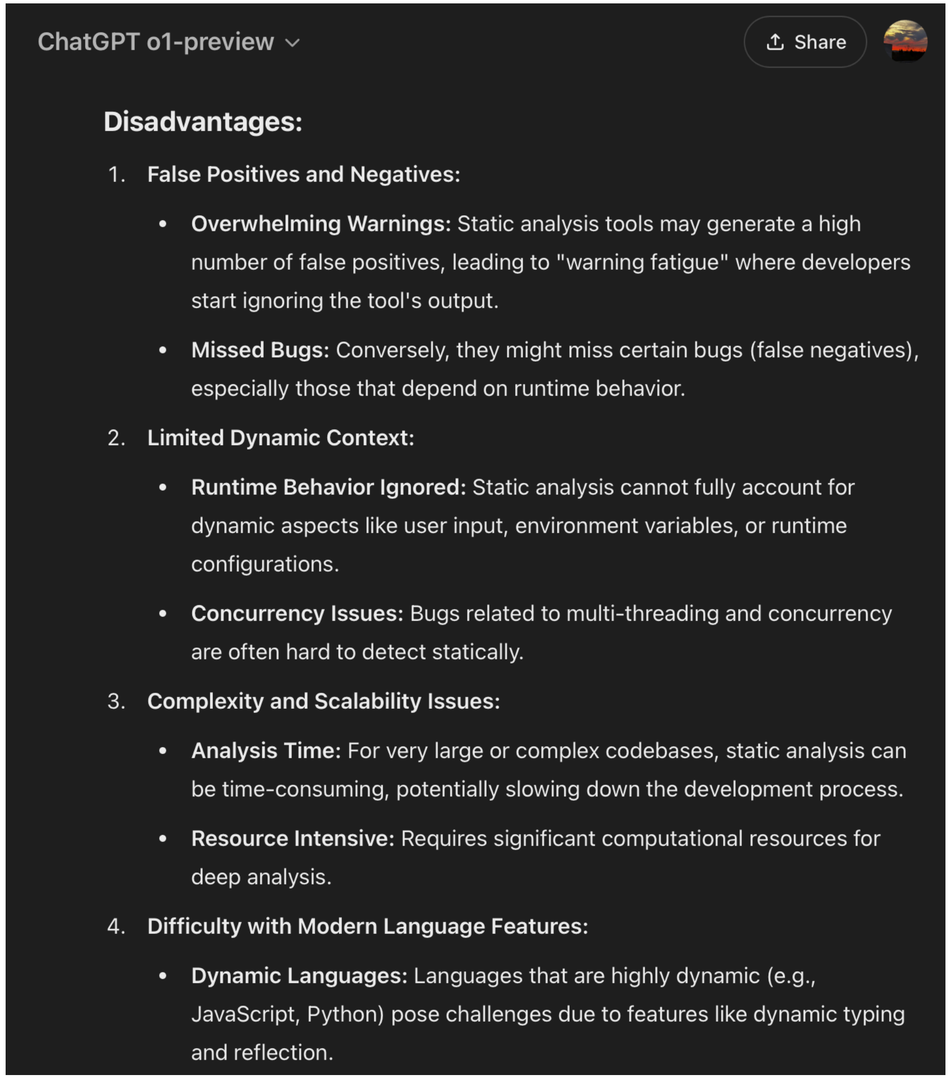
Enhancing Static Analysis for Practical Bug Detection: An LLM-Integrated Approach

Haonan Li, Yu Hao, Yizhuo Zhai, Zhiyun Qian
University of California, Riverside

Motivation: Challenges of Static Analysis

In Bug Detection

- *False Positives and Negatives*
- *Complexity, Scalability*
- *Limited Domain Knowledge*
- **Thinking**: how to solve these challenges? 🤔



Enhance Static Analysis

An LLM-integrated Approach

- Compared to static analysis, LLM has:
  - ✅ Domain knowledge & heuristics
  - ✅ Smart summary for complex code
  - ❌ Unstable response
  - ❌ Limited reasoning on code
- **Problem:** Can we enhance static analysis by integrating LLM?🤔

Case Study: UBITect

- UBITect [1]: finding Use-Before-Initialization (UBI) bugs in the Linux kernel
- UBI: variables may be used before they are initialized.
- Shows the challenges of static analysis:
  - FP and FN
  - ...

```
1   int F(){
2     int v; // declaration of v
3     if (constraint)
4       init(&v); // initializer of v
5     use(v); // use of v
6   }
```

A typical UBI bug, v may not be initialized before reaching Line 5

[1] Yizhuo Zhai, Yu Hao, et.al. “UBITect: A Precise and Scalable Method to Detect Use-before-Initialization Bugs in Linux Kernel.” In FSE’20. https://doi.org/10.1145/3368089.3409686.

Case Study: UBITect

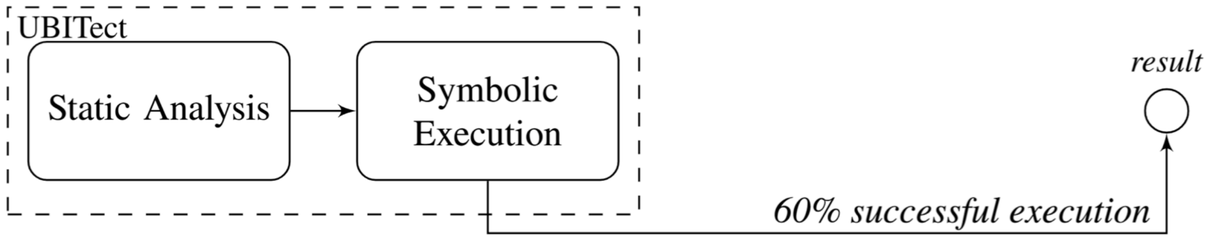
False Positives and Negatives

Two Stages:

1. Static analysis (scalable, imprecise)
2. Symbolic execution (precise, slow)

Dilemma:

- **FP:** *>95% FP* in *static analysis*; utilizing symbolic execution to filter out FP
- **FN:** discarding 40% undecided cases, *may ignore 40% potential bugs* (due to time/memory limitation)



The overview of UBITect.

Motivating Example: `sscanf`

```
1   static int libcfs_ip_str2addr(...){
2     unsigned int a, b, c, d;
3     if (sscanf(str, "%u.%u.%u.%u%n",
4           &a, &b, &c, &d, &n) >= 4 && ...){
5             // use of a, b, c, d
6     }
7   }
```

- ***Static analysis:*** `sscanf` *may not initialize* `a, b, c, d` (path-insensitive)
  - Report a UBI bug at Line 5
- ***Symbolic execution:*** timeout

```
14  int vsscanf(const char *buf, const char *fmt,
    ↪  va_list args){
15    const char *str = buf;
16    ...
17    while (*fmt) {
18      switch (*fmt++) {
19        case 'c': {
20          char *s = (char *)va_arg(args, char*);
21          ...
22          do { *s++ = *str++; }
23          while (...*str); num++;
24        }
25      }
26      ...
27      num++;
28          ...
29    }
30    return num;
31  }
```

Understand `sscanf` with LLM

```
1   static int libcfs_ip_str2addr(...){
2     unsigned int a, b, c, d;
3     if (sscanf(str, "%u.%u.%u.%u%n",
4           &a, &b, &c, &d, &n) >= 4 && ...){
5             // use of a, b, c, d
6     }
7   }
```

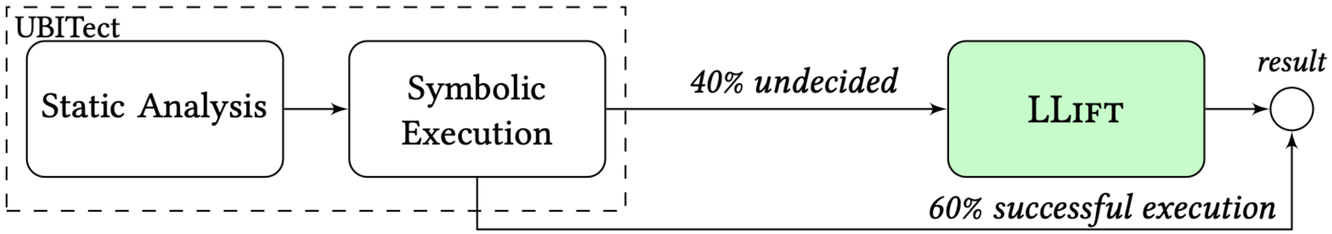
Ask ChatGPT: “are variables `a, b, c, d` to be initialized before reaching Line 5?”

- *...`sscanf` here to parse IP address ... by the time the code reaches the line ... `a, b, c, d` have been initialized*

Scan me to see the chat!

LLift: LLM for Undecided Cases

- LLM-based framework: LLift.
- Asking LLM, "*are these variables initialized before their use*"
- "must_init" -> not a bug



The overview of UBITect w/ LLift

Directly Ask GPT

- Challenge: “simple prompt” doesn’t work
  - Bug detection requires multiple steps w/ rigorous rules
  - LLM is unaware of them
- ***SOLUTION:*** Teach LLM the steps and rules of bug detection

Combination	TN(C)	TP(C)	Precision	Recall	Accuracy	F1 Score
Simple Prompt	12(9)	2(1)	0.12	0.15	0.35	0.13
PGA	13(9)	5(1)	0.26	0.38	0.45	0.31
PGA+PP	5(3)	6(1)	0.21	0.46	0.28	0.29
PGA+PP+SV	5(2)	11(8)	0.33	0.85	0.40	0.48
PGA+PP+TD	22(14)	6(4)	0.55	0.46	0.70	0.50
PGA+PP+TD+SV	25(17)	13(12)	0.87	1.00	0.95	0.93
Oracle	27(27)	13(13)	-	-	-	-

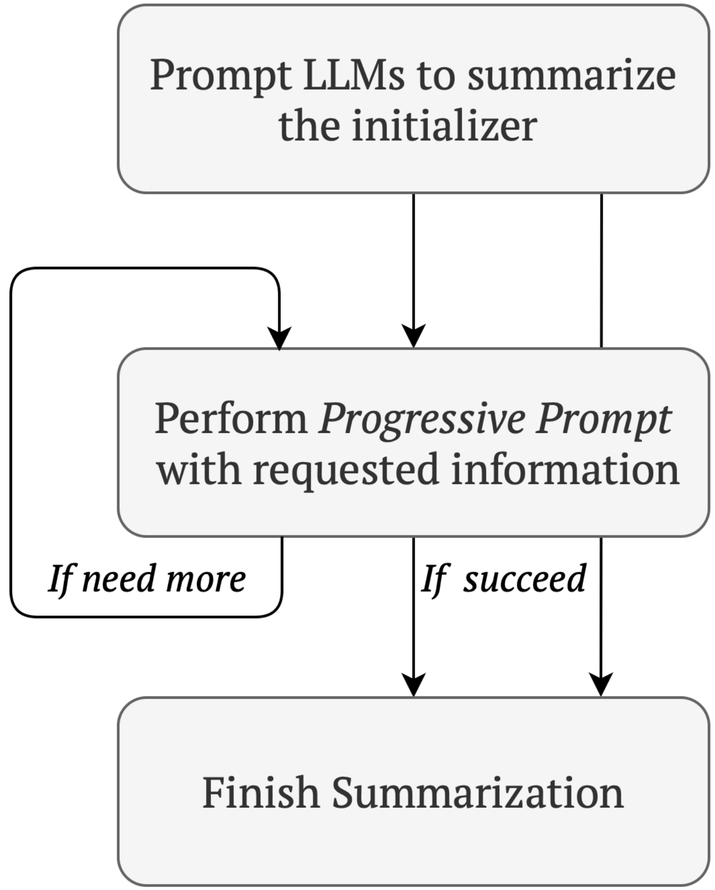
Design Principles

- Task Decomposition (TD): break the task into small pieces
- Post-constraint Guided Path Analysis (PGA)

```
1   int I(){
2     if (...){ //path_1
3       return 0;
4     }
5     else{ //path_2
6       return -1;
7     }
8   }
9   void F(){
10      int v;
11      int ret = I(v);
12      if (ret == 0){
13          use(v);
14      }
15  }
```

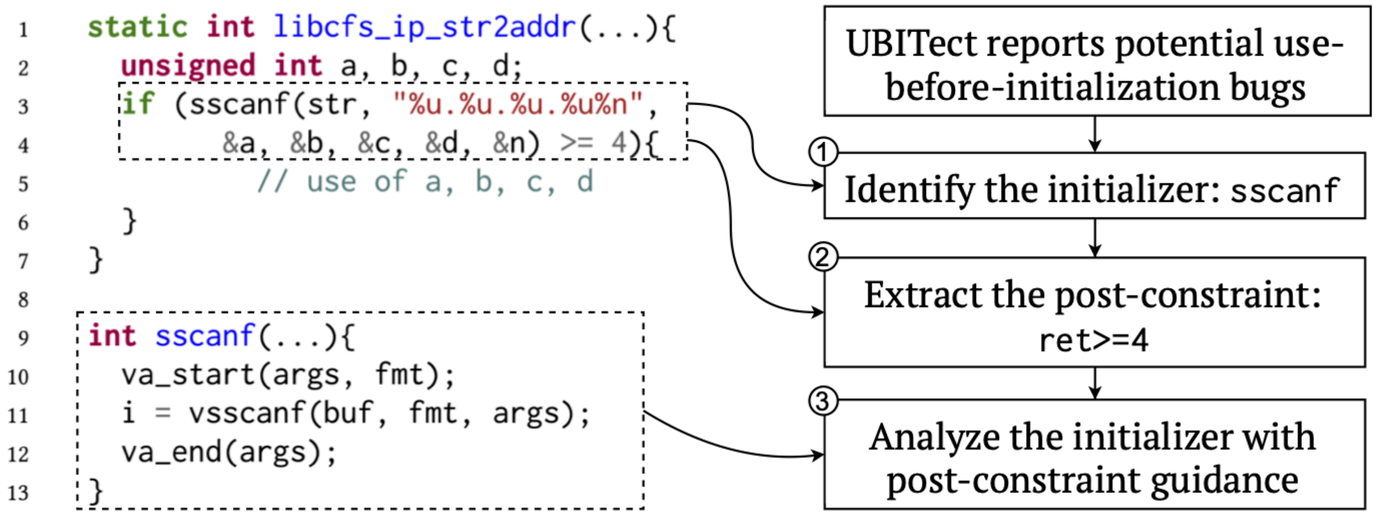
Design Principles

- Task Decomposition (TD): break the task into small pieces
- Post-constraint Guided Path Analysis (PGA)
- Progressive Prompt (PP): Let LLMs decide the what they want (for function definitions)



Workflow

- Identify the initializer
- Extract post-constraints
- Analyze the initializer with post-constraint guidance



Evaluation

- Precision and Recall
- Other Models than GPT-4
- Other projects than Linux
- Contributions of each design component

- Dataset: **Rnd-300**
  - randomly choose 300 from 53,000 *undecided* cases

Precision & Recall

- **Precision:** 5 TP, 50% precision
- **Recall:** no missed bugs found in Rnd-300
- Extend to 1000, 13 TP and 13 FP
- 4 of them are confirmed as real bugs

Initializer	Caller	File Path	Variable	Line
read_reg	get_signal_parameters	drivers/media/dvb-frontends/stv0910.c	tmp	504
regmap_read	isc_update_profile	drivers/media/platform/atmel/atmel-isc.c	sr	664
ep0_read_setup	ep0_handle_setup	drivers/usb/mtu3/mtu3_gadget_ep0.c	setup.bRequestType	637
regmap_read	mdio_sc_cfg_reg_write	drivers/net/ethernet/hisilicon/hns_mdio.c	reg_value	169
bcm3510_do_hab_cmd	bcm3510_check_firmware_version	drivers/media/dvb-frontends/bcm3510.c	ver.demod_version	666
readCapabilityRid	airo_get_range	drivers/net/wireless/cisco/airo.c	cap_rid.softCap	6936
e1e_rphy	__e1000_resume	drivers/net/ethernet/intel/e1000e/netdev.c	phy_data	6580
pci_read_config_dword	adm8211_probe	drivers/net/wireless/admtek/adm8211.c	reg	1814
lan78xx_read_reg	lan78xx_write_raw_otp	drivers/net/usb/lan78xx.c	buf	873
t1_tpi_read	my3126_phy_reset	drivers/net/ethernet/chelsio/cxgb/my3126.c	val	193
pci_read_config_dword	quirk_intel_purley_xeon_ras_cap	arch/x86/kernel/quirks.c	capid0	562
ata_timing_compute	opti82c46x_set_piomode	drivers/ata/pata_legacy.c	&tp	564
pt_completion	pt_req_sense	drivers/block/paride/pt.c	buf	368

True positives of LLift on Rnd-300. Above is 5TP on Rnd-300, below is 8TP scaled up to 1000

Other Models

- *Disclaimer*: not reported in the paper
- LLift works on (most) other models
- Llama shows usable results

Model	Simple Prompt			All Components		
	TN	TP	F1 Score	TN	TP	F1 Score
CodeLlama (34b)	23	11	0.23	42	12	0.31
Llama 2 (70b)	0	13	0.21	49	11	0.31
Palm 2	12	3	0.06	40	6	0.16
Claude 2	64	4	0.17	80	2	0.13
GPT-3.5 (0613)	3	13	0.22	37	9	0.23
GPT-4 (0613)	51	2	0.07	91	13	0.87
Oracle	95	13	-	95	13	-

Summary

- We present LLift, the first work to show how to use LLM to improve static analysis in practice.
- LLift introduces new design strategies (PGA, PP), which can also help future studies.
- We test LLift on real-world codebase, achieving good precision and recall, and finding 13 new TP.

Learn more details:
sites.google.com/view/llift-open