017-03-21	pdf_fuzzer
tiffsplit	libjpeg-turbo-07-2017	pdfimages
pdftotext	libpcap_fuzz_both	pdftoppm
mp42aac	libpng-1.2.56	sndfile_fuzzer
flvmeta	openssl_x509	
imginfo	vorbis-2017-12-11	
exiv2	woff2-2016-05-06	
	zlib_zlib_uncompress_fuzzer	

G²FUZZ 对比实验—通用fuzz

- RQ1: G²FUZZ的性能如何?
  - 10个程序中的9个覆盖率最高，发现143个bugs提升28%
  - 覆盖代码更多，可以发现AFL++无法探索的区域
  - 24小时测试成本GPT-3.5：0.2美元，GPT-4：13美元
  - 协同测试的设计成本低、覆盖高



Programs	G²FUZZ(GPT-3.5)[1]		G²FUZZ(GPT-4)[2]		AFL++(cmplog)		AFL++(fast)		AFL++(mopt)		AFL++(rare)	
	Cov.	#Bug	Cov.	#Bug	Cov.	#Bug	Cov.	#Bug	Cov.	#Bug	Cov.	#Bug
exiv2	5,099	**31**	**5,171**	28	4,965	26	3,776	5	3,758	12	3,851	11
ffmpeg	31,706	0	**34,218**	**1**	22,099	0	17,380	0	15,566	0	14,613	0
flvmeta	228	4	228	4	228	4	228	4	228	4	228	4
gdk	**2,958**	**6**	2,327	2	2,172	5	2,093	4	2,082	2	1,991	4
imginfo	2,839	0	**3,825**	0	2,189	0	1,998	0	2,004	0	1,976	0
jhead	315	13	**491**	**23**	445	21	195	3	195	3	195	4
mp3gain	921	11	921	**12**	**923**	11	900	10	899	8	891	10
mp42aac	2,067	**17**	**2,700**	14	2,091	13	1,178	6	1,157	3	1,135	2
pdftotext	**8,265**	43	7,921	**49**	7,434	25	6,483	25	6,376	28	11,257	28
tiffsplit	1,817	7	**1,840**	**10**	1,659	6	1,619	9	1,644	9	1,626	7

[1] G²FUZZ(GPT-3.5): G²FUZZ based on GPT-3.5.
[2] G²FUZZ(GPT-4): G²FUZZ based on GPT-4.

G²FUZZ 对比实验—类似fuzz

- RQ2: 和格式感知类型fuzz的性能对比如何?
  - 9个程序上取得最大覆盖率，生成复杂有效的二进制输入
  - 相比Fuzztruction在7个程序上取得更大覆盖率
    - Fuzztruction基于突变，受种子质量影响
    - G2FUZZ从头开始在语义上构造不同结构的文件
    - zip相关测试受限于python库较保守，测试出漏洞更少

Program	G²FUZZ	FormatFuzzer	WEIZZ
gdk	**126**	0	-
exiv2	**424**	0	0
ffmpeg	**1951**	6	14
flvmeta	0	-	**1**
imginfo	**148**	0	0
jhead	**8**	0	0
mp3gain	**1**	-	0
mp42aac	**338**	-	0
pdftotext	**334**	-	0
tiffsplit	**10**	-	0

Input Format	Program	Fuzztruction		G²FUZZ	
		Cov	Bug	Cov	Bug
pdf	pdftotext-enc	36853.8	0	**38866.0**	0
pdf	pdftotext	35108.4	0	**39011.4**	0
elf	objdump	12468.8	0	**12851.8**	0
elf	readelf	12347.8	0	**13328.2**	0
png	pngtopng	4414.6	0	**4566.2**	0
der	vfychain	**14937.4**	0	11600.4	0
7z	7zip-enc	28887.2	**8**	**28909.0**	6
zip	7zip	**34585.4**	**8**	31691.4	6
zip	unzip	2788.2	1	**3104.8**	1

Programs	G²FUZZ		FormatFuzzer		WEIZZ	
	line	function	line	function	line	function
exiv2	**5,984**	**1488**	1,138	369	3,732	1025
ffmpeg	**53,664**	**3028**	23,114	1554	26,789	1795
flvmeta	623	59	-[1]	-	**632**	**60**
imginfo	**5,003**	**364**	2,128	193	3,481	275
jhead	**431**	**21**	239	16	300	18
mp3gain	**2,168**	**58**	#[2]	#	2,103	56
mp42aac	**3,378**	**811**	#	#	2,041	504
pdftotext	**13,733**	**1182**	-	-	9,133	914
tiffsplit	**3,176**	**194**	-	-	3,019	185
gdk	**4,856**	**315**	2,287	192	=[3]	=

[1] -: FormatFuzzer does not support PDF, TIFF, and FLV formats.
[2] #: We encountered issues while running FormatFuzz.
[3] =: We are unable to compile *gdk-pixbuf* by WEIZZ.

G²FUZZ 消融实验

- RQ3：各组件的作用和有效性，对成本的影响
  - Generator Mutation发现路径达到14万，接受fuzz反馈并变异对效果提升明显
  - Input Generator Synthesis合成的初始化输入相较于默认种子有提升
  - 两者分工不同，初始生成器将fuzz带入新分支区域，变异器在区域内扩展
  - 仅靠大模型本身的思维能力，token消耗量大速度慢，测试效果不佳

Programs	Initial Seeds	Input Generator Synthesis	Generator Mutation
tiffsplit	101	539	2,549
jhead	2	1,046	0
mp42aac	6,859	522	6,298
gdk	9,832	12,993	1
ffmpeg	4,877	14,603	109,381
exiv2	398	40,719	18,328
flvmeta	1,133	1,066	29
imginfo	21,236	593	0
mp3gain	1,675	1,377	4,266
pdftotext	7,735	8,543	488
Total	53,848	82,001	141,340

Programs	G²Fuzz (LLM-Only)	Throughput	Token Count	Cost($)
flvmeta	150	5,758,008	16,253,249	15.72
exiv2	2,126	7,643	20,831,178	17.80
gdk	830	18,007	18,253,243	16.63
imginfo	909	13,155	17,353,735	16.10
jhead	245	150,634	18,523,648	17.00
mp42aac	728	11,645	16,349,895	15.13
tiffsplit	938	6,729	20,539,769	17.75
mp3gain	691	9,337	18,547,650	16.31
pdftotext	3,300	3,084	22,996,919	19.20

G²FUZZ 真实漏洞发现实验

- RQ4：能否发现真实存在的漏洞
  - 在真实版本的软件上测试，效果领先其他fuzz程序
  - 发现10个漏洞，其中3个获批CVE

Program	G²FUZZ	AFL++	MOPT	AFLFast	LibFuzzer	Entropic
libpng_read_fuzzer	**3**	**3**	2	1	**3**	0
tiff_read_rgba_fuzzer	**5**	**5**	**5**	4	2	3
tiffcp	**7**	6	4	4	0	0
pdf_fuzzer	**5**	**5**	4	2	2	2
pdftoppm	**6**	5	**6**	2	0	0
pdfimages	**6**	5	5	3	0	0

Program	G²FUZZ	AFL++(cmplog)	AFL++(mopt)	AFL++(fast)	AFL++(rare)
mp3gain	1	1	-	1	1
pdftotext	1	1	1	1	1
mp42aac	**3**	**3**	1	-	-
mp42avc	**3**	-	-	-	1
mp42hevc	**2**	-	-	-	-
Total	**10**	5	2	2	3

特点总结与未来展望

- 特点总结

算法	ProphetFuzz	G²FUZZ
优势	1.自动化选取测试软件参数，并提取高危组合，提升漏洞发现效率 2.仅凭文档即可自动完成预测与模糊测试	1. 跨多种非文本格式泛化强 2. 大语言模型与传统fuzz协同高效测试 3. 低频调用LLM成本较可控
劣势	1. 偏重高风险优先，难覆盖全部有效组合 2. 参数取值偏常见，难命中特殊触发条件	1. 依赖python库的能力，较难做到任意位置修改 2. 系统链路复杂，实现难度较高

- 未来展望
  - 依赖建模：显式学习特征间依赖关系，生成更复杂且合法的输入
  - 场景扩展：支持多模态、嵌套格式与跨格式联合模糊测试
  - 自进化能力：利用在线反馈持续学习有效生成器变异策略

参考文献

[1] Zhang K, Li Z, Wu D, et al. {Low-Cost} and Comprehensive Non-textual Input Fuzzing with {LLM-Synthesized} Input Generators[C]//34th USENIX Security Symposium (USENIX Security 25). 2025: 6999-7018.

[2] Wang D, Zhou G, Chen L, et al. Prophetfuzz: Fully automated prediction and fuzzing of high-risk option combinations with only documentation via large language model[C]//Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security. 2024: 735-749.

道德经

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！