



代码大作业

南京大学 软件学院 iSE实验室



目录

- 01. 总体要求
- 02. 作业选题
- 03. 联系方式

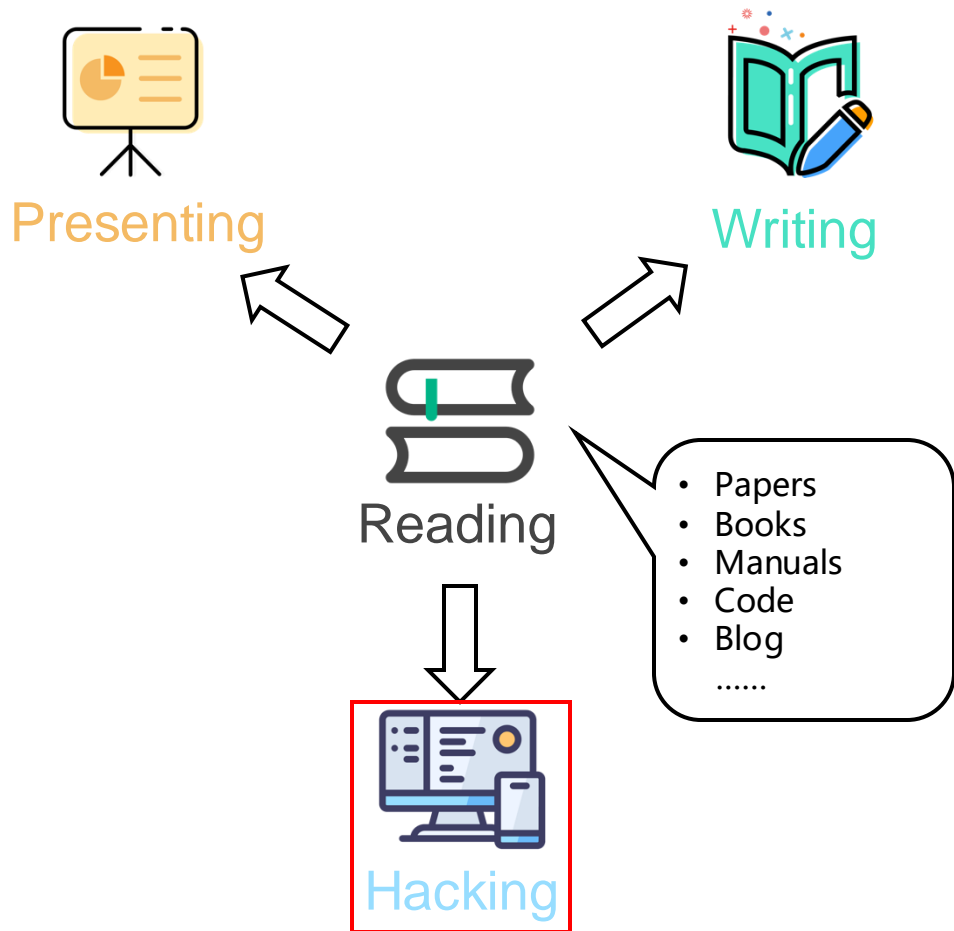


01

总体要求

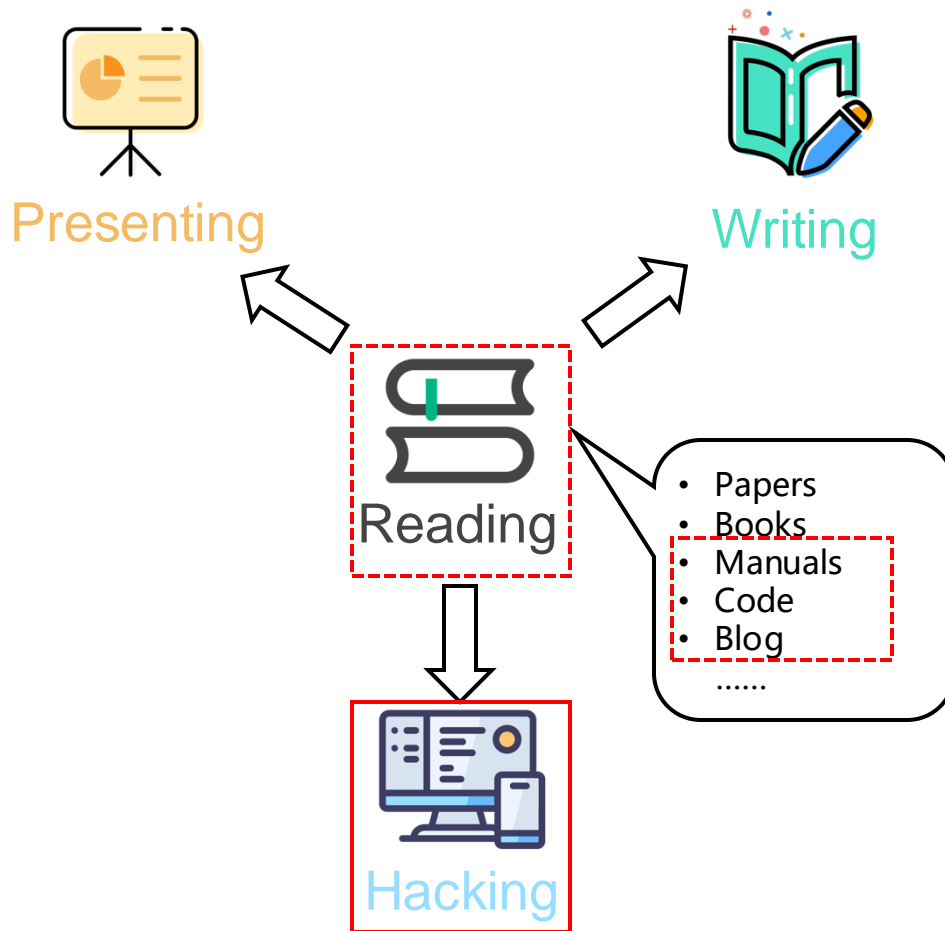


软工研究四维





软工研究四维





作业要求



- 保持学术**诚信**！
 - **约束自身**：作业是对**自己**的训练。各小组应当依靠自身解决问题，主动**不去参考**同选题其他小组的作业
 - **保护成果**：作业代码仓库先私有后公开；作业验收完成前，代码不应该提前公开或私下流通



作业要求



- 遇到**问题**怎么办?
 - **端正心态**: 问题/矛盾是事物进步的根本动力
 - 一切事物中包含的矛盾方面的相互依赖和相互斗争, 决定一切事物的生命, 推动一切事物的发展。——《矛盾论》
 - **学会检索**: 在计算机领域中, 99%以上的问题都有人遇到并解决过
 - 那么剩下1%呢? -- 恭喜你! 你很有可能开启新的技术革命
 - **学会阅读**: RTFM! 读文档、读手册、读代码
 - **学会提问**: 了解问题, 明确需求, 之后向助教提问
 - **善用工具**: 为什么不问问**神奇**的大模型呢?





作业要求



- 大模型保存了大量人类现有的知识（而且非常友好）
- 好的Prompt → 学会提问

用户_HEUudH

我想在shell `ls -alh`之后按照文件大小排序展示文件，应该如何联用sort或者什么其他的工具？

ChatGLM

在shell中使用 `ls -alh` 命令可以列出目录中的所有文件（包括隐藏文件），并显示详细信息（如权限、所有者、大小等），同时以易读的格式显示文件大小（human-readable）。如果你想按照文件大小进行排序，可以结合 `sort` 命令使用。

以下是一个基本的命令组合，用于列出文件并按文件大小排序：

```
sh
1 ls -alh | sort -k5 -h
```

这里的 `sort` 命令使用了两个选项：

- `-k5` 表示按照列表中的第五列进行排序，在 `ls -alh` 的输出中，第五列通常是文件大小。
- `-h` 表示按照易读的格式进行排序，这对于 `ls -alh` 命令输出的文件大小是必要的，因为它们是以K、M、G等格式表示的。

然而，这个命令在某些情况下可能不会按预期工作，因为 `sort` 命令的 `-h` 选项并不总是能够正确处理带有单位（如K、M等）的数字。

一个更健壮的方法是使用 `du` 命令来获取文件大小，然后使用 `sort` 进行排序：

```
sh
1 du -h | sort -h
```

如果你想对当前目录下的所有文件（不包括子目录中的文件）进行排序，可以使用以下命令：

这是一句可以和glm-4-flash交互的代码。现在我想把这段代码做成脚本，并且使content部分替换成\$1（我脚本的第一个参数），但是刚才直接替换发现这样脚本会把content识别成\$1字符串而非\$1里面存储的具体message内容。现在我很苦恼，我应该怎么实现我的需求呢？

记忆已更新

你可以通过使用双引号将整个JSON字符串包裹起来，然后在content部分使用单引号引用参数。这样，shell会正确地将\$1解析为传入的参数。以下是修改后的脚本：

```
zsh
#!/bin/zsh

set -x

curl --location 'https://open.bigmodel.cn/api/paas/v4/chat/completions' \
--header 'Authorization: Bearer e1ec1d8a8d207fc274de19cf6d4bbf38.yUe3HdLxxG940N05' \
--header 'Content-Type: application/json' \
--data '{
  "model": "glm-4-flash",
  "messages": [
    {
      "role": "user",
      "content": "'$1'"
    }
  ]
}'
```

这样，脚本会把\$1的内容正确替换到content字段中，你可以在运行脚本时传入想要的内容。例如：

```
zsh
./your_script.sh "Hello, this is my message"
```

这会把“Hello, this is my message”作为请求中的content发送。

📄 📄 📄 📄 📄



作业要求



- 提交与评分
 - **提交方式：**提交开源链接，先私有后公开
 - 问卷/邮件提交
 - 提交内容：源代码、脚本、日志、文档、设计图、演示视频、模型、数据集.....



作业要求



提交与评分 -- 评分细则（不同课题的标准可能有所不同）

工具实现（100*0.3）	
工具代码：30%	
代码完整、运行流畅	26~30
代码完成绝大部分功能	21~25
代码完成较多功能	16~20
代码完成一半功能	11~15
代码完成小部分功能	6~10
代码完成极少功能	1~5
未提交工具代码	0
分析代码：10%	
分析代码详细、可用性强	6~10
分析代码或流程简陋	1~5
未提交分析代码或说明	0
过程报告：30%	
报告详细且可读性强	21~30
报告充实但可读性较差	11~20
报告比较单薄且可读性较差	1~10
未提交设计文档	0
实验结果：20%	
实验结果展示详细、分析深入、图表清晰	11~20
实验结果展示较少、分析简略、图表简陋	1~10
未提交实验结果	0
代码规范：10%	
命名规范良好，注释详细	6~10
命名规范较差，包含少量注释，	1~5
代码杂乱无章	0

文献综述（100*0.2）	
框架搭建：25%	
能够清晰、完整、有条理地论述研究目标、展示研究框架	21~25
框架性论述有条理，关于研究目标的表述较为清晰	16~20
框架性论述有些混乱，关于研究目标的表述不够清晰	11~15
包含框架性论述但非常混乱，对于研究目标表达不清晰	1~10
在文章中未对研究目标进行框架性论述	0
文献检索：15%	
参考文献数量达标且质量较好	11~15
参考文献数量达标但质量一般	6~10
参考文献数量达标但质量较差	1~5
参考文献数量小于最低限制且质量低下	0
文献分类：15%	
分类合理、符合正交原则	11~15
分类较为合理，存在少数重叠与过分类现象	6~10
分类不太合理，存在大量重叠与过分类现象	1~5
未进行文献分类	0
文献分析：30%	
分析精彩，展现了独到的理解，能够给出有价值的建议	26~30
分析详细且表述清晰	21~25
分析详细，但表述不够清晰、结构较为混乱	16~20
存在较少的分析，表述不够清晰、结构较为混乱	11~15
存在较少的分析，但表达不清晰、结构十分混乱	6~10
流水账式表达，只是对他人观点的堆砌	1~5
未提交实验结果	0
综述格式：15%（助教视情况打分）	
结构 5：文章分节有序、结构合理	5
引用 5：引用符合规范、整齐有序	5
版面 5：文章页数达标、排版有序	5



02

作业选题



选题总览



- 经典测试技术方向
 - 模糊测试 & 变异测试
 - 选题1：覆盖率引导的变异式模糊测试工具
 - 选题2：基于大模型的模糊驱动生成工具
 - 源码警告识别
 - 选题3：仓库级源码警告数据集收集技术
 - 选题4：基于提示工程的智能化源码警告识别实证研究



选题总览

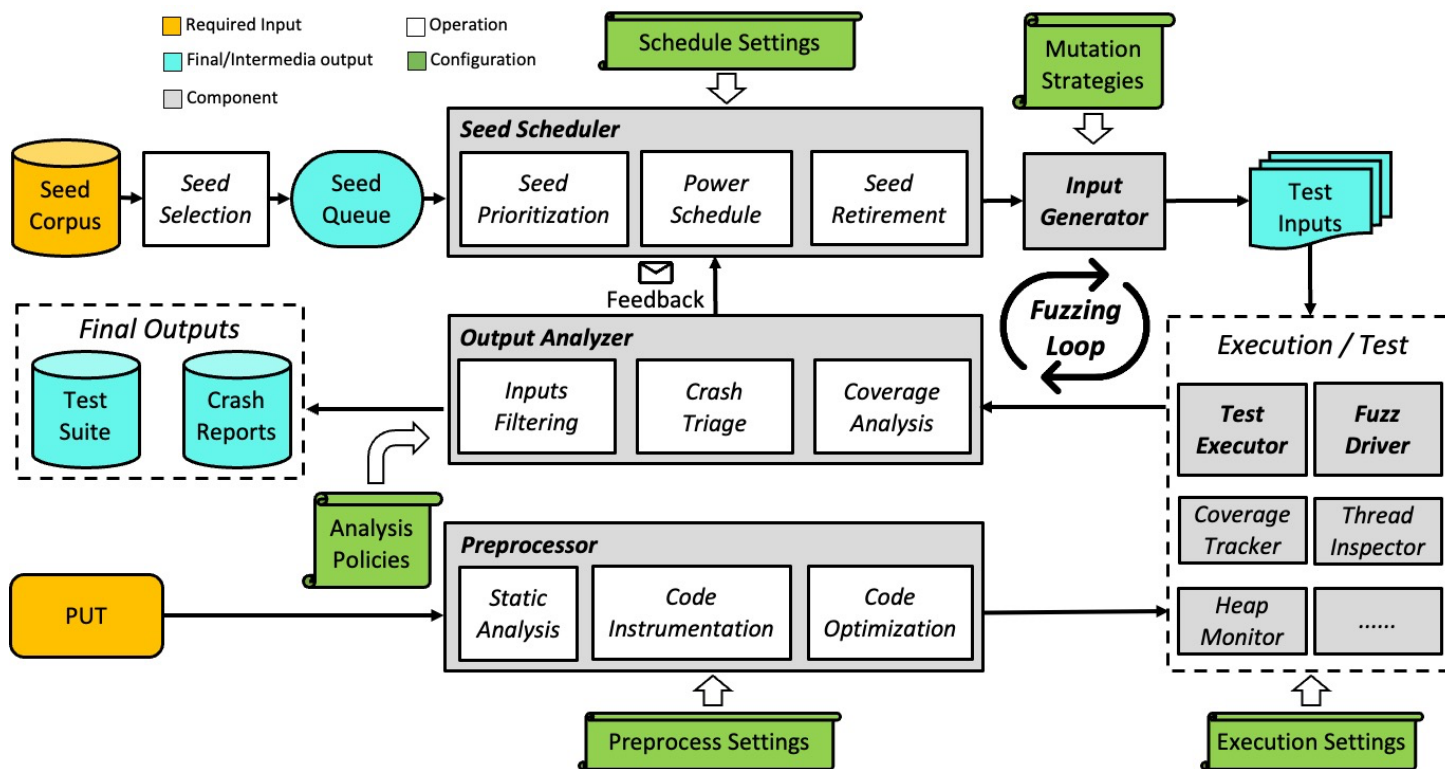


- 智能软件测试方向
 - 选题1：面向xxx场景的深度学习模型测试技术
 - 选题2：基于等价融合算子的深度学习框架差分测试技术
- 图形界面测试方向
 - 选题1：大模型引导的场景感知GUI探索测试
 - 选题2：基于机械臂视觉识别的非侵入式GUI界面自动化探索



- 选题1：覆盖率引导的变异式模糊测试工具
 - 去年的Demo ([Github](#)、[Gitee](#))
 - 实现语言：Java or Python
 - 实现六个组件：插装组件 (afl-cc)、测试执行组件、执行结果监控组件、变异组件、种子排序组件、能量调度组件、评估组件
 - 在10个模糊目标 (Fuzz Target) 上运行24h，将覆盖率结果可视化成统计图 (评估)，模糊目标详情见[Github仓库](#)

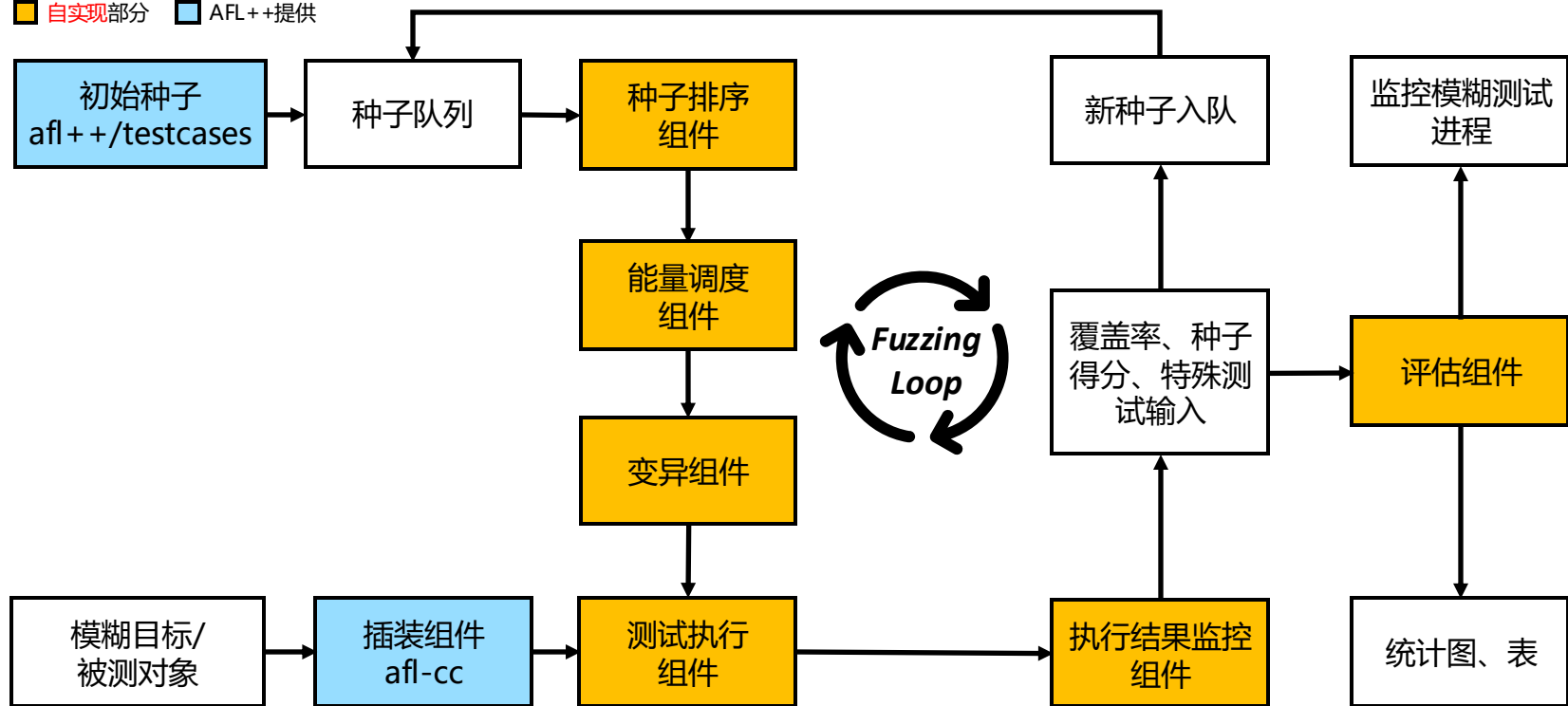
• 选题1：覆盖率引导的变异式模糊测试工具





• 选题1：覆盖率引导的变异式模糊测试工具（工具架构）

■ 自实现部分 ■ AFL++提供

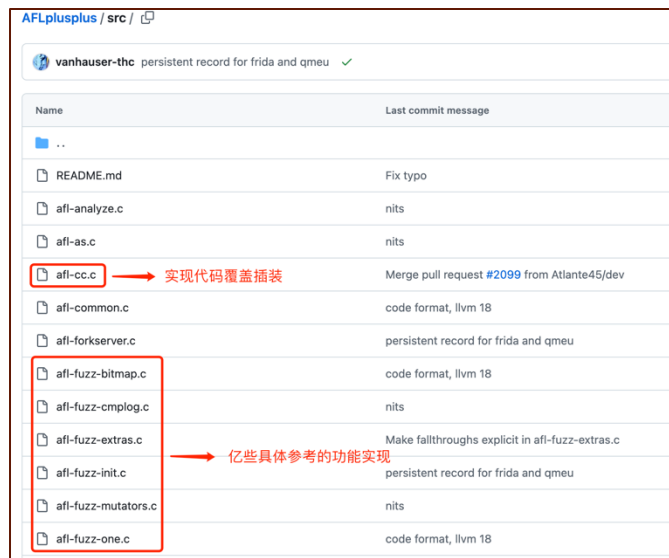
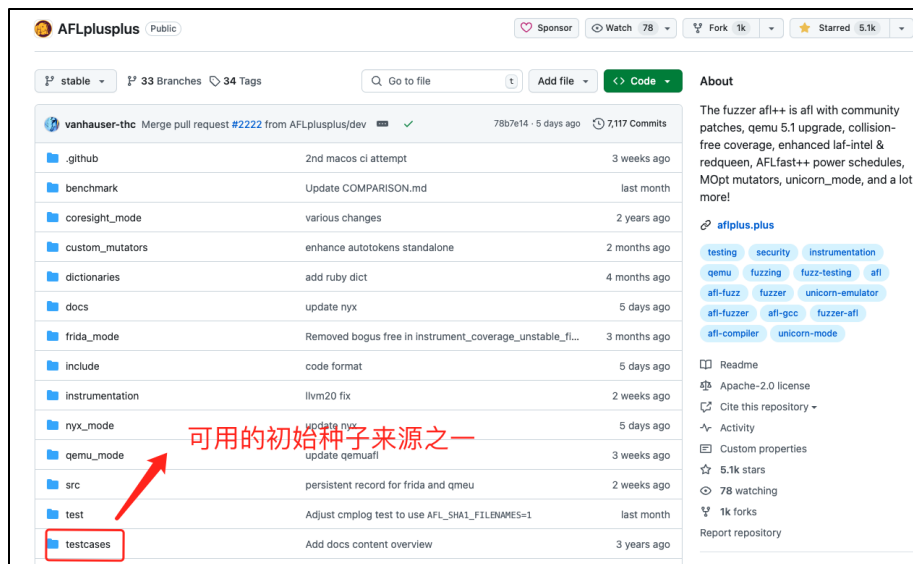




经典测试技术方向



- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
- 主要参考SOTA模糊测试工具[AFL++](#)，但简化

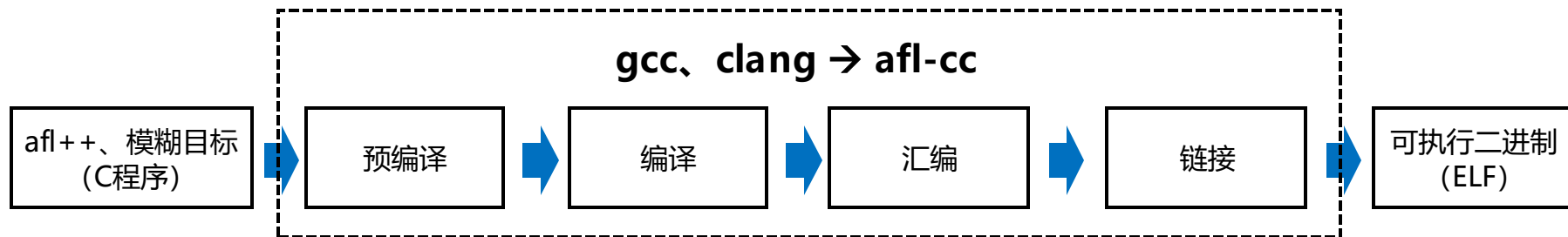




- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - 主要参考SOTA模糊测试工具[AFL++](#)，但简化
 - **种子排序组件**：除按入队顺序选择外，至少再实现一种非随机排序
 - **能量调度组件**：启发式调度，参考afl-fuzz.h#calculate_score()
 - **变异组件**：实现AFL变异算子，详细说明见[AFL文档#Stage Progress](#)
 - bitflip、arith、interest、havoc、splice → 加分项：基于语法的算子
 - **测试执行组件**：创建子进程，运行模糊目标，记录执行时间/次数。
 - **执行结果监控组件**：能够记录覆盖率、执行速度，保存特殊测试用例
 - **评估组件**：至少实现覆盖率曲线分析



- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - **实验环境**：使用docker，在ubuntu 22.04镜像上搭建环境
 - 模糊目标的编译与插装：libtool、build-essential (make)、cmake、python3、clang、llvm、file、binutils (nm, objdump)
 - 基本方法：阅读项目文档，运行构建指令 \rightleftarrows 发现问题（问大模型）



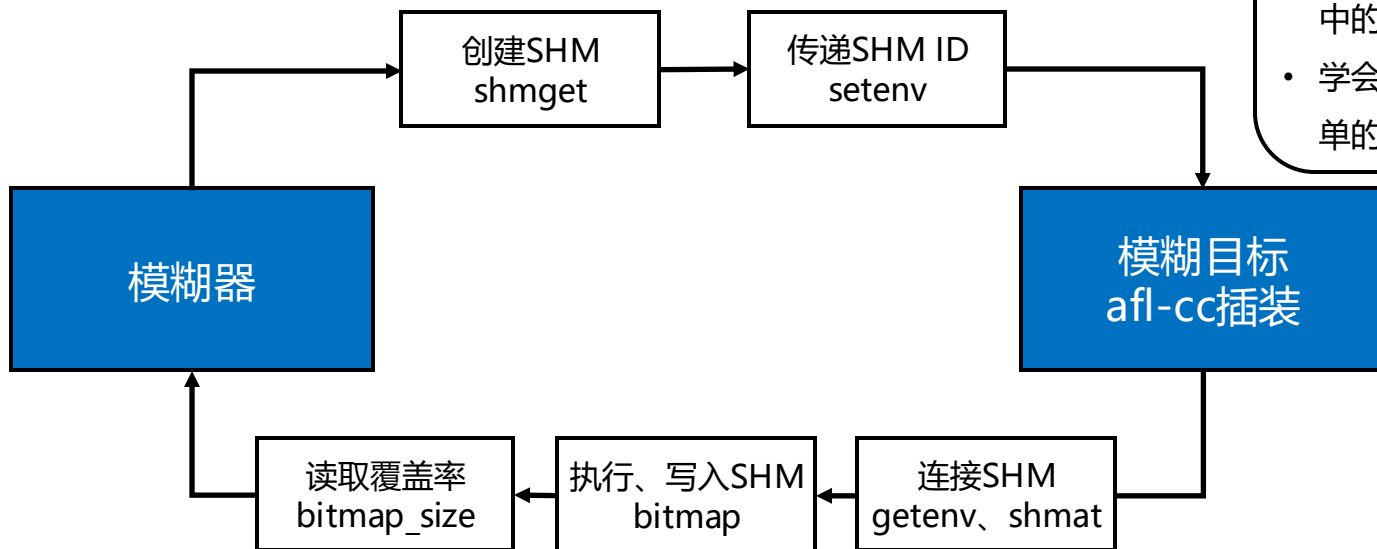


- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - **种子排序组件**：afl-fuzz.c#main(), perform_dry_run()之后
 - **能量调度组件**：afl-fuzz.h#calculate_score()
 - **变异组件**：afl-fuzz.h#fuzz_one(), afl-fuzz.h#fuzz_one_original()
 - **测试执行组件**：afl-fuzz.h# fuzz_run_target()



• 选题1：覆盖率引导的变异式模糊测试工具（实现要点）

• 执行结果监控组件：基于afl-cc的覆盖率监控原理



Tips

- 阅读C的shm家族
- 寻找Java/Python中的对应实现
- 学会做减法，从简单的Toy程序写起



- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - 执行结果监控组件：打印日志，显示模糊测试当前状态

```
[*] Fuzzing test case #12859 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=100, weight=1, favorite=0, was_fuzzed=0, e
xec_us=5598, hits=0, map=97, ascii=0, run_time=0:19:35:36)...
[*] Fuzzing test case #13048 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=400, weight=1, favorite=1, was_fuzzed=1, e
xec_us=4016, hits=0, map=229, ascii=0, run_time=0:19:35:40)...
[*] Fuzzing test case #13156 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=300, weight=1, favorite=0, was_fuzzed=1, e
xec_us=4015, hits=0, map=123, ascii=0, run_time=0:19:35:44)...
[*] Fuzzing test case #13209 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=100, weight=1, favorite=0, was_fuzzed=0, e
xec_us=4195, hits=0, map=163, ascii=0, run_time=0:19:35:48)...
[*] Fuzzing test case #13276 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=400, weight=1, favorite=0, was_fuzzed=1, e
xec_us=4119, hits=0, map=197, ascii=0, run_time=0:19:35:52)...
[*] Fuzzing test case #13622 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=400, weight=1, favorite=1, was_fuzzed=1, e
xec_us=5299, hits=0, map=204, ascii=0, run_time=0:19:35:57)...
[*] Fuzzing test case #14025 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=100, weight=1, favorite=0, was_fuzzed=0, e
xec_us=4313, hits=0, map=101, ascii=0, run_time=0:19:36:01)...
[*] Fuzzing test case #14127 (14622 total, 0 crashes saved, state: in progress, mode=explore, perf_score=100, weight=1, favorite=0, was_fuzzed=0, e
xec_us=4478, hits=0, map=186, ascii=0, run_time=0:19:36:05)...
[*] Fuzzing test case #14140 (14623 total, 0 crashes saved, state: in progress, mode=explore, perf_score=600, weight=1, favorite=1, was_fuzzed=1, e
xec_us=4412, hits=0, map=364, ascii=0, run_time=0:19:36:11)...
[*] Fuzzing test case #14156 (14623 total, 0 crashes saved, state: in progress, mode=explore, perf_score=100, weight=1, favorite=0, was_fuzzed=0, e
xec_us=5967, hits=0, map=450, ascii=0, run_time=0:19:36:18)...
```



- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - 执行结果监控组件：保存模糊测试输出

```
total 2088
drwx----- 6 root root      191 Oct 12 08:06 ./
drwxr-xr-x  7 root root       76 Oct 11 12:27 ../
-rw-----  1 root root       37 Oct 11 12:27 cmdline
drwx-----  2 root root        6 Oct 11 12:27 crashes/
-rw-----  1 root root      118 Oct 12 08:06 .cur_input
-rw-----  1 root root    32256 Oct 12 08:04 fuzz_bitmap
-rw-----  1 root root     313 Oct 11 12:27 fuzzer_setup
-rw-----  1 root root    1345 Oct 12 08:05 fuzzer_stats
drwx-----  2 root root        6 Oct 11 12:27 hangs/
-rw-r--r--  1 root root        0 Oct 11 12:27 is_main_node
-rw-----  1 root root 1002673 Oct 12 08:06 plot_data
drwx-----  3 root root   835584 Oct 12 08:04 queue/
drwx-----  2 root root     138 Oct 11 12:31 .synced/
```

运行 target 使用的命令行

保存的产生 crash 的测试输入

临时文件，当前正在执行的输入

亿些 fuzzer 配置和内部状态

模糊测试统计

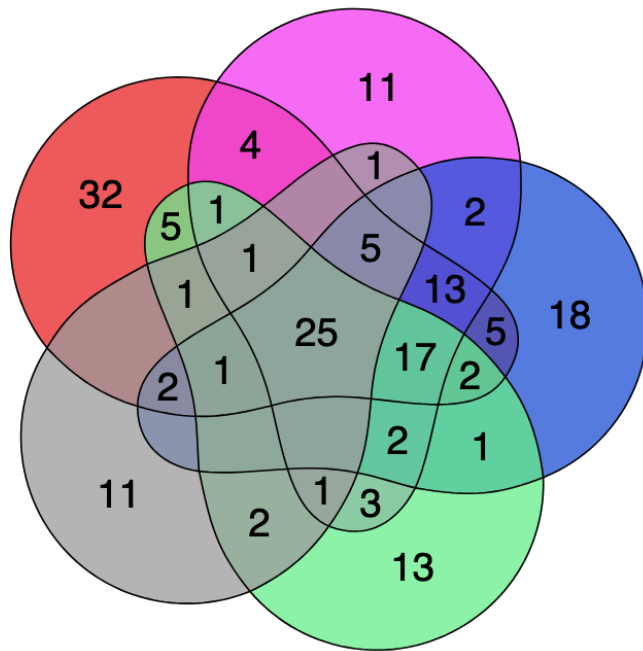
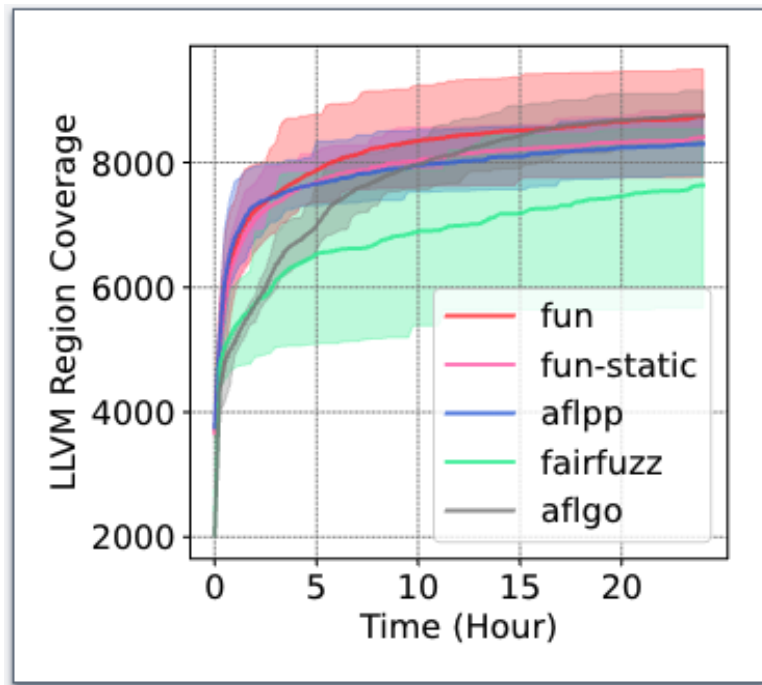
保存的产生覆盖增量的测试输入



- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - 执行结果监控组件：保存模糊测试输出

```
root@ubuntu:~/vulner/aflpp-4.21/tcpdump/output/master# head -n20 plot_data
# relative_time, cycles_done, cur_item, corpus_count, pending_total, pending_favs, map_size, saved_crashes, saved_hangs, max_depth, execs_per_sec,
total_execs, edges_found
61, 0, 0, 447, 447, 292, 12.01%, 0, 0, 2, 271.49, 19089, 3868
66, 0, 0, 449, 449, 292, 12.01%, 0, 0, 2, 269.92, 20463, 3870
71, 0, 0, 450, 450, 292, 12.01%, 0, 0, 2, 268.34, 21850, 3870
76, 0, 0, 451, 451, 292, 12.01%, 0, 0, 2, 269.49, 23208, 3870
81, 0, 0, 453, 453, 292, 12.07%, 0, 0, 2, 271.56, 24592, 3889
86, 0, 0, 453, 453, 292, 12.07%, 0, 0, 2, 271.86, 25967, 3889
92, 0, 0, 453, 453, 292, 12.07%, 0, 0, 2, 271.12, 27347, 3889
97, 0, 0, 460, 460, 292, 12.11%, 0, 0, 2, 269.07, 28726, 3901
102, 0, 0, 461, 461, 292, 12.11%, 0, 0, 2, 267.66, 30048, 3901
107, 0, 0, 462, 462, 292, 12.11%, 0, 0, 2, 266.60, 31405, 3901
112, 0, 0, 463, 463, 292, 12.11%, 0, 0, 2, 269.18, 32787, 3901
117, 0, 0, 463, 463, 292, 12.11%, 0, 0, 2, 269.29, 34162, 3901
122, 0, 0, 463, 463, 292, 12.11%, 0, 0, 2, 270.12, 35557, 3901
127, 0, 0, 466, 466, 292, 12.12%, 0, 0, 2, 269.31, 36943, 3903
133, 0, 0, 468, 468, 292, 12.12%, 0, 0, 2, 270.69, 38332, 3904
138, 0, 0, 468, 468, 292, 12.12%, 0, 0, 2, 48.31, 39657, 3904
143, 0, 0, 470, 470, 292, 12.13%, 0, 0, 2, 272.18, 41041, 3907
148, 0, 0, 471, 471, 292, 12.13%, 0, 0, 2, 273.25, 42414, 3908
153, 0, 0, 472, 472, 292, 12.14%, 0, 0, 2, 268.47, 43795, 3911
```


- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - **评估组件：**模糊器运行结果统计图表示例





- 选题1：覆盖率引导的变异式模糊测试工具（实现要点）
 - 评估组件：模糊器运行结果统计图表示例

TID	Target	AFL++	FUNFUZZ		FUNFUZZ-Static		FairFuzz		AFLGo	
		#Reg.	#Reg.	Δ (%)	#Reg.	Δ (%)	#Reg.	Δ (%)	#Reg.	Δ (%)
T01	cxxfilt	4546.2	4557.2	0.2	4545.5	0.0	4362.1	-4.1	4163.1	-8.4
T02	nm-new	5609.1	5731.7	2.2	5632.8	0.4	5480.3	-2.3	5265.3	-6.1
T03	objdump	8297.1	9080.2	9.4	8400.5	1.3	7631.1	-8.0	8751.9	5.5
T04	readelf	14333.1	14959.0	4.4	14342.4	0.1	14535.1	1.4	14603.9	1.9
T05	djpeg	3441.5	3611.5	4.9	3496.1	1.6	3431.5	-0.3	3271.8	-4.9
T06	readpng	1617.7	1685.4	4.2	1650.2	2.0	1634.7	1.1	1611.3	-0.4
T07	mjs	2709.6	2824.5	4.2	2763.2	2.0	2524.4	-6.8	2574.1	-5.0
T08	mutool	3876.8	3881.2	0.1	3877.2	0.0	3865.3	-0.3	—	—
T09	tcpdump	8385.9	9926.7	18.4	9002.4	7.4	10160.6	21.2	7265.6	-13.4
T10	xmllint	17482.0	18610.6	6.5	17898.4	2.4	18616.2	6.5	15071.7	-13.8



- **选题1：覆盖率引导的变异式模糊测试工具**
 - **提交内容：**源代码、脚本、开发日志、项目文档、演示视频、Docker镜像（可访问的Dockerhub链接/Dockerfile）
 - **开发日志：**Markdown文件，devlog.md
 - 开发过程中的进度安排、任务分配、碰到的难题和解决的过程等。
 - **项目文档：**Markdown文件，README.md
 - 工具的运行方法，[示例](#)
 - 项目的设计方案（架构、流程、类层次）和使用方法
 - 注意描述的逻辑性，善用“总-分”结构



- 选题2：基于大模型的模糊驱动生成工具（探索向）
 - 模糊驱动是测试执行的入口，是模糊测试的重要组件
 - main()、JUnit @Test、LLVMFuzzerTestOneInput

```
C readpng.c x
funfuzz > mg-fun > libpng > repo > contrib > libtests > C readpng.c > ...
102     png_read_end(png_ptr, info_ptr);
103     png_destroy_read_struct(&png_ptr, &info_ptr, NULL);
104     free(row);
105     free(display);
106     return 1;
107 }
108
109 int
110 main(void)
111 {
112     /* Exit code 0 on success. */
113     return !read_png(stdin);
114 }
115
```

```
C libpng_read_fuzzer.cc M x
funfuzz > mg-fun > libpng > repo > contrib > oss-fuzz > C libpng_read_fuzzer.cc > ...
99
100 // Entry point for LibFuzzer.
101 // Roughly follows the libpng book example:
102 // http://www.libpng.org/pub/png/book/chapter13.html
103 extern "C" int LLVMFuzzerTestOneInput(const uint8_t* data, size_t size) {
104     if (size < kPngHeaderSize) {
105         return 0;
106     }
107
108     std::vector<unsigned char> v(data, data + size);
109     if (png_sig_cmp(v.data(), 0, kPngHeaderSize)) {
110         // not a PNG.
111         return 0;
112     }
113
114     PngObjectHandler png_handler;
115     png_handler.png_ptr = nullptr;
116     png_handler.row_ptr = nullptr;
117     png_handler.info_ptr = nullptr;
118     png_handler.end_info_ptr = nullptr;
119 }
```



- 选题2：基于大模型的模糊驱动生成工具（探索向）
 - 模糊驱动的本质是一段代码，而LLM则具备强大的代码生成能力

用户_HEUudH

你知道Libfuzzer么？你知道Libfuzzer的LLVMTestOneInput么？你能给出一个LLVMTestOneInput的实例么？

ChatGLM

是的，我了解LibFuzzer。LibFuzzer是LLVM项目的一部分，它是一个覆盖率引导的模糊测试引擎，可以用来进行有效的错误检测。它通过与编译器集成，使得可以创建对特定函数进行模糊测试的轻量级驱动程序。

LLVMFuzzerTestOneInput 是使用LibFuzzer时需要实现的一个函数，这个函数会被模糊测试引擎重复调用，每次调用都会传入一个作为输入的字符串。这个函数的目的是测试你的代码是否可以正确处理这个输入，而不会出现如崩溃、内存泄漏等问题。

下面是一个简单的 LLVMFuzzerTestOneInput 示例。这个例子中的函数会尝试解析一个简单的整数，并将其加到全局计数器中。

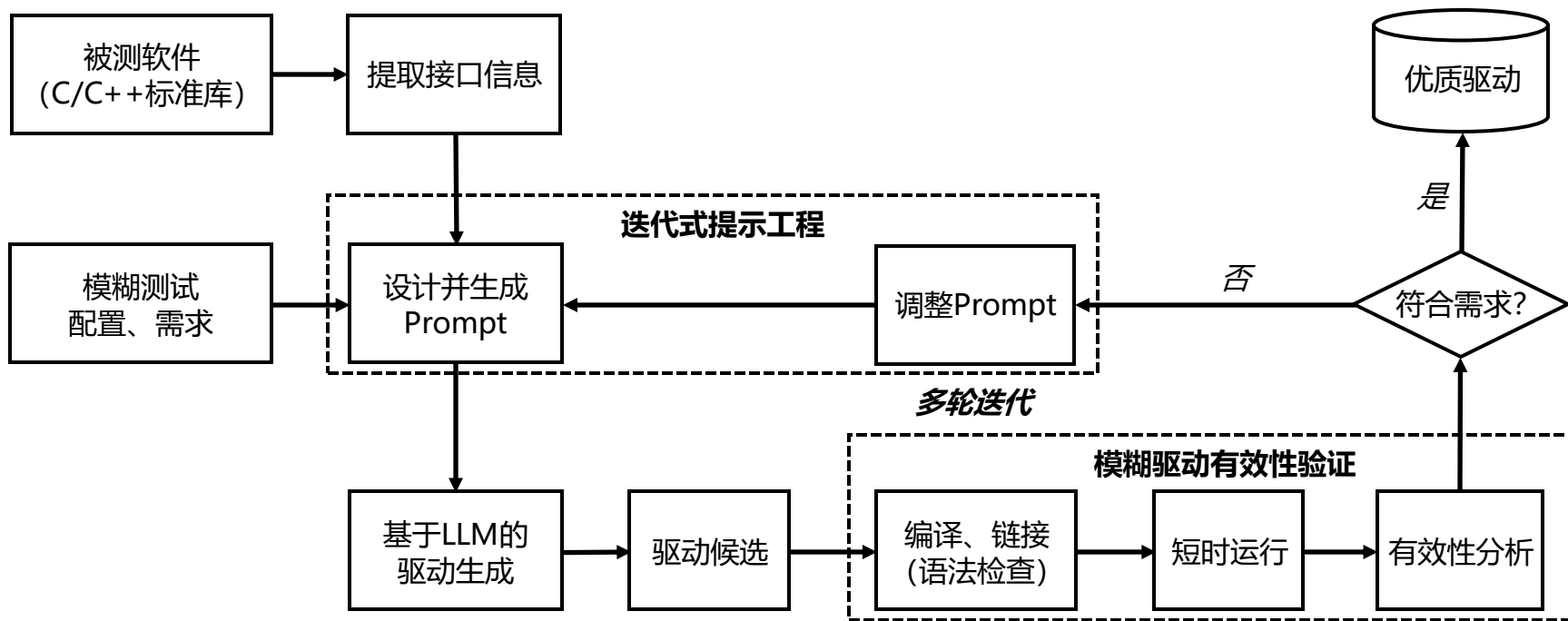
```
c 复制
1  #include <stdint.h>
2  #include <stddef.h>
3
4  // 全局变量，用于统计调用次数
5  static int g_count = 0;
6
7  // LLVMFuzzerTestOneInput 的实现
8  extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
```

```
#include <stdint.h>
#include <stddef.h>
static int g_count = 0;
extern "C" int LLVMFuzzerTestOneInput(const
uint8_t *Data, size_t Size) {
    if (Size < sizeof(int)) {
        return 0;
    }
    int num = 0;
    memcpy(&num, Data, sizeof(num));
    g_count += num;
    return 0;
}
extern "C" int main(int argc, char **argv) {
    LLVMFuzzerTestOneInput(NULL, 0);
    return 0;
}
```



• 选题2：基于大模型的模糊驱动生成工具（探索向）

- 探索如何引导大模型生成模糊驱动，实现相应的功能和评估流程^[1,2]



[1] Zhang C, Zheng Y, Bai M, et al. How Effective Are They? Exploring Large Language Model Based Fuzz Driver Generation[C]//Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis. 2024: 1223-1235.

[2] Lyu Y, Xie Y, Chen P, et al. Prompt Fuzzing for Fuzz Driver Generation[J]. arXiv preprint arXiv:2312.17677, 2023.



- 选题2：基于大模型的模糊驱动生成工具（探索向）
 - **提交内容**：源代码、脚本、开发日志、项目文档、演示视频、Docker镜像（可访问的Dockerhub链接/Dockerfile）
 - **开发日志**：Markdown文件，devlog.md
 - 开发过程中的进度安排、任务分配、碰到的难题和解决的过程等。
 - **项目文档**：Markdown文件，README.md
 - 工具的运行方法，[示例](#)
 - 项目的设计方案（架构、流程、类层次）和使用方法
 - 注意描述的逻辑性，善用“总-分”结构



- 选题3：仓库级源码警告数据集收集技术
 - 现有研究提出了一些自动警告数据集构建方法以收集源码警告数据集，这些方法本质上是通过跟踪警告演变历史进而收集数据集。
 - 现有技术仍然存在以下问题
 - 警告匹配策略是不精确，导致收集到的警告数据集存在很多噪音；
 - 现有的开发语言多种多样，但现有数据集仅聚焦少数静扫工具（如SpotBugs和Infer）和少数开发语言上（Java和C/C++）。
 - **目标：**构建一个面向多源异构源码静态扫描工具的警告基准数据集构建。



- 选题3：仓库级源码警告数据集收集技术
 - **技术流程**：调研开源静扫工具 → 收集开源项目 → 跟踪警告演变历史 → 警告报告预处理 → 警告汇总 → 整理实验结果
 - **提交内容**：源代码、工具类型映射（表格）、警告数据集及其对应的属性



- 选题4：基于提示工程的智能化源码警告识别实证研究

- 提示工程（Prompt Engineering）是指设计和优化输入给人工智能模型（如ChatGPT）的指令，以确保模型生成预期的输出。通过构建精准、详细的提示，能够提升模型的理解和响应能力，从而实现高效的人机交互。
- 提示工程已经在基于大模型的软件工程任务（如：代码摘要和自动缺陷修复）。然而，提示工程在源码警告识别上的性能还未被充分调研。
- **目标：**系统调研提示工程在源码警告识别的可行性，并比较不同的提示模版在源码警告识别上的优缺点。



- 选题4：基于提示工程的智能化源码警告识别实证研究
 - **技术流程**：调研分析源码警告数据集（助教提供） → 调研提示模板 → 设计提示策略 → 部署/访问大模型 → 开展结果评估
 - **提交内容**：源代码、所有原始实验结果、针对实验结果的可视化，分析和总结



- 选题1：面向xxx场景的深度学习模型测试技术
 - **选题背景：**在智能模型的测试中，不同场景的领域特性会对测试方法的设计带来而外的要求，例如自动驾驶场景下无意义的噪声添加并不能代表现实生活中会遇到的天气、光照等真实问题。
 - **选题目标：**掌握分析特定场景的能力，建立某一个特定场景下测试需求，设计并实现可用的测试方法，场景自选，可大（图像场景、点云场景等）可小（门禁人脸识别场景、课堂教学语音生成）。



- 选题1：面向xxx场景的深度学习模型测试技术
 - **数据来源和被测对象**：本方向提供可选的待测模型和数据集，各位小组也可以根据小组的特殊场景自选其他模型和数据集。
 - **提交内容**：源代码、数据集、项目文档（README.md）、演示视频。



- **选题2：基于等价融合算子的深度学习框架差分测试技术**
 - **选题背景：**深度学习框架为深度学习模型的开发提供了必要算子的开源接口，例如Conv1D、Conv2D等。为了避免框架的缺陷被引入到深度学习模型中，需要对深度学习框架开展测试来发现其中的缺陷。在深度学习框架的测试中，深度学习模型是测试输入，为了检测深度学习框架是否出现了缺陷，需要利用差分测试，对比多个等价的深度学习模型来分析是否出现了不一致的异常行为。
 - **选题要求：**探索如何利用等价的融合算子生成等价的深度学习模型，梳理所选框架下的融合算子列表，并至少实现其中较为常用的5组融合算子，生成测试数据、开展深度学习框架的差分测试



- 选题2：基于等价融合算子的深度学习框架差分测试技术
 - **数据来源和被测对象**：在深度学习框架的内部，每个接口对应的算子运算可以是基础算子，例如`tf.add`，也可以由多个其他基础算子组成的复合算子，例如`tf.einsum`。执行复合算子等效于执行其包含的每个基础算子。本题目提供了部分框架下示例融合算子，选择本题目的小组可以自行选择被测的深度学习框架。
 - **提交内容**：源代码、数据集、项目文档（README.md）、演示视频。



- **选题1：大模型引导的场景感知GUI探索测试**
 - **选题背景：**传统GUI 自动化探索测试生成大量测试结果，并从中发现待测应用中的缺陷。然而，已有的探索策略缺乏对于真实业务逻辑的理解，从而难以生成复杂交互操作序列以全面对待测应用的功能点进行覆盖。在自动化探索中引入对测试场景的感知能够从用户视角出发，做到更有效的测试覆盖。
 - **选题要求：**设计一种基于大语言模型引导的场景感知GUI 自动化探索测试技术，通过大模型理解测试场景、生成并执行测试操作，将基于场景领域知识融入自动化GUI测试，实现更加可靠且可解释的GUI测试生成。



- 选题1：大模型引导的场景感知GUI探索测试
 - **实现要点**：测试场景的感知与理解、基于场景的测试操作生成、基于场景的应用状态验证、完整的自动化GUI测试探索 workflow、交互界面设计与实现、**领域知识引导的GUI场景感知测试生成（加分项）**
 - **提交内容**：源代码、项目文档、演示视频



- 选题2：基于机械臂视觉识别的非侵入式GUI界面自动化探索
(不超过3组，选题前与虞助教邮件沟通)
 - **选题背景：**传统GUI 自动化探索测试往往使用侵入式测试框架完成，通过侵入移动应用安装文件来捕获 GUI 元素的标识符，进而完成测试任务。然而，当前的方法存在诸多问题，如用户隐私问题、平台不兼容问题等。
 - **选题要求：**利用机械臂模拟真正用户对app的使用，利用计算机视觉技术，完成待测应用及GUI元素的识别与分析，进一步结合已有的 GUI 界面自动化探索策略，从而实现基于机械臂视觉识别的非侵入式GUI界面自动化探索。



图形界面测试方向



- 选题2：基于机械臂视觉识别的非侵入式GUI界面自动化探索
(不超过3组，选题前与虞助教邮件沟通)
 - **实现要点：**目标识别、机械臂操作、GUI界面探索策略、机械臂调度、交互界面设计与实现
 - **提交内容：**源代码、项目文档、演示视频



03

联系方式



联系方式



- 经典测试技术方向
 - 钱瑞祥, qrx_at@163.com
 - 葛修婷, 1683245057@qq.com
- 智能软件测试方向
 - 刘佳玮, jw.liu@smail.nju.edu.cn
- 图形界面测试方向
 - 虞圣呈, yusc@smail.nju.edu.cn
 - 凌毓辰, yuchenling@smail.nju.edu.cn

养成先想再问的习惯!



zychen@nju.edu.cn
fangchunrong@nju.edu.cn

Thank you!