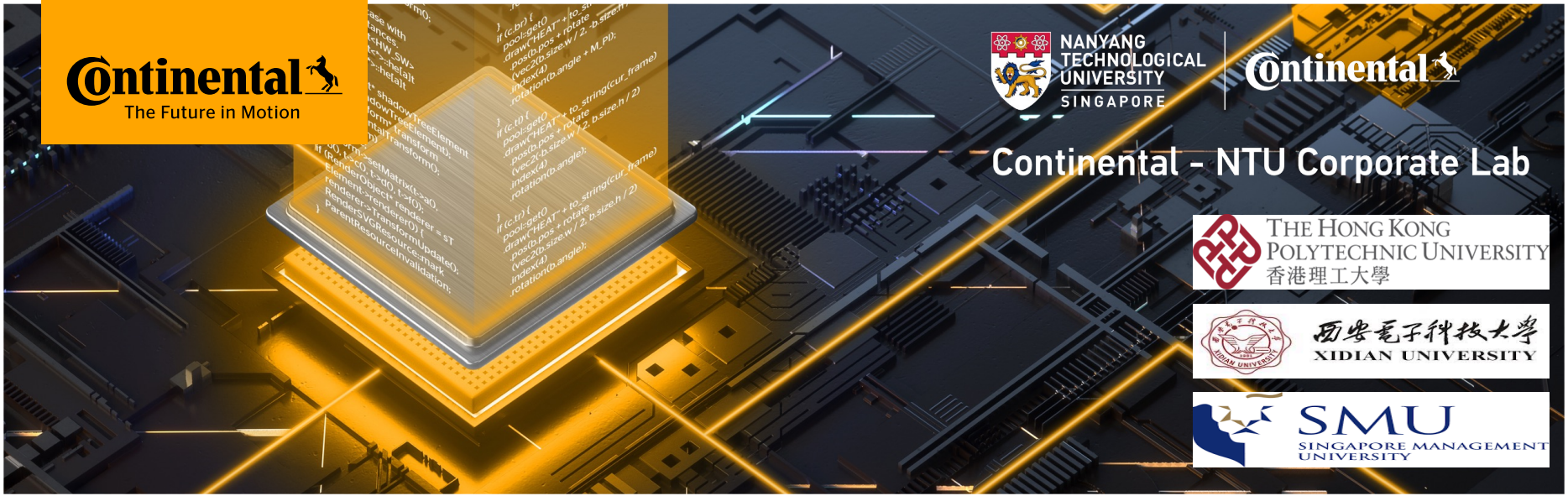
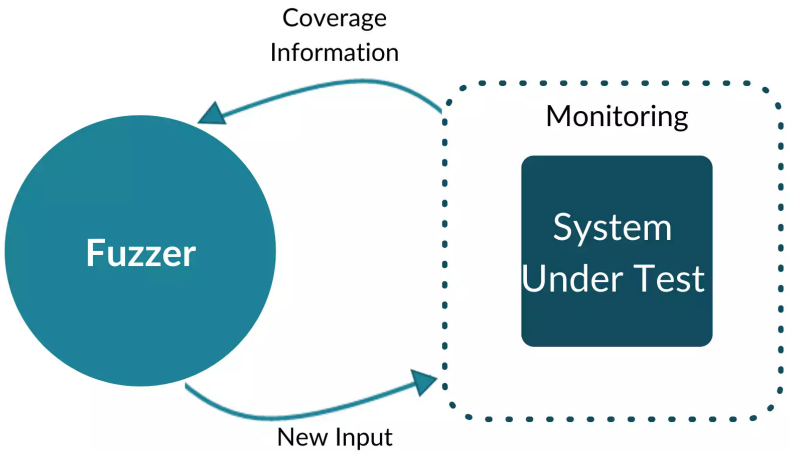


Automata-Guided Control-Flow-Sensitive Fuzz Driver Generation

Cen Zhang, Yuekang Li, Hao Zhou, Xiaohan Zhang, Yaowen Zheng, Xian Zhan,
Xiaofei Xie, Xiapu Luo, Xinghua Li, Yang Liu, Sheikh Mahbub Habib
Continental-NTU Corporate Lab
USENIX 2023

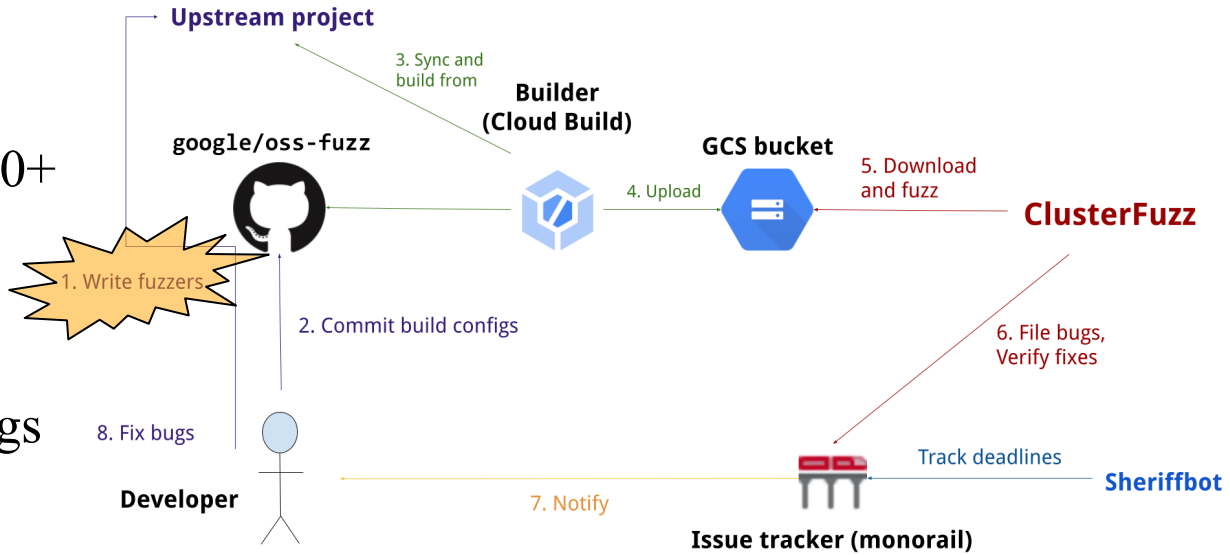
Fuzzing



- SUT should be an executable
- Fuzz drivers are the SUT for library APIs
- The quality of fuzz driver matters for fuzzing's effectiveness

Fuzz Driver Generation in Industrial

- 7 years, 700+ contributors, 1,000+ drivers
- 8,900/28,000 vulnerabilities/bugs (Feb 2023)



Automatic Generation Can Help !

A Fuzz Driver Example for Library APIs

```
PDFDoc ParsePDF(File pdfFile);
```

API Function, Not Executable 😕

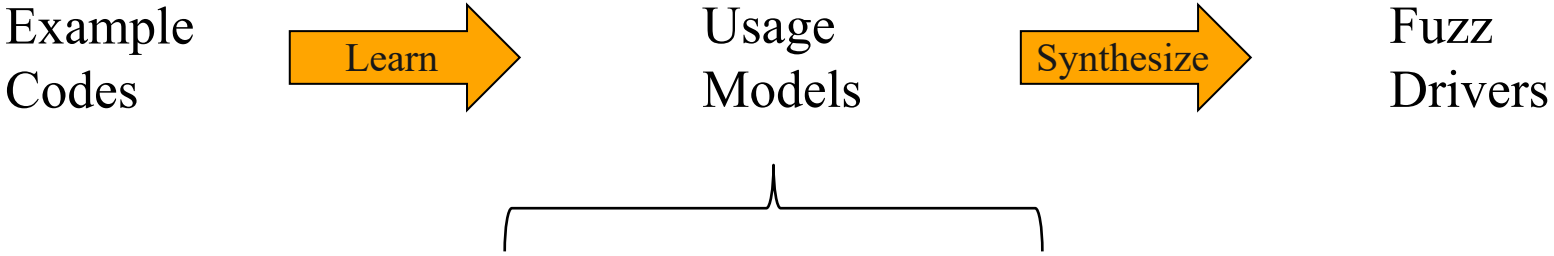
```
void main(…) {
  pdf = ParsePDF("fuzzed_input.pdf");
  while(pdf.hasNextPage()) {
    page = pdf.getNextPage();
    page.render();
  }
}
```

Fuzz Driver, Executable 😊

A Paradigm for Existing Solutions

- ”Learn and Synthesize” workflow

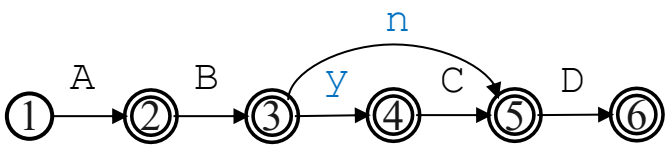
Key Observation: The Missing of “Ctrl-Flow Dep”



Where are the “`if{…}else{…}`”, “`while{…}`” ?

Automata for Representing “Ctrl-Flow Dep”

```
void main(…) {
  pdf = new PDF(“input.pdf”);
  if (pdf.parse())
    pdf.extractText();
  pdf.close();
}
```



“AB(n|yC)D”

API Function Events:

```
A -> ret_A = new PDF(..)
B -> ret_B = pdf.parse()
C -> pdf.extractText()
D -> pdf.close()
```

API Condition Events:

```
y -> ret_B == true
n -> ret_B == false
```

Works on sequences & branches.

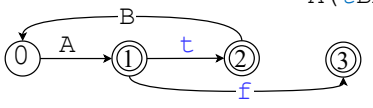
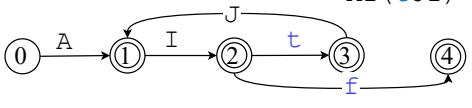
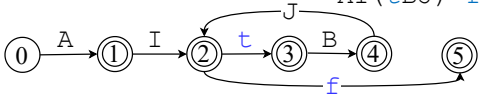
Automata for Representing “Ctrl-Flow Dep” – Cont’d

	Loop Case I	Loop Case II	Loop Case III

Code	`while (pdf.hasNextPage())`
	`  pdf.getNext();`
		`it = pdf.pageIterator();`
		`while (it.hasNext())`
		`  it.getNext();`
			`n = pdf.getPageNum();`
			`for (i = 0; i < n; i = i + 1)`
			`  pdf.getPage(i);`

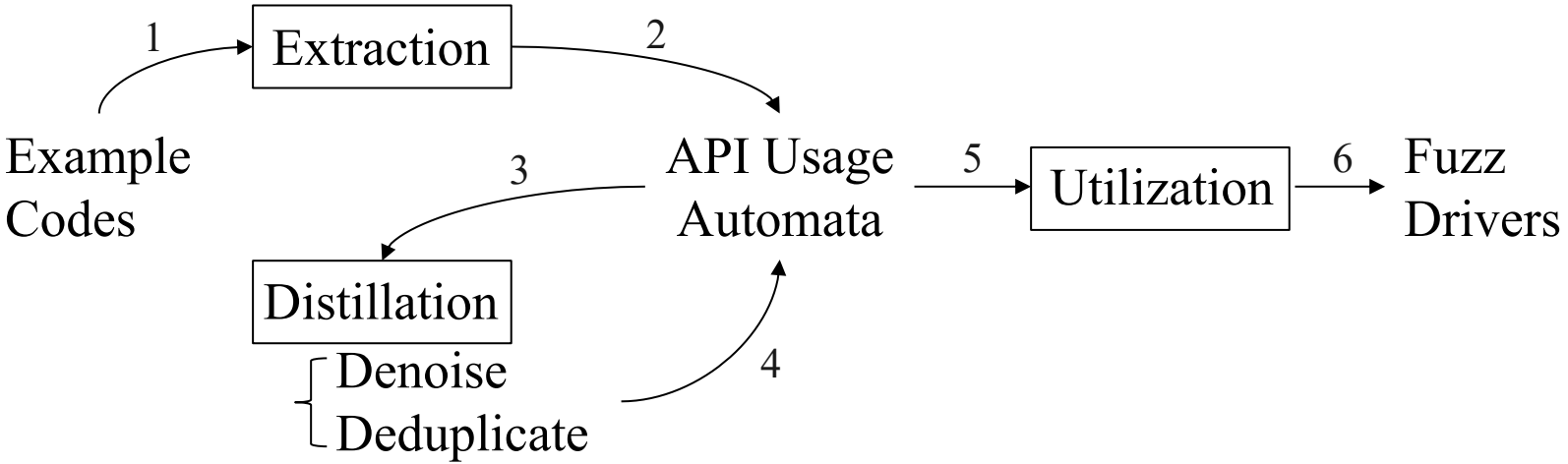
Mock APIs		`bool mock_I(Iterator it)`
		`{ return it.hasNext(); }`
		`PdfPage mock_J(Iterator it)`
		`{ return it.getNext(); }`
			`int mock_I()`
			`{ return 0; }`
			`int mock_J(int i)`
			`{ return i + 1; }`

Alphabet	`A:ret_A = pdf.hasNextPage();`
	`B:ret_B = pdf.getNext();`
	`t:ret_A == true;`
	`f:ret_A == false;`
		`A:ret_A = pdf.pageIterator();`
		`I:ret_I = mock_I(ret_A);`
		`J:ret_J = mock_J(ret_A);`
		`t:ret_I == true;`
		`f:ret_I == false;`
			`A:ret_A = pdf.getPageNum();`
			`B:ret_B = pdf.getPage(ret_I);`
			`I:ret_I = mock_I();`
			`J:ret_I = mock_J(ret_I);`
			`t:ret_I < ret_A;`
			`f:ret_I >= ret_A;`

Automaton	A(tBA)*f 	AI(tJI)*f 	AI(tBJ)*f 

Works on loops.

Automata-Based Modeling X Fuzz Driver Generation



Evaluation – Applying to Real-World Libraries

Popular Java libraries

- From Apache, Github, …
- Popular in both PC & Android

Open-source consumer programs

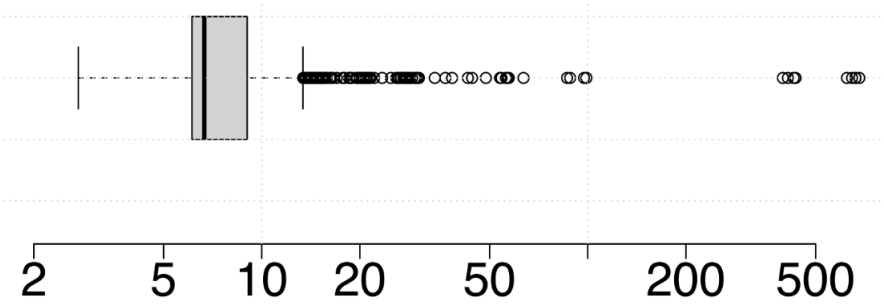
- Searched from Sourcegraph (Github, Gitlab, …)

Attack Surface	Crawling		Learning Inputs Preparation					Usage Automaton Learning									Fuzz Driver Synthesis		
	# of Projs	# of Jars	# of Entries	# of Raw NFAs	# of APIs	# of Data Deps	CPU Sec (Pct.)	# of C_C	# of C_{MG}	# of E_{API}	# of E_{Mock}	# of E_{Cond}	# of E_{Total}	# of State	# of Tran	CPU Sec (Pct.)	# of Ctrl Flow	# of Data Flow	CPU Sec
apachetar	36/911	39/92	34,905	91	33	2,089	1,647 (**66%**)	0	0	22/33	0/1	16/54	38/88	165	223	832 (34%)	319	319	< 1
apachepoi	40/984	74/1,197	26,534	247	243	6,619	656 (34%)	1	3	69/243	8/15	12/109	89/367	89	94	1,289 (**66%**)	20	20	< 1
itextpdf	25/89	33/44	14,632	2,236	311	8,560	194 (3%)	1	3	75/311	59/88	78/365	212/764	308	348	7,223 (**97%**)	16	131,088	< 1
junrar	16/72	24/441	9,737	143	147	1,023	114 (16%)	0	0	49/147	0/0	14/52	63/199	120	150	617 (**84%**)	12	13	< 1
pdfbox	34/326	138/4,835	83,481	455	339	13,680	1,260 (29%)	0	2	54/339	9/14	36/184	99/537	127	148	3,127 (**71%**)	21	21	< 1
zip4j	62/514	28/262	9,175	41	49	1,635	298 (**69%**)	0	1	34/49	1/2	14/22	49/73	65	75	132 (31%)	5	5	< 1

Rubick can practically learn API usage from public code repositories

Evaluation – Performance Assessment

Lower Whisker	1st Quartile	Median	3rd Quartile	Upper Whisker
2.74	6.11	6.66	9.02	13.38



Most automata are distilled fast

- < 14 seconds

Best lookahead values

- 2 to 3

Metric	L = 1	L = 2	L = 3	L = 4
Time (CPU Sec)	1.21E+04	1.74E+04	7.89E+04	1.65E+06
# of Queries	1.22E+07	1.38E+08	2.86E+09	1.14E+11

Evaluation – Comparison with SOTA

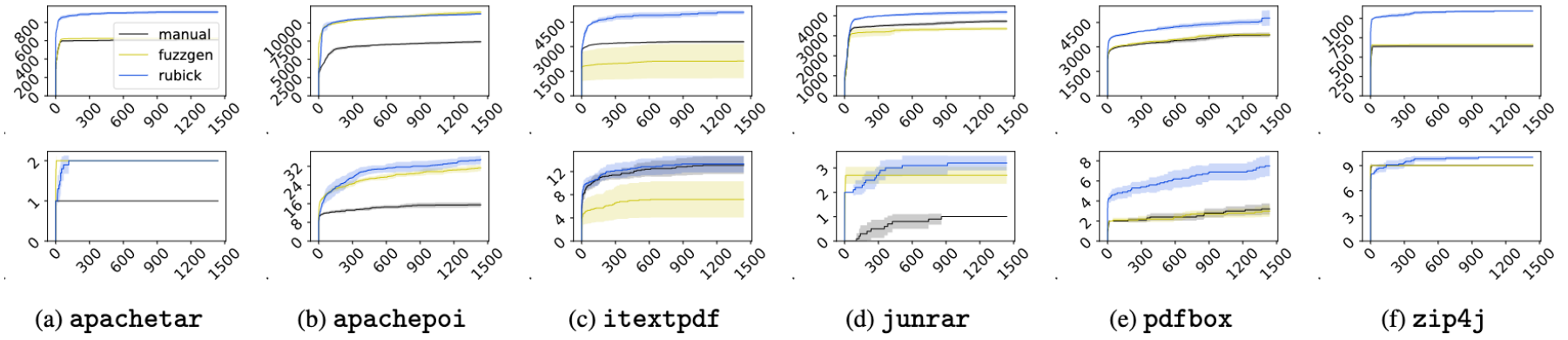


Figure 5: Coverage/Unique Bug (1st/2nd row) Per Time Comparisons for RQ2. X-, Y-axis are time (sec) and edge coverage/# of unique bugs.

Rubick has higher code coverage & found more bugs than SOTA

Evaluation – Ablation Study

The solution for each challenge contributes to the final performance!

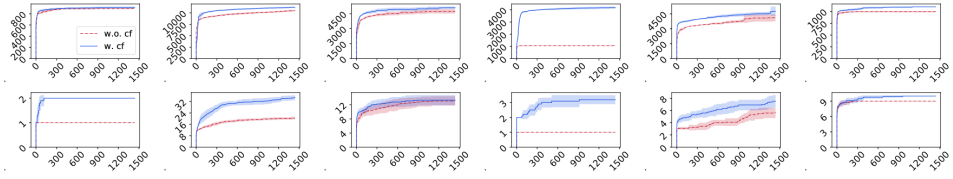


Figure 6: Coverage/Unique Bug (1st/2nd row) Per Time Comparisons for RQ3 C1. X-, Y-axis are time (sec) and edge coverage/# of unique bugs.

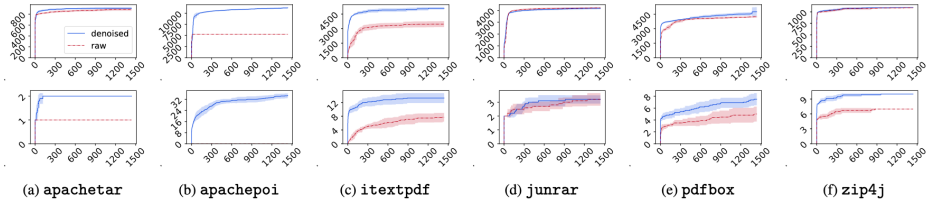


Figure 7: Coverage/Unique Bug (1st/2nd row) Per Time Comparisons for RQ3 C2. X-, Y-axis are time (sec) and edge coverage/# of unique bugs.

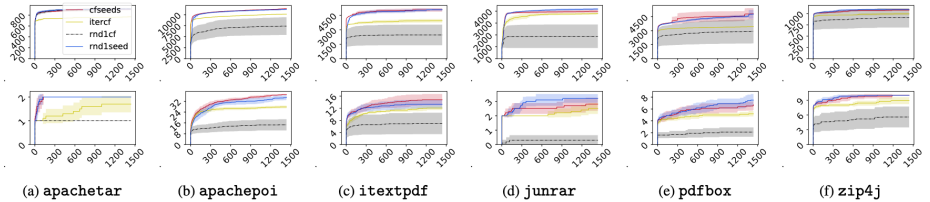


Figure 8: Coverage/Unique Bug (1st/2nd row) Per Time Comparisons for RQ3 C3. X-, Y-axis are time (sec) and edge coverage/# of unique bugs.

Evaluation – False Positive

Rubick also faces FP issue, but it always has lowest FP rate.

Table 6: Unique False Positives Statistics. **Bad Data Dep**, **Invalid Call Seq**, and **Improper Cond** stand for the false crashes caused by missing or invalid API data dependencies, invalid API call sequences, and missing API control dependencies, respectively. The number of false dead loops are counted into a standalone type **Dead Loop**.

	fuzzgen	rubick	w.o. cf	raw
Bad Data Dep	7	0	0	4
Invalid Call Seq	2	0	0	0
Improper Cond	22	7	10	2
Dead Loop	0	0	0	2
Total	31	7	10	8

Evaluation – Real World Fuzzing

- 3 months fuzzing
- 199 unique bugs with 4 CVEs
- Affected popular softwares in both PC & Android platforms
- Exploited 11 popular Android Apps (10,000,000+ download count)

Thank you!

Cen Zhang
cen001@e.ntu.edu.sg

Sheikh Mahbub Habib
sheikh.mahbub.habib@continental-corporation.com