北京大学
PEKING UNIVERSITY

ChatFuzz 相关论文分享

2025.3.25

Augmenting Greybox Fuzzing with Generative AI

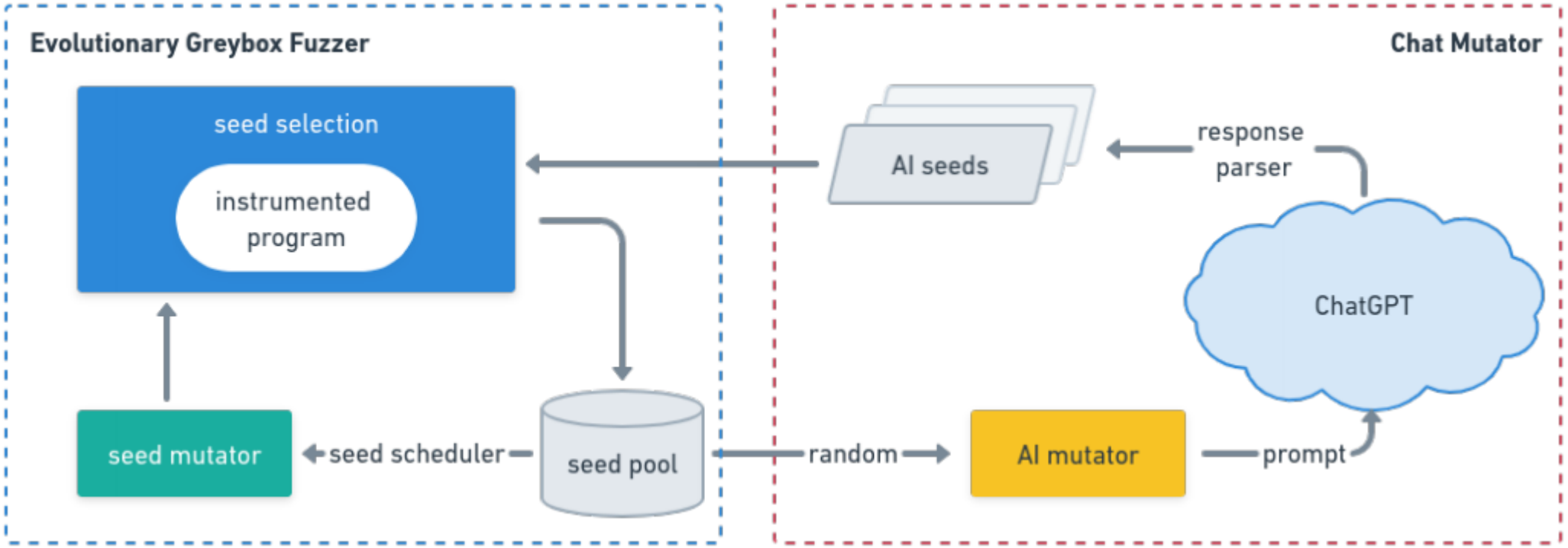
研究背景

在模糊测试中，为了深入探索程序的执行路径和状态，测试输入的有效性至关重要。如果输入格式不正确，程序会在解析阶段直接拒绝，从而限制了对更深层次程序状态的探索。

- **基于突变的方法：**通过对已有的合法输入数据进行随机或半随机的修改来生成新的测试用例。这些修改可以包括改变字节、插入或删除数据、翻转位等。当输入数据的结构复杂时，随机突变可能无法生成有意义的测试用例。
- **基于生成的方法：**依赖于对输入数据的格式或协议的理解，通过预先定义的规则或模型生成测试用例。这种方法通常需要对被测程序的输入格式建模，然后根据模型生成符合格式要求的测试用例。实现复杂，定义模型需要对输入数据的格式或协议有深入的理解。

工作内容

作者提出了一个新的模糊测试框架CHATFUZZ，通过生成AI增强模糊测试。CHATFUZZ在灰盒模糊测试的典型工作流程之外，会从种子池中挑选一个种子，并通过prompt引导LLM生成类似的输入。之后由灰盒模糊器进行评估，将其中一些（能够提高分支覆盖率）保留为新种子以供进一步探索。



选择超参数

1. 对于不同功能的各种模型，哪种模型更适合用来生成测试用例输入。
2. 为模型设计不同的提示模板，比较提供不同提示信息对模型输出的影响。
3. 评估了效率相关的max_tokens和 n的参数值选择，这两个超参数与模型查询延迟高度相关。
4. 为了生成可提高代码覆盖率的高质量种子，需要确定超参数temperature的最优值。

benchmark。为了避免过拟合，在确定上述超参数时对jq、php、mujs 和 xml四个程序进行了实验。其中：mujs 以 JavaScript 源代码作为输入，jq、php、xml 分别 以 JSON、PHP、XML文件作为输入。通过评估分支覆盖率增长、测试用例唯一率和语法有效率三项指标来选取超参数。

选择超参数—模型选择

作者对 Chat Model 和 Completion Model 两种类型的模型进行了评估。

- Chat Model 可以根据提供的对话的消息列表做出响应
- Completion Model 可以根据提示词补全内容

type	Chat model	Completion model
model	gpt-4、gpt-4-0314、gpt-4-32k、gpt-4-32k-0314、**gpt-3.5-turbo**、gpt-3.5-turbo-0301	text-davinci-003、text-davinci-002、**text-curie-001**、text-babbage-001、text-ada-001.
price	$0.0002 per 1000 tokens	$0.0020 per 1000 tokens

选择超参数—提示词设计

为了评估不同提示对生成测试用例的影响，作者评估了三种不同的提示词配置。① **AI**提供示例样本和文件格式；②**AI_noINPUT**只提供文件格式，不提供示例样本；③**AI_noFORM**只提供示例样本，不提供文件格式。

Config	Prompt Info.		Model	Prompt Template
	Sample Input	Format		
AI	✓	✓	CT	System: ”You are a <format>file generator” User: ”Here is an example <format>file, generate another one.” + <sample input>
			CP	<sample input>+ ”And here is another <format>file like above: ”
AI_noINPUT	✗	✓	CT	System: ”You are a <format>file generator” User: ”Generate a <format>file.”
			CP	”Here is a <format>file: ”
AI_noFORM	✓	✗	CT	System: ”You are a file generator” User: ”Here is an example file, generate another one.” + <sample input>
			CP	<sample input>+ ”And here is another one like above: ”

选择超参数—提示词设计

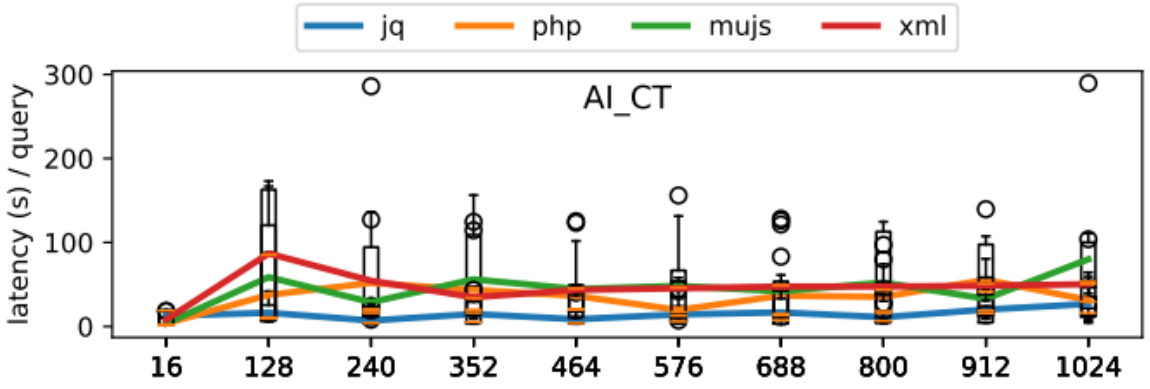
为了评估不同提示对生成测试用例的影响，作者评估了三种不同的提示词配置。① **AI**提供示例样本和文件格式；②**AI_noINPUT**只提供文件格式，不提供示例样本；③**AI_noFORM**只提供示例样本，不提供文件格式。

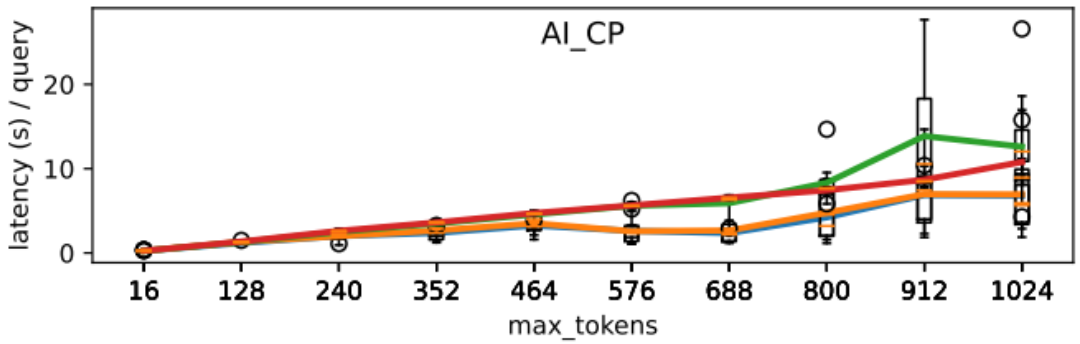
Program	CT endpoint					CP endpoint				
	AI	AI_noFORM		AI_noINPUT		AI	AI_noFORM		AI_noINPUT	
	cov	cov	vs. AI	cov	vs. AI	cov	cov	vs. AI	cov	vs. AI
jq	3837	4015	+4.64%	3555	-7.35%	4043	4015	-0.69%	3555	-12.07%
php	18995	19609	+3.23%	18364	-3.32%	20021	19609	-2.06%	18364	-8.28%
mujs	11233	10924	-2.75%	6819	-39.29%	13763	10924	-20.63%	6819	-50.45%
xml	7217	6988	-3.17%	6209	-13.97%	7832	6988	-10.78%	6209	-20.72%
Average			**+0.49%**		**-15.98%**			**-8.54%**		**-22.88%**

选择超参数—max_tokens

max_tokens表示模型响应的最大长度。如果max_tokens取值较小，可能导致生成的测试用例被截断。如果取值较大，模型可能需要更多的时间来完成请求和生成响应。因此，需要在效率和生成测试用例质量之间取得平衡。

为了找到这个超参数的最佳配置，作者分别研究了在相同提示词下使用 chat model 和 completion model 时，max_tokens 从16到1024的模型延迟。

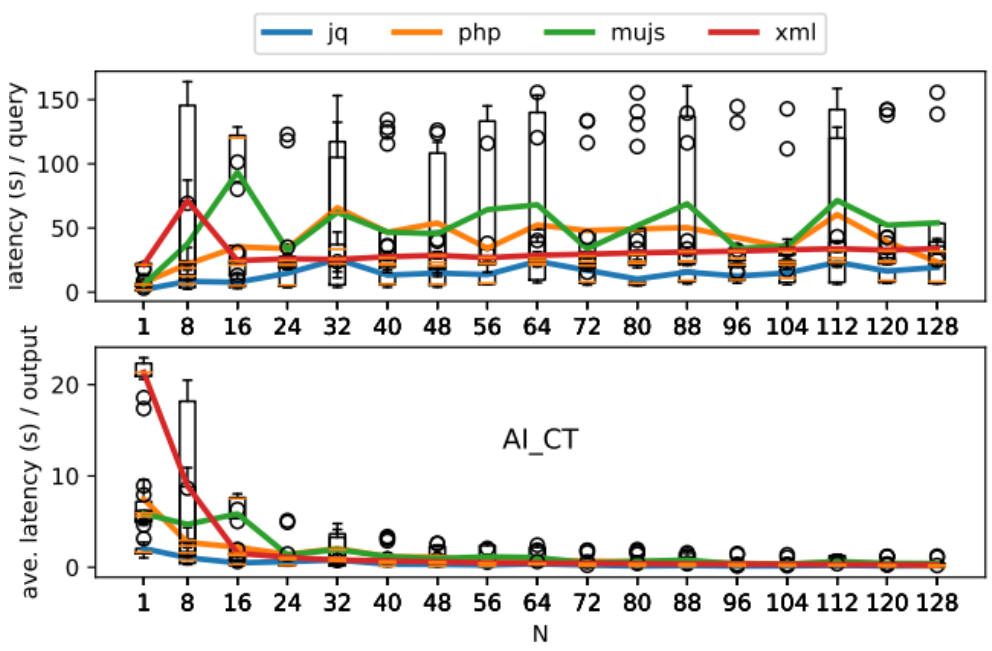


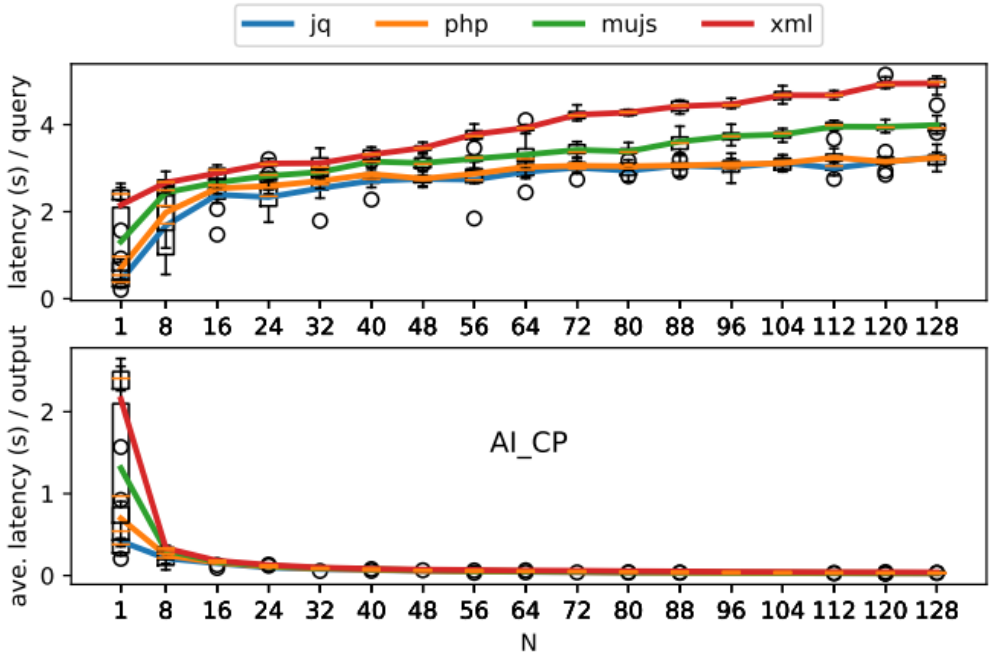


选择超参数—n

n表示模型响应中生成的独立回复数。作者评估了两种模型在不同n值（1到128）下每次模型查询的延迟时间以及平均每个生成测试用例的延迟时间。

根据结果可以看到，随着n值的增大，单次查询成本逐渐增加，而生成单个测试用例的平均成本逐渐降低（即效率提高）。





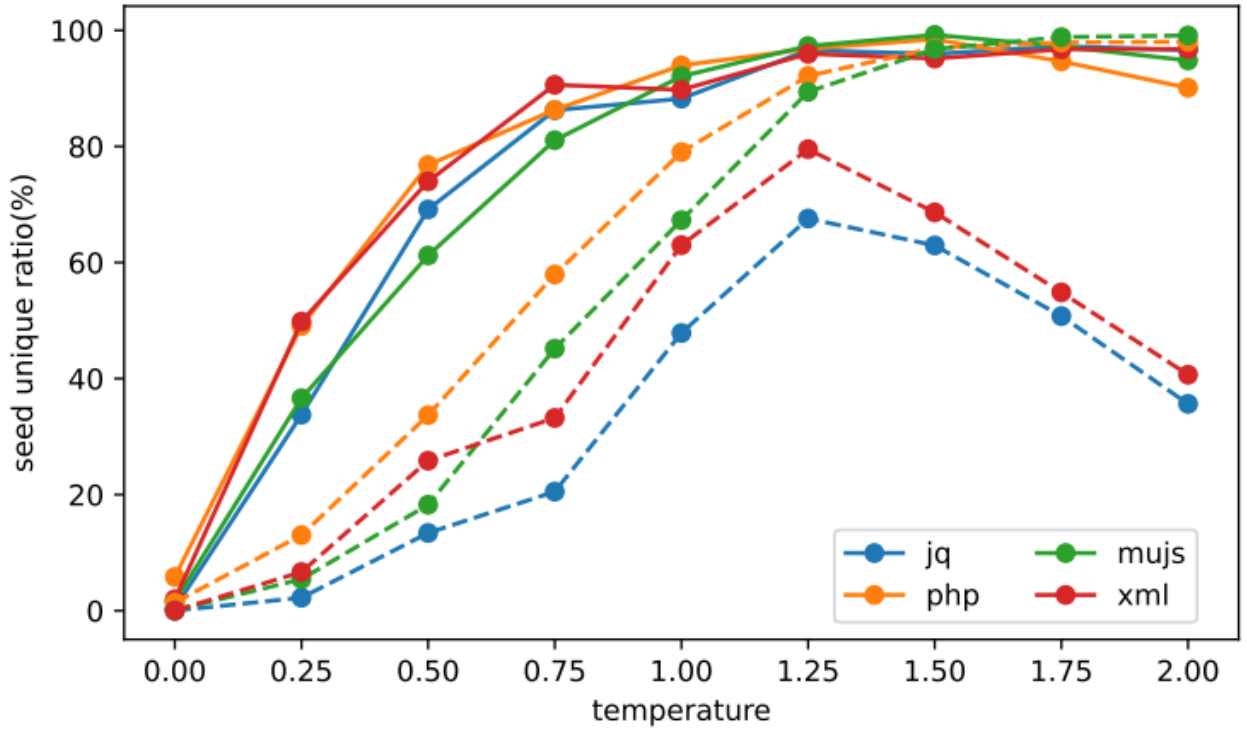
选择超参数—temperature

temperature表示模型响应内容的随机程度，是一个介于0和2之间的浮点值，默认值为1。当temperature值越小，模型的确定性越高，反之，模型的输出会更加随机。在temperature值较小时，模型会输出一个与输入样本相似的响应，甚至完全相同。当temperature值较大时，输出可能会变得过于随机，甚至输出的格式不再符合语法规则。为了找到最佳的temperature值，作者用两种模型分别连续运行2小时。对于每个模型，将temperature值设置为取0到2之间的值，步长为0.25。试验结束后，使用三个指标来评估生成的种子的质量： 1)种子唯一性比率，2)种子有效性比率和3)代码覆盖率。

Sample Input	**ChatGPT Response Examples**	
	Example 1	Example 2
<doc> <clean> </clean> <dirty> A B </dirty> <mixed> A <clean> </clean> B <dirty> A B </dirty> C </mixed> </doc>	<doc> <clean> </clean> <dirty> X Y Z </dirty> <mixed> X <clean> </clean> Y <dirty> X Y Z </dirty> Z </mixed> </doc>	<?xml version="1.0" encoding="UTF-8"?> <books> <book> <title>The Great Gatsby</title> <author>F. Scott Fitzgerald</author> <genre>Drama</genre> </book> <book> <title>Pride and Prejudice</title> <author>Jane Austen</author> <genre>Romance</genre> </book> </books>

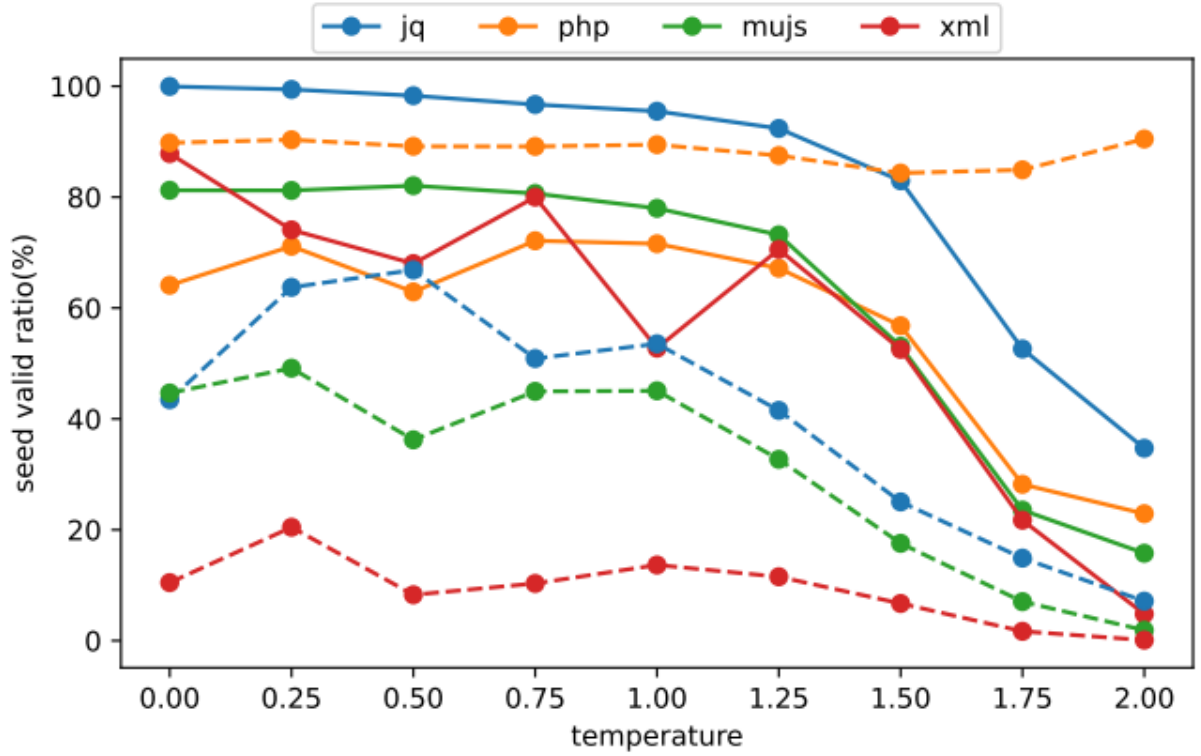
选择超参数—temperature

种子唯一性比率。整体来看，随着参数值的增大，生成的种子越来越多样。对于chat model，独特种子的比率随着temperature的增加而增加。对于completion model，程序jq (JSON)和xml (XML)的种子独特比随着temperature从0增加到1.25而增加，之后开始下降。



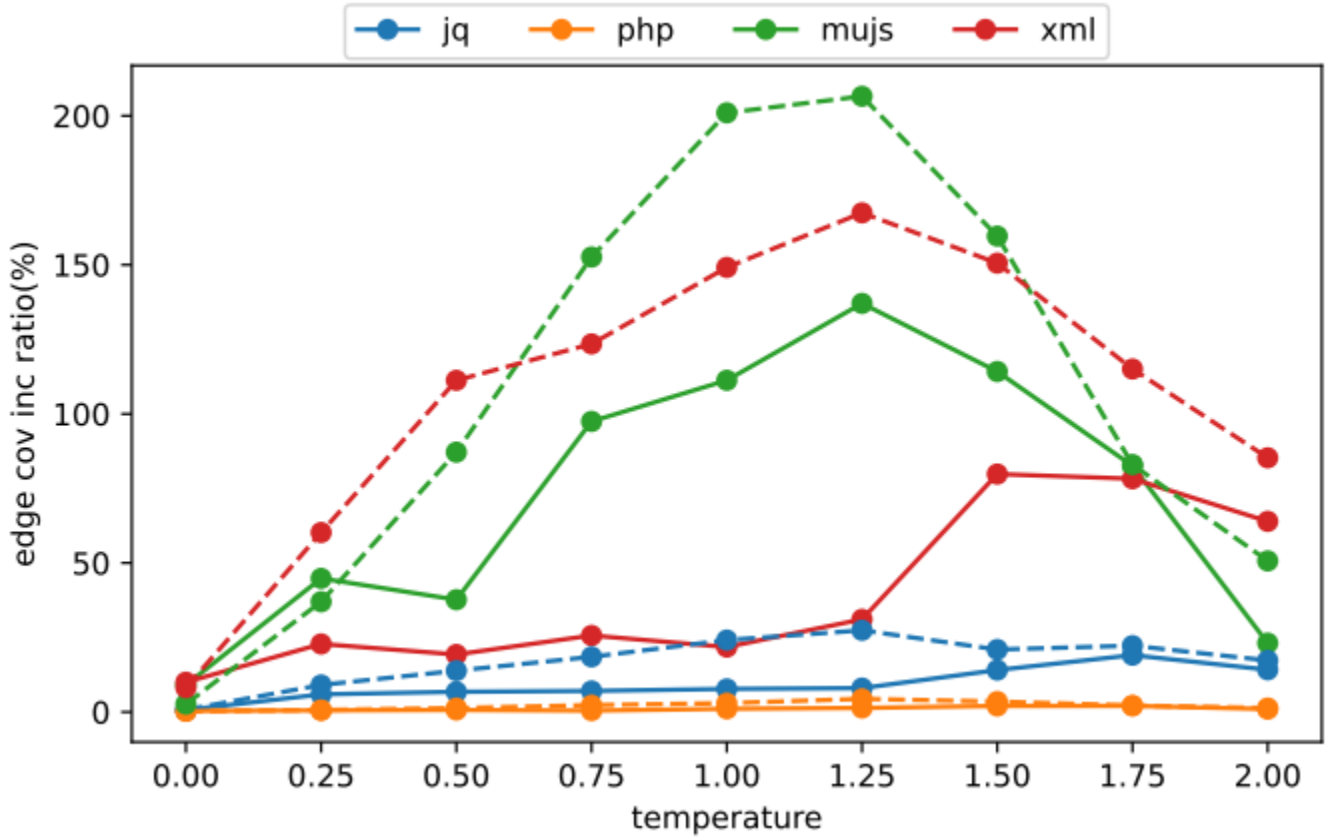
选择超参数—temperature

种子有效性比率。可以看到，有效种子的比例随着temperature的升高而降低。这是由于随着temperature的升高，模型的输出更加随机，因此更有可能违反语法规则。



选择超参数—temperature

代码覆盖率。在较小的temperature下，模型会产生大量的重复种子；在较大temperature的情况下，模型会生成很多语法无效的种子，因此对代码覆盖率的改进都不大。



效果评估

CHATFUZZ只能支持以格式化文本为输入的目标程序。作者从之前工作中大量用于评估的45个程序中选出了CHATFUZZ可以生成输入种子的12个程序。根据输入种子的格式，可以将12个被测程序分为三类：1)格式化的数据文件；2)不同编程语言中的源代码；3)没有显式语法规则的文本。

Type	Program	Version	Input Format
data	jq	jq-1.5	json
	php	php-fuzz-parser_0dbedb	PHP
	xml	libxml2-v2.9.2	XML
	jsoncpp_fuzzer	jsoncpp	json
code	mujs	mujs-1.0.2	js
	ossfuzz	sqlite3_c78cbf2	SQL
	cflow	cflow-1.6	C
	lua	lua_dbdc74d	lua
text	curl_fuzzer_http	curl_fuzzer_9a48d43	HTTP response
	openssl_x509	openssl-3.0.7	DER certificate
	base64	LAVA-M	.b64 file
	md5sum	LAVA-M	md5 checksum

效果评估—覆盖率

为了验证CHATFUZZ对AFL++覆盖率的效果，作者在四种模型配置下测量了12个项目连续运行8小时的分支覆盖率。

- CHATFUZZ：AFL++ & completion model & AI
- CHATFUZZ-F： AFL++ & completion model & AI_noFORM
- CHATFUZZ-C： AFL++ & chat model & AI
- CHATFUZZ-C： AFL++ & chat model & AI_noFORM

Program	8h Code Coverage					Improvement over AFL++			
	AFL++	CHATFUZZ	CHATFUZZ-F	CHATFUZZ-C	CHATFUZZ-CF	CHATFUZZ	CHATFUZZ-F	CHATFUZZ-C	CHATFUZZ-CF
jq	4045	4109	4122	4101	**4124**	1.58%	1.90%	1.38%	**1.95%**
php	22083	**22507**	22308	22388	22389	**1.92%**	1.02%	1.38%	1.39%
xml	7687	**9555**	9067	9027	9189	**24.30%**	17.95%	17.43%	19.54%
jsoncpp_fuzzer	1236	**1248**	1247	**1248**	1250	**0.97%**	0.89%	**0.97%**	1.13%
mujs	9149	**16171**	11982	13194	13023	**76.75%**	30.97%	44.21%	42.34%
ossfuzz	28880	32751	**33105**	32582	32120	13.40%	**14.63%**	12.82%	11.22%
cflow	2499	**2513**	2511	2495	2501	**0.56%**	0.48%	-0.16%	0.08%
lua	15213	15530	**15707**	15633	15585	**2.08%**	3.25%	2.76%	2.45%
curl	13476	13761	**13762**	13687	13699	2.11%	**2.12%**	1.57%	1.65%
openssl_x509	18000	17976	18015	**18016**	17970	-0.13%	0.08%	**0.09%**	-0.17%
base64	368	368	368	368	**372**	0.00%	0.00%	0.00%	**1.09%**
md5sum	482	**525**	482	**525**	482	**8.92%**	0.00%	**8.92%**	0.00%
Average	10259.8	**11417.8**	11056.3	11105.3	11058.6	**11.04%**	6.11%	7.61%	6.89%

效果评估—贡献率

为了验证CHATFUZZ生成种子的有效性，作者通过计算模型产生的有效种子占种子总数的比率，来评估模型对模糊测试的贡献。

Program	ChatFuzz			ChatFuzz-F			ChatFuzz-C			ChatFuzz-CF		
	FQ	imported from AI		FQ	imported from AI		FQ	imported from AI		FQ	imported from AI	
		seed count	ratio		seed count	ratio		seed count	ratio		seed count	ratio
jq	1856	348	**18.75%**	1919	113	5.89%	1850	115	6.22%	1715	137	7.99%
php	17199	1807	**10.51%**	16962	279	1.64%	16485	575	3.49%	17456	436	2.50%
xml	6849	972	**14.19%**	6297	166	2.64%	6437	154	2.39%	6561	149	2.27%
jsoncpp_fuzzer	1321	5	0.38%	1231	1	0.08%	1198	3	0.25%	1283	11	**0.86%**
mujs	11060	3546	**32.06%**	9093	544	5.98%	8756	603	6.89%	9358	400	4.27%
ossfuzz	8298	2995	**36.09%**	7431	1274	17.14%	7183	1275	17.75%	7241	883	12.19%
cflow	1357	31	**2.28%**	1305	8	0.61%	1302	3	0.23%	1253	0	0.00%
lua	2945	1013	34.40%	3301	1319	**39.96%**	3015	853	28.29%	2834	732	25.83%
curl	2235	0	0.00%	2177	6	**0.28%**	2085	0	0.00%	2178	5	0.23%
openssl_x509	2890	0	0.00%	2915	0	0.00%	2982	0	0.00%	2918	0	0.00%
base64	133	3	**2.26%**	124	1	0.81%	128	0	0.00%	138	2	1.45%
md5sum	395	9	**2.28%**	289	3	1.04%	413	2	0.48%	262	3	1.15%
Average			**12.77%**			6.34%			5.50%			4.89%

效果评估—安全性

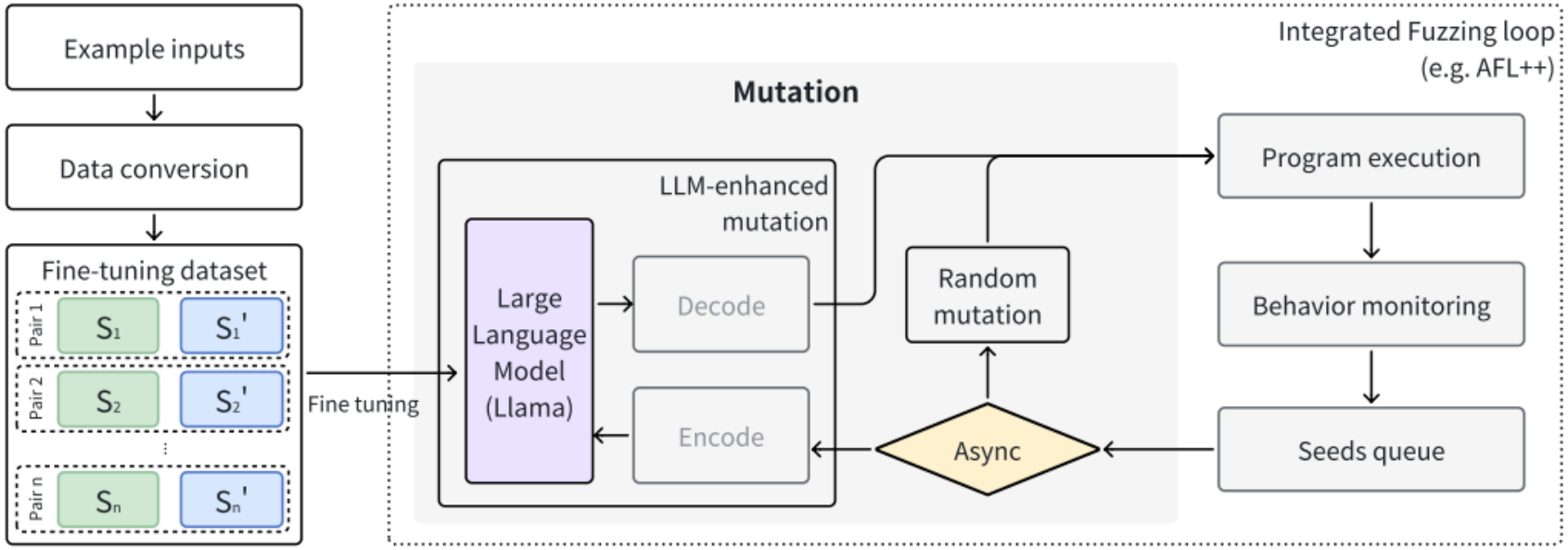
作者还评估了CHATFUZZ在检测bug方面的能力。作者检查了6个程序的程序崩溃输入，并给出了每个被测程序在8小时内检测到的唯一bug的数量。

Program	AFL++	ChatFuzz			
			-F	-C	-CF
jq	-	-	-	-	-
mujs	-	-	-	-	-
ossfuzz	-	-	-	-	-
cflow	2	**3**	**3**	2	1
base64	5	-	-	-	**12**
md5sum	-	-	-	-	-

LLAMAFUZZ: Large Language Model Enhanced Greybox Fuzzing

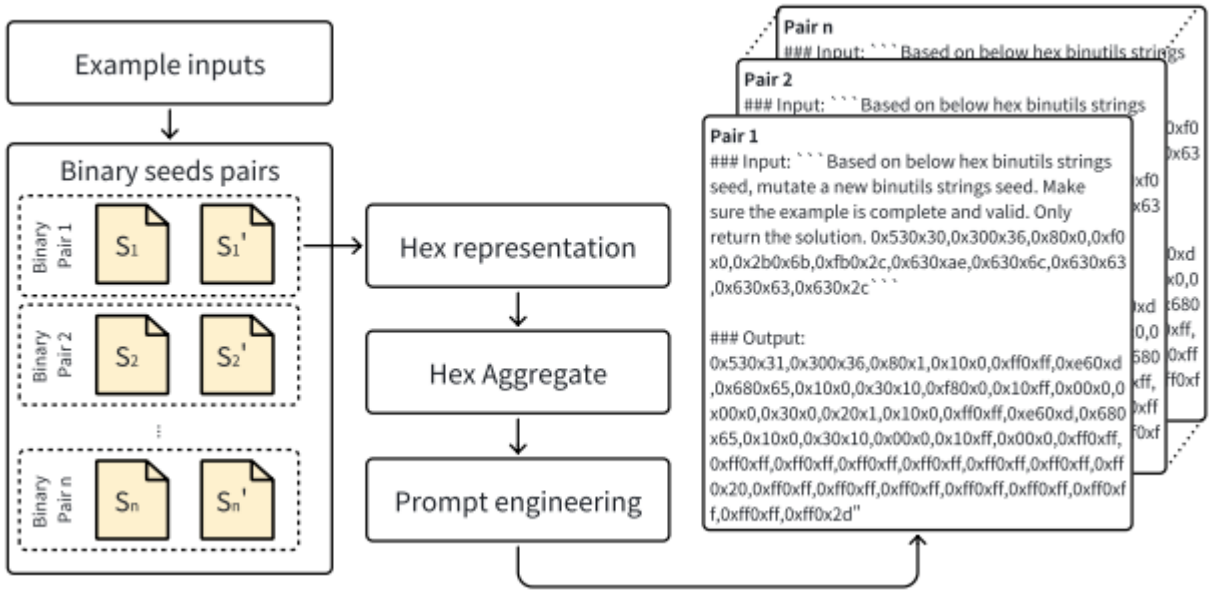
研究背景

同样是为了解决现有变异策略的问题，作者提出使用LLM来增强fuzz的变异过程。作者在不同的数据集上对LLM进行预训练，让模型学习数据格式信息的复杂模式，从而完成对结构化数据的突变。此外，作者对LLM进行了微调，以学习特定的结构化种子模式，并突变出结构化种子。通过这种方式，实现在通用模糊器和专用模糊器之间找到平衡。



微调准备

作者从真实的模糊测试过程中收集数据，对预训练模型进行微调。作者使用这些数据让LLM学习结构化数据的模式和突变，使它能够对给定的种子进行变异，在保持原始结构的同时生成有价值的种子。



数据收集：

- 触发新的分支覆盖
- 触发新的命中次数
- 触发bug

数据预处理：

- 对于二进制输入文件统一转为十六进制编码表示
- 对于文本输入数据不作额外处理

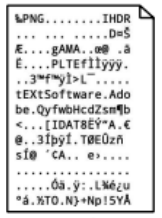
LLM微调

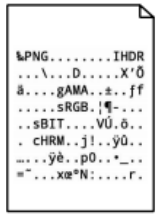
作者通过监督微调来使预训练模型学习数据格式的底层结构和变异方法。进行微调的第一步是准备有效的prompt。在作者设计的prompt中，以十六进制表示当前的结构化数据和所需的突变结果。此外还强调了格式关键字，使LLM能够从预先训练过的知识中获取对格式的一般理解来进行突变。

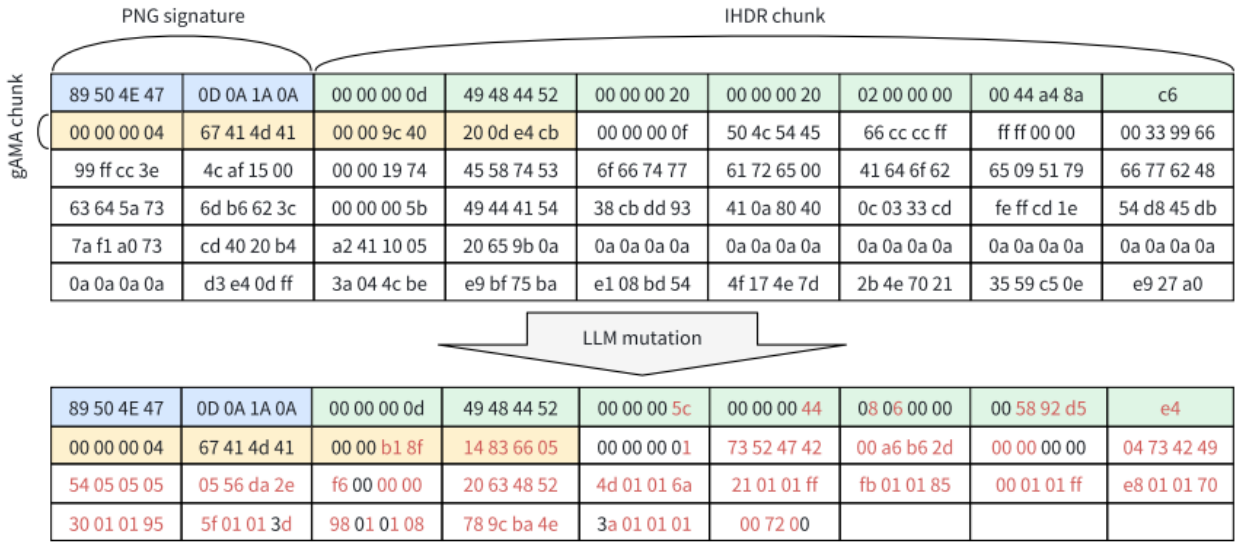
Pair 1

Input: ```Based on below hex binutils strings seed, mutate a new binutils strings seed. Make sure the example is complete and valid. Only return the solution. 0x530x30,0x300x36,0x80x0,0xf0x0,0x2b0x6b,0xfb0x2c,0x630xae,0x630x6c,0x630x63,0x630x63,0x630x2c```

Output:
0x530x31,0x300x36,0x80x1,0x10x0,0xff0xff,0xe60xd,0x680x65,0x10x0,0x30x10,0xf80x0,0x10xff,0x00x0,0x00x0,0x30x0,0x20x1,0x10x0,0xff0xff,0xe60xd,0x680x65,0x10x0,0x30x10,0x00x0,0x10xff,0x00x0,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0x20,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0xff,0xff0x2d"


.PNG file


Mutated .PNG file



89 50 4E 47	0D 0A 1A 0A	00 00 00 0d	49 48 44 52	00 00 00 20	00 00 00 20	02 00 00 00	00 44 a4 8a	c6
00 00 00 04	67 41 4d 41	00 00 9c 40	20 0d e4 cb	00 00 00 0f	50 4c 54 45	66 cc cc ff	ff ff 00 00	00 33 99 66
99 ff cc 3e	4c af 15 00	00 00 19 74	45 58 74 53	6f 66 74 77	61 72 65 00	41 64 6f 62	65 09 51 79	66 77 62 48
63 64 5a 73	6d b6 62 3c	00 00 00 5b	49 44 41 54	38 cb dd 93	41 0a 80 40	0c 03 33 cd	fe ff cd 1e	54 d8 45 db
7a f1 a0 73	cd 40 20 b4	a2 41 10 05	20 65 9b 0a	0a 0a 0a 0a	0a 0a 0a 0a	0a 0a 0a 0a	0a 0a 0a 0a	0a 0a 0a 0a
0a 0a 0a 0a	d3 e4 0d ff	3a 04 4c be	e9 bf 75 ba	e1 08 bd 54	4f 17 4e 7d	2b 4e 70 21	35 59 c5 0e	e9 27 a0

89 50 4E 47	0D 0A 1A 0A	00 00 00 0d	49 48 44 52	00 00 00 5c	00 00 00 44	08 06 00 00	00 58 92 d5	e4
00 00 00 04	67 41 4d 41	00 00 b1 8f	14 83 66 05	00 00 00 01	73 52 47 42	00 a6 b6 2d	00 00 00 00	04 73 42 49
54 05 05 05	05 56 da 2e	f6 00 00 00	20 63 48 52	4d 01 01 6a	21 01 01 ff	fb 01 01 85	00 01 01 ff	e8 01 01 70
30 01 01 95	5f 01 01 3d	98 01 01 08	78 9c ba 4e	3a 01 01 01	00 72 00			

LLM集成

速度对于模糊测试来说是至关重要的，将LLM集成到模糊测试当中，可能会损害整体的吞吐量，并对性能产生负面影响。为了解决这种速度不匹配问题，作者设计了一种异步通信方法来集成 LLM 和 fuzzer。

- **发送通道：**Fuzzer将当前有效种子进行Hex编码即时推送至LLM的输入队列。
- **接收通道：**Fuzzer持续监听LLM的输出队列，如果存在LLM生成的变异种子，就立即抓取进行测试。

- **Fuzzer端：**在LLM进行变异期间，模糊测试器继续执行传统变异策略，不会因等待LLM而暂停工作。
- **LLM端：**维护一个优先级处理队列，仅对最新接收的种子进行变异。当队列满时自动丢弃旧种子，确保LLM始终处理最有价值的数据。

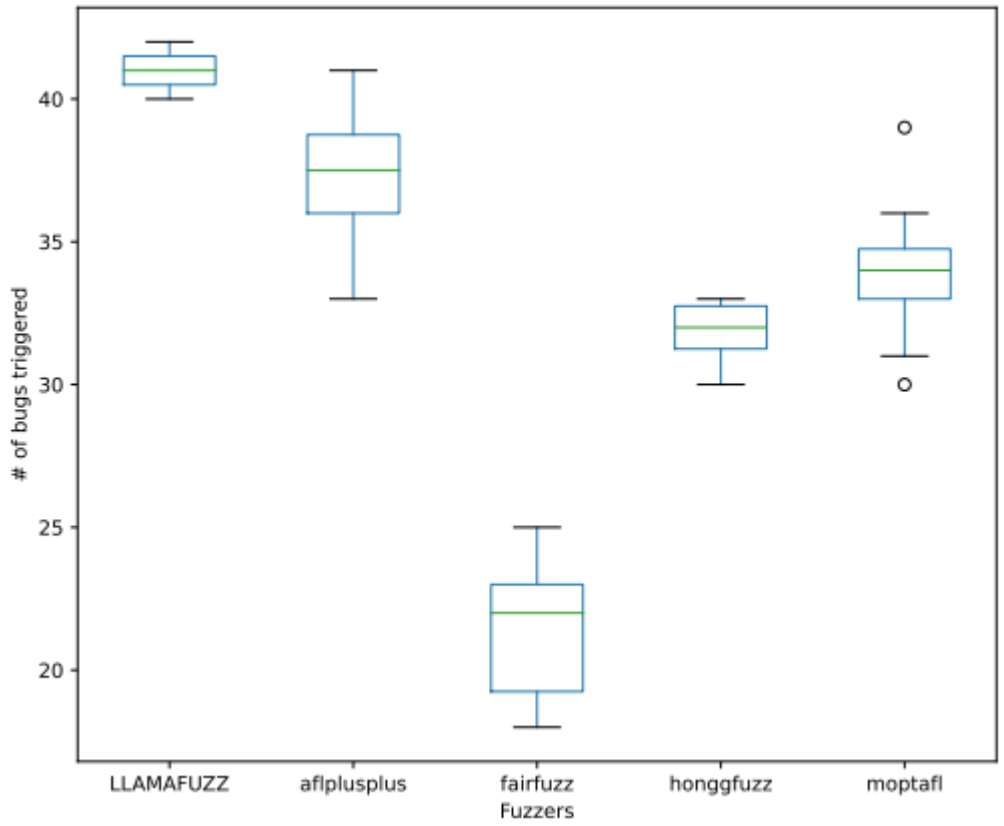
实验评估

作者选取llama2-7b-chat-hf作为LLM。设置max_tokens为4096，以便处理PDF等占用较大存储空间的结构化输入。设置temperature为较低的1.0，保证LLM输出的有效性。作者选择Magma V1.2作为benchmark，包括138个真实漏洞，涵盖除零错误、内存溢出、double-free等多种漏洞类型，程序输入也包含图像、音频、XML、编程语言源码、PDF等多种结构化数据格式。

Parameter	Value
Model	lama2-7b-chat-hf
Max_tokens	4096
Temperature	1.0
Queue_size	30
Benchmark	Magma V1.2

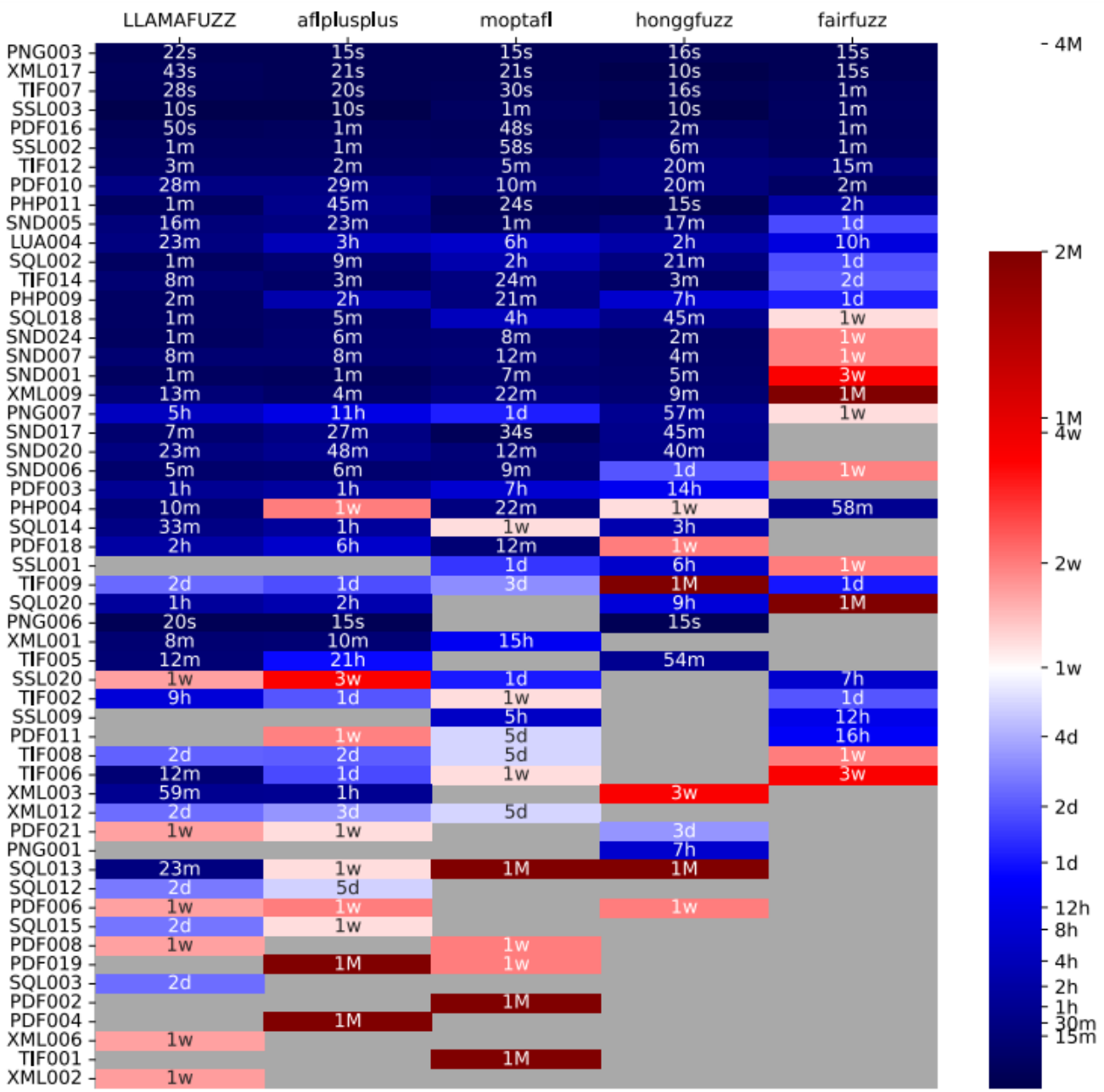
实验评估

运行24h后，LLAMAFUZZ、AFL++、Moptafl、Honggfuzz 和 Fairfuzz 在 Magma V1.2 上触发的bug数目如下图所示。可以看到LLAMAFUZZ触发的bug数目明显优于其他fuzzer。



实验评估

为了评估加入LLM对fuzzer性能的影响，作者统计了漏洞ID及触发所需时间。可以看到，与AFL++相比，LLAMAFUZZ不仅触发了更多漏洞，还显著加快了触发速度。具体而言，在LLAMAFUZZ和AFL++共同触发的43个漏洞中，LLAMAFUZZ在29个漏洞上表现出显著加速，其余漏洞触发时间相近。相较于Moptafl、Honggfuzz和Fairfuzz，LLAMAFUZZ分别在25、23和21个漏洞上实现更快触发。总体结果表明，LLAMAFUZZ在漏洞挖掘效率上具有显著优势。



实验评估

除了验证LLAMAFUZZ在benchmark上的表现，作者选取了一系列正在使用的开源项目进行全面评估，对比LLAMAFUZZ 与AFL++在分支覆盖率上面的表现（afl-cov）。这一步骤可以验证LLAMAFUZZ方法在真实场景下处理多种文件格式的实用价值。在15个测试目标中，LLAMAFUZZ在10个项目上相比AFL++展现出显著覆盖率提升。

Fuzz target	Fuzz object	Branch coverage (avg)		
		LLAMAFUZZ	AFL++	Improv.
bloaty	fuzz_target	5972	5722	4.37%
zlib	zlib_uncompress_fuzzer	384	385	-0.47%
binutils	fuzz_nm	13958	9017	54.81%
	fuzz_objcopy	22318	12494	78.64%
	fuzz_readelf	6576	4437	48.19%
	fuzz_strings	6442	5295	21.65%
grok	grk_decompress_fuzzer	3750	2313	62.12%
kamailio	fuzz_parse_msg	3743	2692	39.06%
	fuzz_uri	1392	1391	0.04%
libavc	avc_dec_fuzzer	9872	9838	0.35%
	mvc_dec_fuzzer	6463	5933	8.94%
	svc_dec_fuzzer	11812	6403	84.48%
openh264	decoder_fuzzer	7394	7396	-0.03%
libhevc	hevc_dec_fuzzer	15154	15122	0.21%
freetype2	freetype2-ftfuzzer	10521	9978	5.45%
Sum		-	-	**27.19%**

北京大学
PEKING UNIVERSITY

谢谢观看

2025.3.25