Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

大模型指导的内核模糊测试

硕士研究生 杨语航

2025年06月22日

问题回溯

- 存在的问题
  - 语气较为平淡，语言不够精炼
  - 看ppt时间较多
- 相关内容
  - **2025.01.19** 杨语航 《面向操作系统的模糊测试》
  - **2024.09.03** 张浩然 《大模型赋能的模糊测试用例生成》
  - **2024.06.09** 谢宁 《基于突变的模糊测试》

内容提要

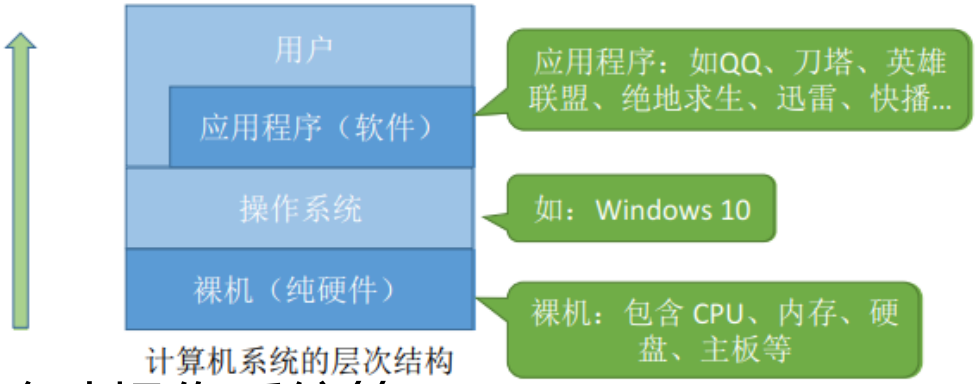
- 预期收获
- 题目内涵解析
- 研究背景与意义
- 研究历史与现状
- 知识基础
- 算法原理
  - **KernelGPT**
  - **ECG**
- 特点总结与工作展望
- 参考文献

预期收获

- 预期收获
  - 了解内核模糊测试的基础知识和特殊挑战
  - 理解内核模糊测试中LLM的应用策略和独特价值
  - 掌握两种和LLM结合的模糊测试技术原理与方法

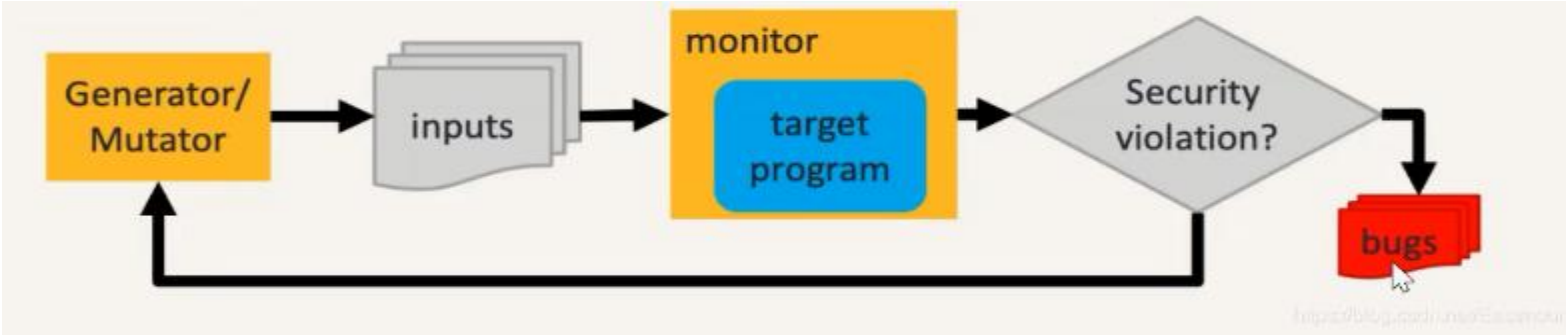
内涵解析与研究目标

- 题目内涵解析（大模型指导的内核模糊测试）
  - 内核：操作系统的核心，管理计算机硬件资源，提高接口供上层软件使用
  - 模糊测试：根据一定规则，产生大量随机数据然后输入到程序中并监视异常情况出现，以此发现可能的程序错误和漏洞



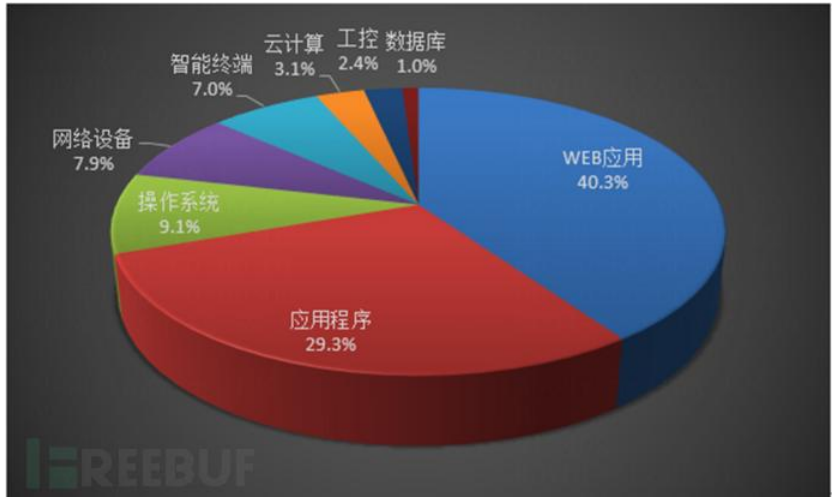
计算机系统的层次结构

- 研究目标
  - 以系统内核为研究对象，如**Linux**、**Windows**、移动操作系统等
  - 研究领域包括种子生成 、种子选择 、引导突变和系统调用规范生成

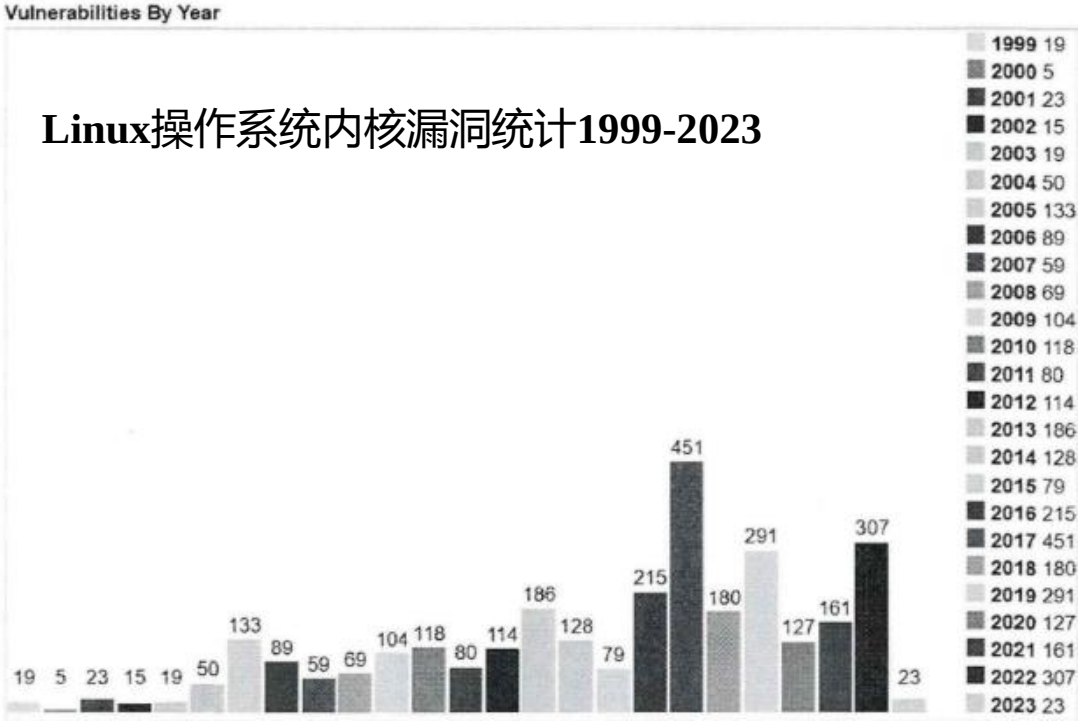


研究背景与意义 操作系统安全越发严峻

- 研究背景
  - 重要性：操作系统是计算机系统的核心，其漏洞被恶意利用，威胁整个计算机安全
  - 复杂性：**1991年Linux Kernel 源码行数1万，2022年5.19版本LOC超过3300万行**
- 研究意义：
  - 目前各类操作系统正被广泛使用在个人电脑、移动设备和企业服务器等，因此增强操作系统稳定性显得异常重要



2023年漏洞影响对象占比



研究历史 大模型指导的内核模糊测试

Hao等人提出基于静态分析与内核驱动编程系统调用描述自动生成工具SyzDescribe，建模内核模块初始化流程与驱动-设备对象关联机制，精准识别系统调用接口及参数类型；采用模块化分析框架与间接调用解析技术

Zhang等人提出基于LLM的嵌入式操作系统模糊测试工具ECG，通过静态分析提取系统调用约束，然后利用LLM生成符合的程序输入；生成阶段，采用多阶段输入生成策略，根据执行反馈不断优化测试用例，大幅提高代码覆盖率和漏洞发现率

2023

2024

2023

Hu等人提出利用LLM自动化生成Linux内核系统调用规范工具KernelGPT，通过迭代分析内核源代码，推断syscall标识符值、参数类型和依赖关系，生成高质量的syscall描述，显著提高Syzkaller代码覆盖率，并检测到24个未知内核漏洞，其中11个分配了CVE编号

2024

Xu等人提出基于LLM的定向灰盒模糊测试ISC4DGF，先利用精炼LLM对项目信息、代码、CVE详情及补丁做语义提炼，生成结构化提示；生成LLM创建多样化种子输入，基于编译成功率、格式合规性等筛选最优初始种子

内核模糊测试

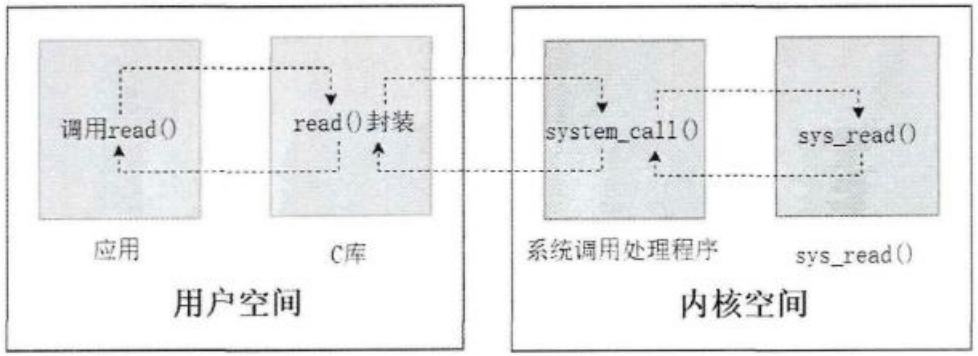
- LLM指导的模糊测试
- 传统模糊方法

研究现状

- 传统内核模糊测试方法
  - 测试用例输入
    - 与静态分析结合：借助静态分析提供针对性输入，优化输入空间
    - 与符号执行结合：帮助解决“状态爆炸”的问题，但会受到诸如浮点运算精度的限制
  - 系统调用规范生成
    - 手动编写，如**Syzkaller**中用**Syzlang**编写的系统调用规范
    - 静态分析，如**SyzDescribe**
  - 反馈引导策略
    - 覆盖率引导：目标是最大化代码覆盖率
- **LLM**在内核模糊测试中的应用
  - 常见有输入生成、种子选取、引导变异策略、系统调用规范生成

知识基础

- 系统调用（Syscall）
  - 定义：用户空间程序与内核交互的接口，本质是函数
  - 作用：用户空间程序通过Syscall请求内核执行特定操作，如文件操作open\write 、进程控制fork 、通信exit等
- 参数类型（Type）
  - 定义：系统调用中各个参数的数据类型
  - 示例：常见有基本数据类型、结构体等
  - **open(fd fd, const char *pathname, int flags);**
- 标识符（Identifier）
  - 定义：区分不同系统调用操作的特定值
  - 示例：**ioctl(fd fd,cmd const [DRM_IOCTL_MSM_SUBMITQUEUE_CLOSE], ...);**



知识基础

- 系统调用规范（System-Call Descriptions）
  - 定义：使用领域特定语言编写的对系统调用的详细说明
  - 作用1：详细记录了系统调用所需的参数类型、返回值、约束条件等许多信息
  - 作用2：描述了系统调用如何管理资源，以及不同系统调用之间的依赖关系
  - 作用3：帮助内核模糊测试器生成有效的测试用例
  - 示例：使用Syzlang对ioct1编写的部分系统调用描述

```
ioctl$VHOST_SET_VRING_ADDR(fd fd_vhost,
                           cmd const[VHOST_SET_VRING_ADDR],
                           arg ptr[in, vhost_vring_addr])

VHOST_SET_VRING_ADDR = 1076408081

vhost_vring_addr {
 index flags[vhost_vring_index, int32]
 flags int32[0:1]
  desc_user_addr ptr64[out, array[int8]]
  used_user_addr ptr64[out, array[int8]]
  avail_user_addr ptr64[out, array[int8]]
  log_guest_addrs flags[kvm_guest_addrs, int64]
}
```

【ASPLOS】
KernelGPT: Enhanced Kernel Fuzzing via Large Language Models

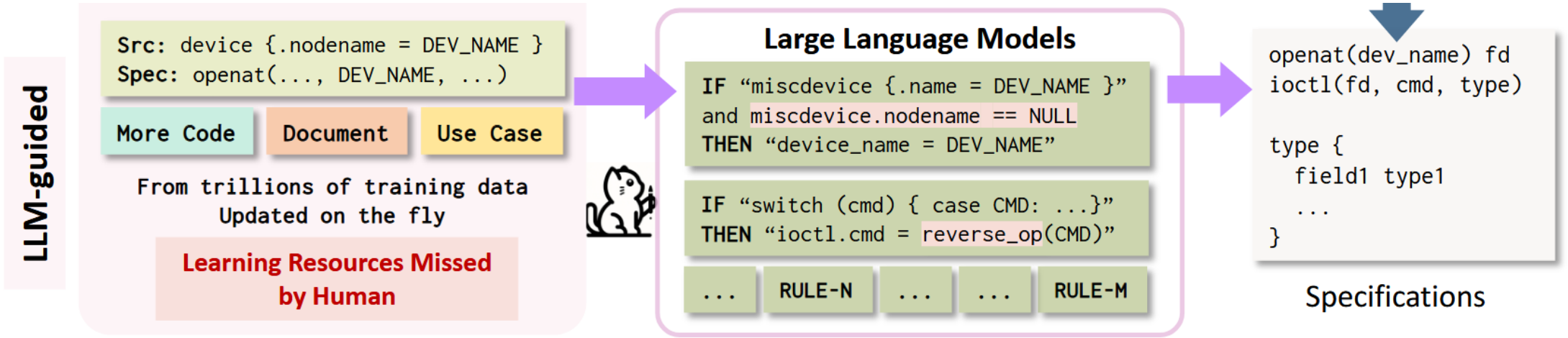
KernelGPT TIPO

T	目标	自动生成**syscall**规范，提高内核模糊测试效率
I	输入	**Linux**内核代码库、定位的操作处理函数
P	处理	**1.** 规范生成，利用**LLM**多阶段迭代分析内核源代码，推断**syscall**规范 **2.** 规范验证修复，基于验证工具对规范语法检查并反馈**LLM**修正
O	输出	系统调用规范***n**、漏洞检测报告

P	问题	现有模糊器需要手动或静态分析得到**syscall**规范，自动化程度低
C	条件	基于LLM的代码理解和生成能力，需要对Linux内核有足够的与训练数据和知识
D	难点	**1.** 如何缓解大模型幻觉在规范生成中的影响 **2.** 如何规避大模型的上下文窗口限制
L	水平	**ASPLOS 2023 （CCF-A）**

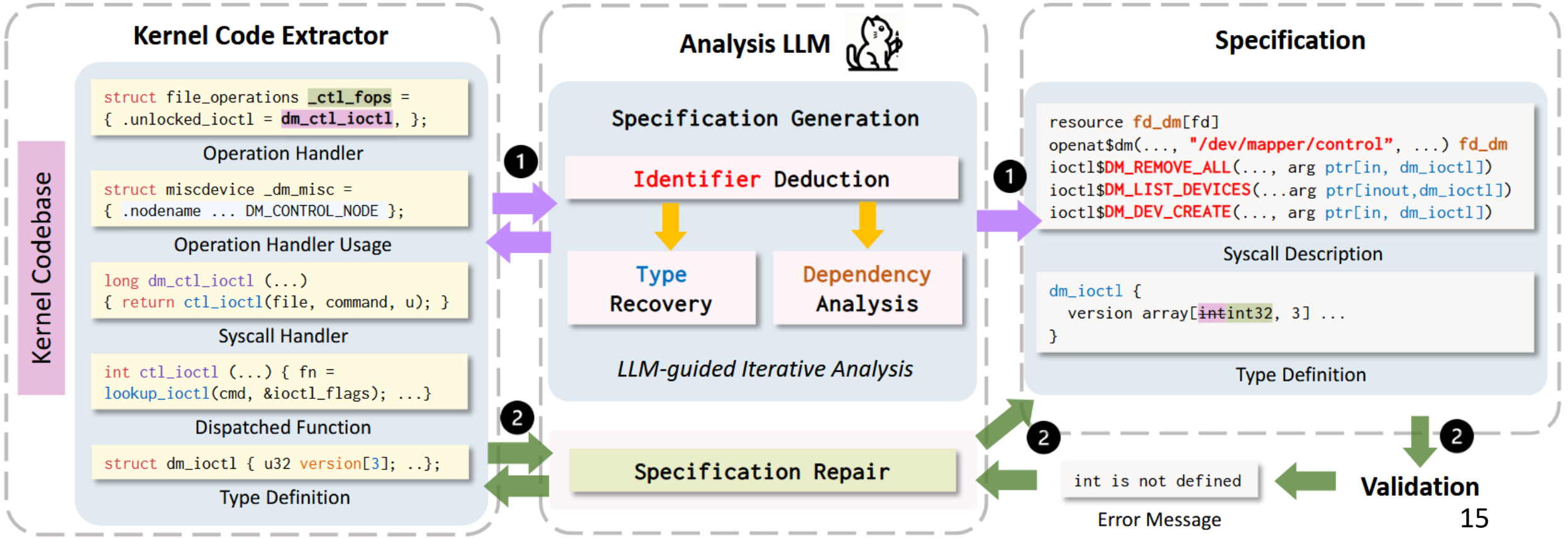
KernelGPT 创新性引入

- 现有方法推断系统调用的局限性
  - 建模不完整：内核代码变动频繁、实现多样化，传统方法适用性差
  - 可读性差：静态分析生成的规范一般难以理解
  - 文本利用率差：现有工具无法有效利用代码中的注释、文档和上下文信息
- **LLM指导内核模糊测试**
  - 动态适应内核代码更新，自动化程度高
  - 可高效利用代码中文本信息，生成的规范可读性强



KernelGPT 算法原理图

- 思路分析
  - 规范生成：LLM多阶段迭代分析对内核代码进行理解和推断，生成系统调用规范
  - 规范验证与修复：结合验证工具对生成规范进行语法和语义检查，并由LLM修复



算法原理 规范生成

- 规范生成（Specification Generation）
  - 标识符推断
    - 目标：自动识别系统调用中用于区分具体操作的关键参数
    - 操作
      - 定位的操作处理函数及相关代码片段输入LLM
      - 少量示例引导LLM推断标识符值
      - 若推理需要采用未知函数，提取对应函数递归分析，直到获取完整标识符集合或达到最大迭代次数

```
ioctl$NEW(fd fd_msm, cmd const[DRM_IOCTL_MSM_SUBMITQUEUE_NEW],
          arg ptr[inout, drm_msm_submitqueue])
ioctl$CLOSE(fd fd_msm, cmd const[DRM_IOCTL_MSM_SUBMITQUEUE_CLOSE],
            arg ptr[in, msm_submitqueue_id])
```

ioctl中cmd是标识符值

算法原理 规范生成

- 规范生成（Specification Generation）
  - 类型恢复
    - 目的：推断系统调用参数的数据类型和语义关系
    - 操作
      - 提取相关函数提交给LLM，推断参数类型
      - 对于基本数据类型，提取内核中相关定义
      - 对于嵌套类型，先标记为未知类型，再进一步分析
  - 依赖关系分析
    - 目的：识别系统调用间的数据依赖关系
    - 操作
      - 将相关函数源代码以及函数返回值，提交给LLM
      - 分析函数返回值的使用场景，判断返回值是否作为其他系统调用输入

```
## Source Code
struct vfio_pci_hot_reset_info {
    int32   count;
    struct vfio_pci_dependent_device    devices[];
};
```

```
## Specification Generated by Static Analysis
vfio_pci_hot_reset_info {
  field_0 int32
  field_1 array[vfio_pci_dependent_device]
}
```

```
## Specification Generated by LLM
vfio_pci_hot_reset_info {
  count len[devices, int32]
  devices ptr[inout, array[vfio_pci_dependent_device]]
}
```

静态分析和LLM生成的系统调用规范

算法原理 规范验证与反馈

- 规范验证与修复（**Specification Validation and Repair**）
  - 验证机制
    - 自动验证工具检查生成规范的语法和简单的语义规则
    - 将发现的错误信息和需要修复的规范形成提示，提供给**LLM**
  - 修复机制
    - 通过少样本修复示例，**LLM**进行迭代修复
    - 修复过程中涉及的未知函数和类型，由代码提取器自动提取

多阶段迭代生成——大模型上下文限制
规范验证与修复——大模型幻觉

实验设计 数据资源

- 实验配置
  - 基础模糊器：**Syzkaller**
  - 测试对象：**Linux kernel v6.7**
  - 硬件配置：**96核、512GB内存的工作站**
  - **LLM：GPT-3.5、GPT-4、GPT-4o**
  - 实验设置：**4个QEMU虚拟机，每个虚拟机配置2个CPU核心，运行24小时模糊测试**
  - 操作系统：**Ubuntu 20.04.5 LTS 64 bit OS**
- 对比方法：
  - **Syzkaller (开源工具)：默认模糊器，手动编写的syscall规范**
  - **SyzDescribe (2023)：基于静态分析的syscall规范生成工具**

实验设计 实验结果

- **RQ1：新漏洞发现**
  - 新发现的漏洞：**24**
  - 已确认的漏洞：**21**
  - 已修复的漏洞：**12**
  - 分配CVE编号：**11**

Crash with new specs	New	Confirmed	Fixed	CVE	Syzkaller	SyzDescribe
kmalloc bug in ctl_ioctl	✓	✓	✓	CVE-2024-23851	×	×
kmalloc bug in dm_table_create	✓	✓	✓	CVE-2023-52429	×	×
KASAN: slab-use-after-free Read in cec_queue_msg_fh	✓	✓	✓	CVE-2024-23848	×	×
ODEBUG bug in cec_transmit_msg_fh	✓	✓	✓		×	×
WARNING in cec_data_cancel	✓	✓	✓		×	×
INFO: task hung in cec_claim_log_addrs	✓	✓			×	×
general protection fault in cec_transmit_done_ts	✓	✓	✓		×	×
kernel BUG in btrfs_get_root_ref	✓	✓	✓	CVE-2024-23850	×	×
general protection fault in btrfs_update_reloc_root	✓	✓			×	×
zero-size vmalloc in ubi_read_volume_table	✓	✓	✓	CVE-2024-25739	×	×
UBSAN: array-index-out-of-bounds in rds_cmsg_recv	✓	✓	✓	CVE-2024-23849	×	×
memory leak in ubi_attach	✓	✓		CVE-2024-25740	×	×
memory leak in posix_clock_open	✓	✓	✓	CVE-2024-26655	×	×
memory leak in __ip6_append_data	✓	✓			×	×
possible deadlock in dvb_demux_release	✓				×	×
INFO: task hung in __rq_qos_throttle	✓				×	×
WARNING in usb_ep_queue	✓	✓		CVE-2024-25741	×	×
memory leak in dvb_dmxdev_add_pid	✓	✓			×	×
memory leak in dvb_dvr_do_ioctl	✓				×	×
general protection fault in dvb_vb2_expbuf	✓	✓	✓	CVE-2024-50291	×	×
general protection fault in cleanup_mapped_device	✓	✓	✓	CVE-2024-50277	×	×
WARNING in vb2_core_reqbufs	✓	✓			×	×
BUG: corrupted list in vep_queue	✓	✓			×	×
divide error in uvc_queue_setup	✓	✓			×	×
Total	**24**	**21**	**12**	**11**	**0**	**0**

实验设计 对比实验

- **RQ2：** 生成规范的质量和效率
  - **KernelGPT推断得到的系统调用规范总量显著高于其他方法**
  - **KernelGPT多阶段迭代生成和规范验证模块对生成有效系统调用规范效果显著**
  - **KernelGPT产生532个syscall规范需要4.7h，共处理 556 万输入token，生成 40 万输出token，总花费34美元**

	SyzDescribe		KernelGPT	
	# Syscalls	# Types	# Syscalls	# Types
Driver	146	168	288	170
Socket	N/A	N/A	244	124
Total	146	168	532	294

新生成的系统调用规范总数

		SyzDescribe	KernelGPT
	# Total	# Incomplete	# Valid	# Valid (Fixed)
Driver	278	75	20	70 (30)
Socket	81	66	N/A	57 (12)
Total	359	141	20	127 (42)

为驱动程序/套接字程序生成的有效规范

算法总结 KernelGPT

- 算法流程
  - 规范生成：标识符推断、参数类型恢复、系统调用依赖关系分析
  - 规范验证和修复：通过验证工具对生成的规范进行语法和语义检查，并通过大模型迭代修复
- 算法优势
  - 自动化推断系统调用规范，规范可读性高
  - 可产生多样性、有效的系统调用规范
- 算法不足
  - **LLM**可能产生不准确或不真实的信息，即使引入了验证和修复机制

【 EMSOFT 】

ECG: Augmenting Embedded Operating System Fuzzing via LLM-based Corpus Generation

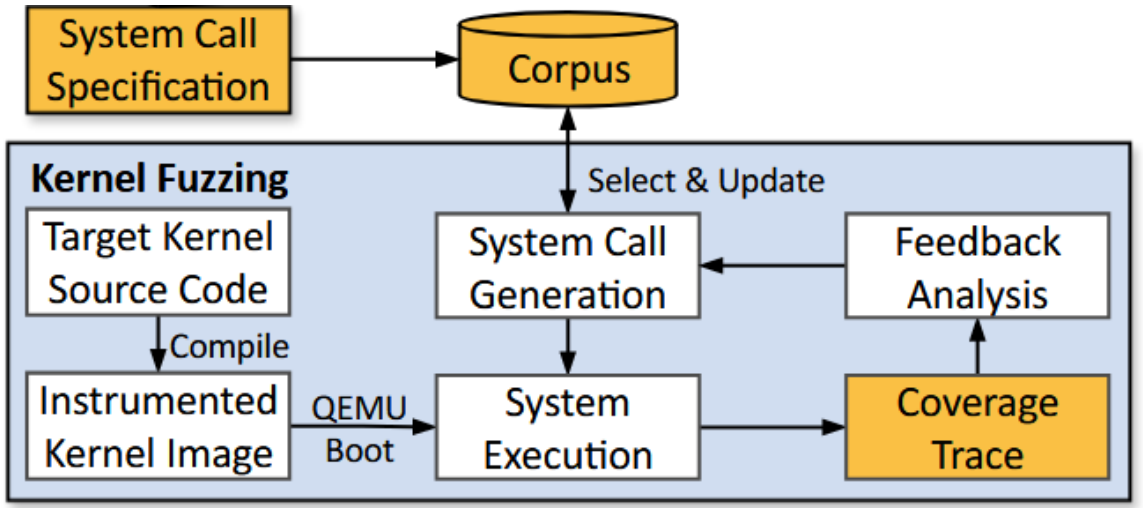
ECG TIPO

T	目标	通过**LLM**自动构建系统调用规范，并生成高质量测试用例
I	输入	嵌入式操作系统源代码、文档
P	处理	**1.** 规范构建阶段，静态分析提取约束，**LLM**生成与精炼可执行代码，以此进行规范提取 **2.** 输入生成阶段，采用多阶段生成策略，基于方向引导变异
O	输出	系统调用规范、漏洞检测报告

P	问题	**1.** 嵌入式系统调用规范缺乏、文档稀缺 **2.** 执行反馈难以被传统**Fuzzer**有效利用，测试效率低
C	条件	源代码和文档需要较为完善，具备本地运行**LLM**的计算资源
D	难点	**1. LLM**难以直接生成正确的嵌入式系统调用规范 **2.** 如何提高执行反馈的利用效率
L	水平	**EMSOFT 2024 （CCF-B）**

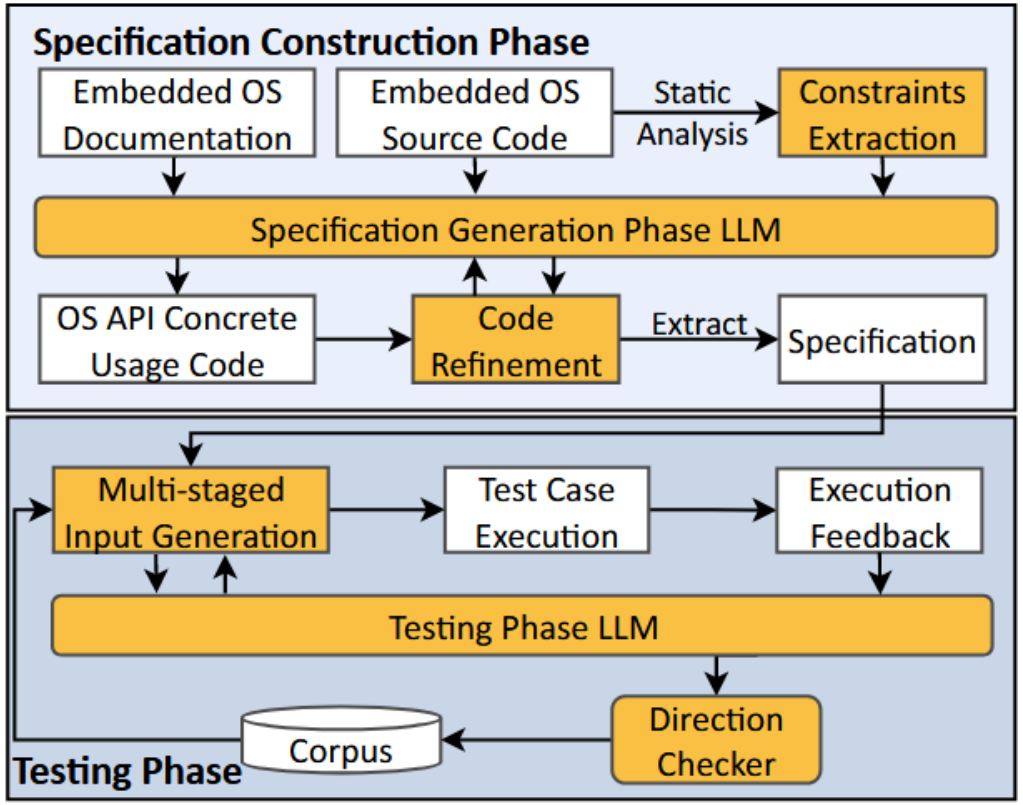
ECG 创新性引入

- 传统内核模糊测试器
  - 背景：嵌入式系统属于下游操作系统，兼具多样化和定制化的特性，文档相对缺少
  - 问题1：嵌入式系统特定模块的系统调用规范缺乏、编写门槛高
  - 问题2：输入生成大多依赖语法层面规则，缺乏对系统调用语义的深入理解
- **ECG**
  - 创新点1：结合静态分析和LLM，自动提取系统调用参数约束并生成测试用例
  - 创新点2：设计方向引导的输入变异策略，多阶段逐参数生成测试用例



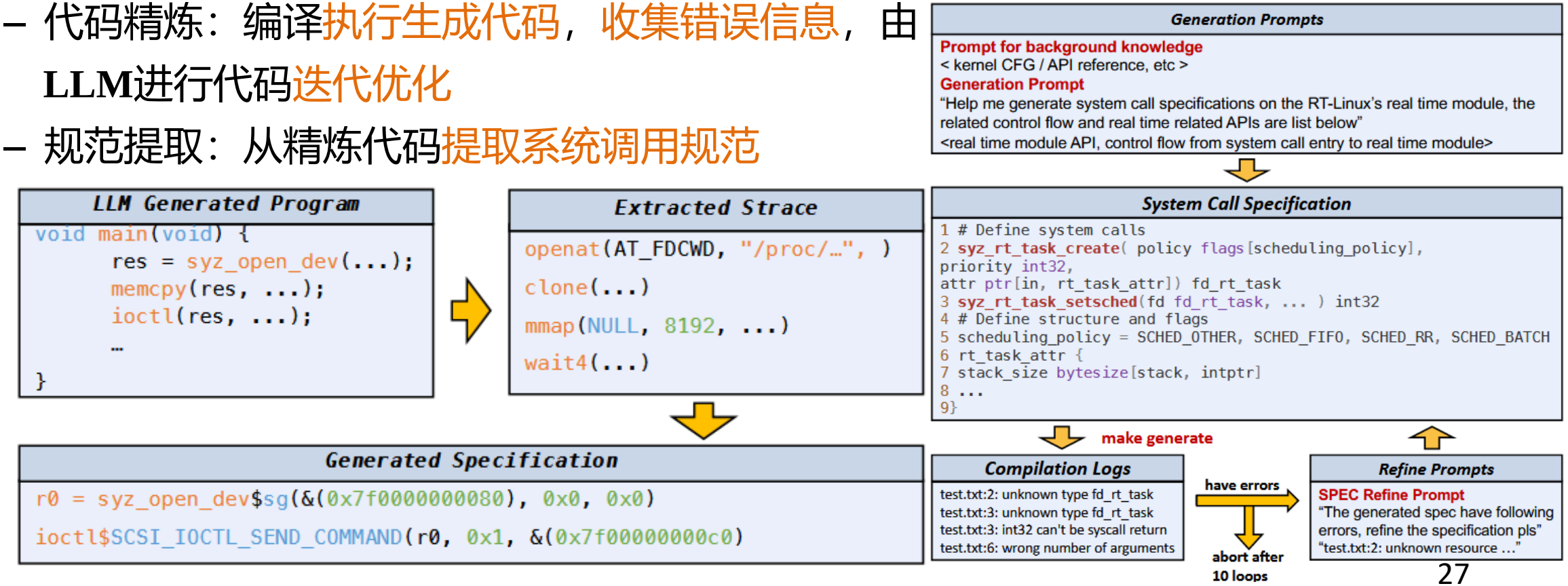
ECG 算法原理图

- 思路分析
  - 规范构建阶段
    - 获取嵌入式系统源代码和相关文档
    - 静态分析源代码获取约束信息
    - **LLM生成可执行代码并精炼**
    - 基于精炼代码，提取系统调用规范
  - 输入生成阶段
    - 多阶段逐参数生成测试用例
    - **LLM指导测试用例生成**
    - 方向检查器优化变异方向



算法原理 规范构建阶段

- 规范构建阶段
  - 输入准备：嵌入式系统文档、源代码
  - 静态分析：对源代码进行静态分析，提取约束信息，指导LLM生成C程序代码
  - 代码精炼：编译执行生成代码，收集错误信息，由LLM进行代码迭代优化
  - 规范提取：从精炼代码提取系统调用规范



算法原理 输入生成阶段

- 多阶段输入生成
  - 参数分步生成：按照参数类型逐步生成参数值
    - 基于目标模块，先过滤相关系统调用
    - 结合选定Syscall的参数类型、约束以及方向向量确定变异策略
    - 具体生成参数值时，先生成基本参数值，再生成复杂参数
  - 方向检查器：收集执行反馈，计算与目标模块距离，生成方向向量
    - 根据目标模块CFG，确定代码基本块和控制流关系
    - 通过图搜索算法BFS，计算当前执行路径与基本块之间的最短距离
    - 对于目标模块中所有基本块，计算最短距离然后加权得到模块距离MD
    - 根据MD，为系统调用中每个参数生成一个方向向量
  - LLM生成：根据方向向量，生成新测试用例或变异现有用例
  - 语料库更新

实验设计 数据资源

- 实验配置
  - 基础模糊器：**Syzkaller**
  - 测试对象：**RT-Linux(6.7，6.8)、RaspiOS(6.7，6.8)、OpenWrt(5.15，6.1)**
  - **LLM：Mixtral-8x7b(规范生成)、Mistral-7b(输入生成)**
  - 硬件配置
    - **128核AMD EPYC CPU，32GB内存**
    - **Tesla V100S-PCIE-32GB GPU*2**
  - 操作系统： **Ubuntu 20.04.4 LTS 64 bit OS**
- 对比方法
  - **Syzkaller (开源工具)**
  - **KernelGPT(2023)**
  - **Moonshine(2018)**
  - **DRLF(2023)**

实验设计 实验结果

- **RQ1**：漏洞检测能力
  - 发现未知漏洞：**32**
  - 逻辑错误：**15**
  - 数据竞争：**12**
  - 内存损坏：**5**

#	Modules	Versions	Locations	Bug Types
1	fs/buffer	RT-Linux 6.7	mark_buffer_dirty	logic error
2	drivers/pci	RT-Linux 6.7	vga_put	logic error
3	kernel/sched	RT-Linux 6.7	select_task_rq_fair	deadlock
4	mm/filemap	RT-Linux 6.7	filemap_fault / page_add_file_rmap	data race
5	drivers/net/	RT-Linux 6.7	e1000_update_stats	memory corruption
6	fs/inode	RT-Linux 6.7	inode_update_timestamps	data race
7	kernel/kprobes	RT-Linux 6.7	arch_adjust_kprobe_addr	logic error
8	fs/dcache	RT-Linux 6.7	d_splice_alias	data race
9	fs/ext4	RT-Linux 6.7	__ext4_new_inode /_find_next_zero_bit	data race
10	drivers/e1000	RT-Linux 6.7	e1000_clean	data race
11	lib/find_bit	RT-Linux 6.7	_find_first_bit	data race
12	kernel/sched	RT-Linux 6.8	__wake_up_common	null-ptr defer
13	lib/kasprintf	RT-Linux 6.8	kvasprintf	logic error
14	kernel/events	RT-Linux 6.8	perf_cgroup_switch	logic error
15	fs/inode	RT-Linux 6.8	generic_update_time / inode_needs_update_time	data race
16	fs/ext4	RT-Linux 6.8	generic_write_end / mpage_submit_folio	data race
17	fs/kernfs	RT-Linux 6.8	kernfs_dop_revalidate	memory corruption
18	fs/ext4	RT-Linux 6.8	ext4_split_extent_at	memory corruption
19	arch/x86/lib	RT-Linux 6.8	memcpy_orig	out-of-bounds
20	kernel/events	RT-Linux 6.8	free_event	logic error
21	drivers/scsi	RT-Linux 6.8	__bitmap_weight /scsi_device_unbusy	data race
22	kernel/rcu	RT-Linux 6.8	__call_rcu_common /mas_walk	data race
23	mm/swap	RT-Linux 6.8	__folio_end_writeback / lru_add_fn	data race
24	arch/x86/lib	RT-Linux 6.8	memmove	memory corruption
25	arch/x86/events/intel	RT-Linux 6.8	intel_pmu_lbr_counters_reorder	logic error
26	arch/x86/kernel	RT-Linux 6.8	deref_stack_reg	logic error
27	arch/x86/kernel	RT-Linux 6.8	__orc_find	memory leak
28	net/9p	RT-Linux 6.8	p9pdu_readf	memory leak
29	arch/x86/lib	OpenWrt 5.15	memset_erms	logic error
30	kernel/smp	OpenWrt 5.15	smp_call_function_single	logic error
31	arch/arm64/kvm	RaspberryPi OS 6.7	kvm_init_stage2_mmu	memory leak
32	arch/arm64/kvm	RaspberryPi OS 6.7	kvm_age_gfn	logic error

实验设计 对比实验

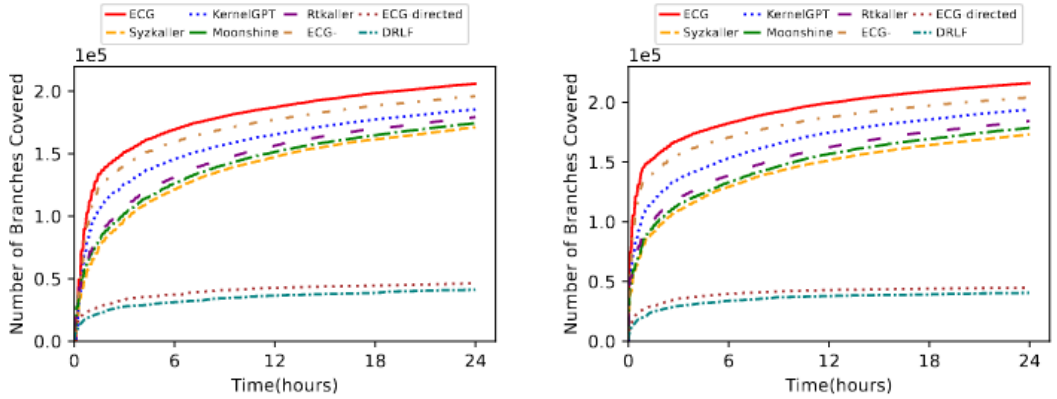
- **RQ1**：漏洞检测能力
  - 相同实验时间和设置下，**10**轮测试，**ECG**在三个系统上平均漏洞数**45.9**
  - **ECG-**：禁用**LLM**引导生成，验证**LLM**对性能的贡献
  - **ECG-directed**：启用定向模糊测试，对比**DRLF**的定向能力

Metrics	OpenWrt		RT-Linux		RaspiOS		Total
	v5.15	v6.1	v6.7	v6.8	v6.7	v6.8	
ECG	7.1	7.8	8.9	8.2	6.5	7.4	45.9
ECG-	6.3	7.2	7.8	7.3	5.8	6.4	40.8
KernelGPT	5.5	6.3	6.7	6.1	5.2	5.7	35.5
Moonshine	4.5	4.9	4.4	5.1	4.2	4.7	27.8
Rtkaller	4.7	5.3	5.2	5.5	4.6	5.1	30.4
Syzkaller	4.1	4.6	4.3	4.8	3.9	3.7	25.4
ECG-directed	4.3	4.6	3.9	4.2	4.1	3.5	24.6
DRLF	3.5	4.1	3.2	3.7	3.6	3.1	21.2

ECG和其他Fuzzer漏洞数量对比

实验设计 对比实验

- **RQ2**：代码覆盖率提升
  - 相同实验时间和设置下，10轮测试，ECG在三个系统上平均覆盖183452分支
  - **ECG-**：取消LLM引导生成，覆盖率下降6.4%
  - **ECG-directed**：启用定向模糊测试，对比DRLF的定向能力



(a) RT-Linux v6.7 (b) RT-Linux v6.8

Subject	OpenWrt		RT-Linux		RaspiOS		Overall
	v5.15	v6.1	v6.7	v6.8	v6.7	v6.8	
ECG	**160210.6**	**173660.5**	**206004.3**	**215922.5**	**176111.4**	**168805.2**	**183,452.4**
ECG-	151153.5(+5.99%/0.014)	166260.5(+4.55%/0.022)	196004.3(+5.10%/0.035)	203922.5(+5.88%/0.025)	165550.9(+6.38%/0.011)	151640.6(+11.32%/0.026)	**172422.1(+6.4%)**
Syzkaller	130876.4(+22.41%/0.001)	144863.5(+19.88%/0.003)	171117.3(+20.39%/0.009)	172977.7(+24.83%/0.003)	139509.9(+26.24%/0.003)	134090.6(+25.89%/0.005)	**148,905.8(+23.20%)**
Moonshine	135201.3(+18.50%/0.007)	148661.9(+16.82%/0.004)	174380.2(+18.14%/0.013)	178459.3(+20.99%/0.007)	145825.3(+20.77%/0.008)	138891.7(+21.54%/0.008)	**153,569.9(+19.46%)**
KernelGPT	143582.5(+11.58%/0.011)	160346.4(+8.30%/0.012)	185393.4(+11.12%/0.021)	193854.7(+11.38%/0.015)	158850.5(+10.87%/0.01)	149928.1(+12.59%/0.013)	**165,325.9(+10.96%)**
Rtkaller	139634.2(+14.74%/0.001)	153689.7(+12.99%/0.003)	179253.7(+14.92%/0.016)	184209.4(+17.22%/0.006)	151840.7(+15.98%/0.005)	144622.3(+16.72%/0.006)	**158873.6(+15.47%)**
ECG-directed	**47228.6**	**49845.2**	**46180.7**	**44865.5**	**47228.6**	**49856.5**	**47534.1**
DRLF	43283.3(+9.12%/0.023)	44646.5(+11.64%/0.017)	41099.7(+12.36%/0.012)	40445.2(+10.93%/0.041)	43585.6(+8.36%/0.011)	43754.4(+13.95%/0.024)	**42802.4(+11.05%)**

ECG和其他Fuzzer代码覆盖率对比

算法总结 ECG

- 算法流程
  - 规范构建：结合静态分析和LLM，从精炼后的C程序提取系统调用规范
  - 多阶段生成：多阶段逐参数生成系统调用，设计基于方向引导的LLM变异策略
- 算法优势
  - ECG可自动化规范生成，减少对人工编写规范的依赖
  - ECG利用LLM语义理解能力，根据系统调用规范和执行反馈，提高测试用例质量
- 算法不足
  - LLM推理过程相对较慢，ECG单位时间生成测试用例数量远小于传统模糊测试器
  - ECG实验中部署两个本地LLM，需要大量计算资源
  - 提取的系统调用规范质量一定程度上依赖文档完整性和静态分析的结果

特点总结与未来展望

特点总结与未来展望

- **KernelGPT**
  - 自动化多阶段迭代推断系统调用规范
  - 可产生多样化、有效的系统调用
  - **LLM**仅负责系统调用规范生成
- **ECG**
  - 自动化推断系统调用规范
  - 基于方向引导，多阶段逐参数生成测试用例
  - **LLM**同时参与系统调用规范构建和测试用例生成
- 未来发展
  - 扩展**KernelGPT**支持的内核子系统，目前仅支持驱动和套接字系统调用
  - 提高**ECG**中测试用例生成速度、减少资源消耗

参考文献

[1] Yang C, Zhao Z, Zhang L. Kernelgpt: Enhanced kernel fuzzing via large language models. Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems[C]. Volume 2. 2025: 560-573.

[2] Zhang Q, Shen Y, Liu J, et al. ECG: Augmenting Embedded Operating System Fuzzing via LLM-Based Corpus Generation[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2024, 43(11): 4238-4249.

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！