An Infrastructure Approach to Improving Effectiveness of Android UI Testing Tools

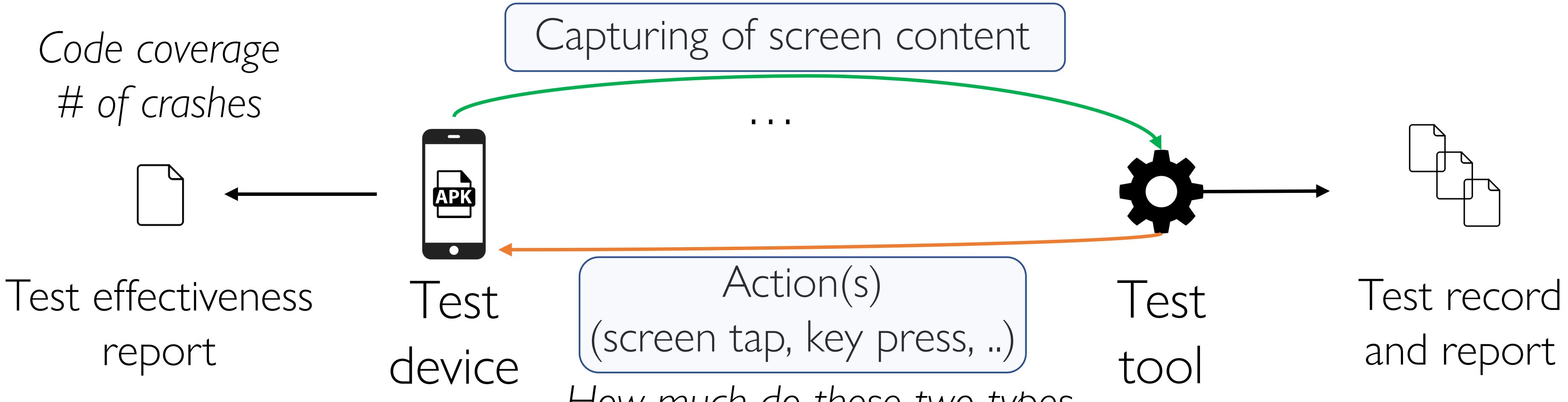
Wenyu Wang, Wing Lam, Tao Xie

{**wenyu2**, winglam2}@illinois.edu
taoxie@pku.edu.cn

CNS-1564274
CCF-1763788
CCF-1816615

Automated UI Testing For Android Apps

Automatically explore the app through UIs, just like human users



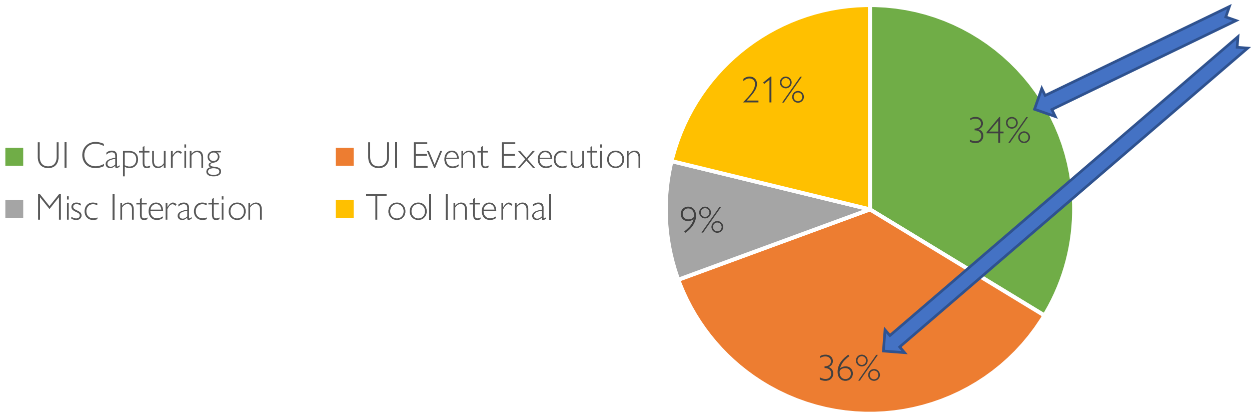
Code coverage
# of crashes
Test effectiveness report
Capturing of screen content
. . .
APK
Test device
Action(s) (screen tap, key press, ..)
Test tool
Test record and report

How much do these two types of operations affect testing effectiveness?

👍 Little human effort 👍 Scalable with numerous devices 👍 Deeper functionality saturation

Infrastructure Efficiency: A Motivational Study

- Break down of testing time usages
  - 3 tools using UIAutomator from the 2018 study [1]
    - Including one re-implemented Monkey (baseline tool), *Chimp*
  - 15 industrial apps from the study, each run for 1-hour
    - 1m+ to 1b+ downloads, 3.3MB to 93MB APK sizes



70% of testing time is spent on just two types of operations!

Save time from these operations →

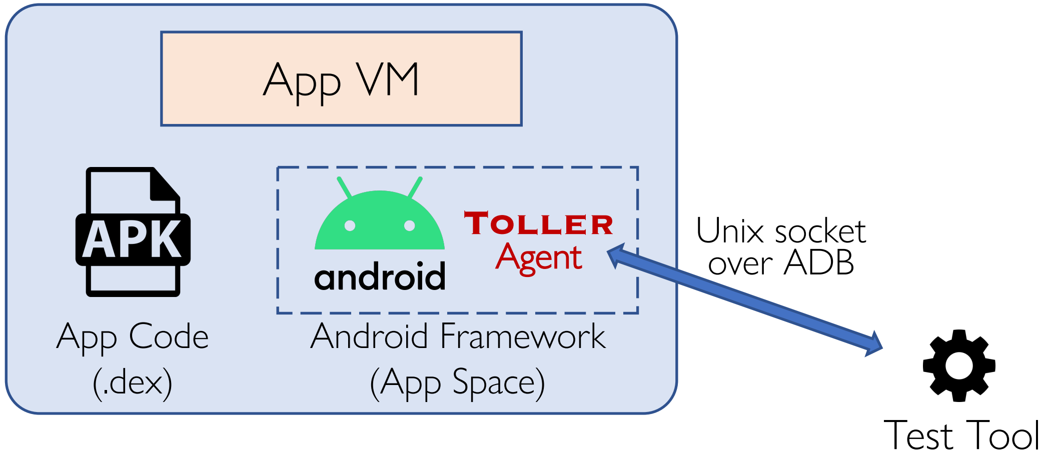
More actions within limited testing time →

Better testing effectiveness!

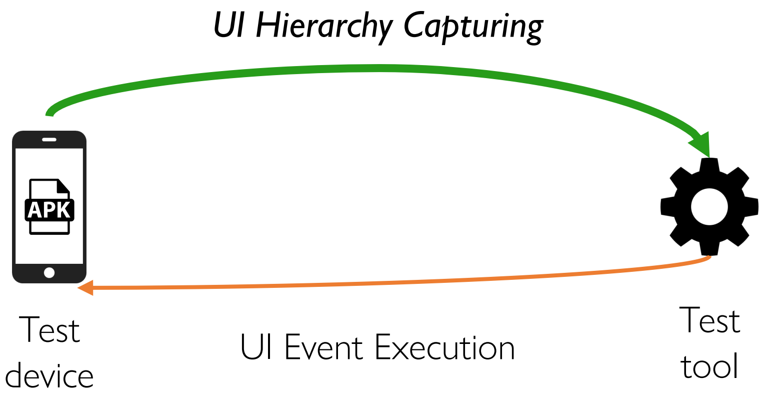
[1] Wenyu Wang, Dengfeng Li, Wei Yang, Yurui Cao, Zhenwen Zhang, Yuetang Deng, and Tao Xie. *An empirical study of android test generation tools in industrial cases* (ASE 2018)

Our Approach (Toller)

- **Goal**: Fast UI Hierarchy Capturing + UI Event Execution
- **Direct access** to app UI data structures & event handlers
- Low-overhead communication with in-app agent



UI Hierarchy Capturing


UI Hierarchy Capturing

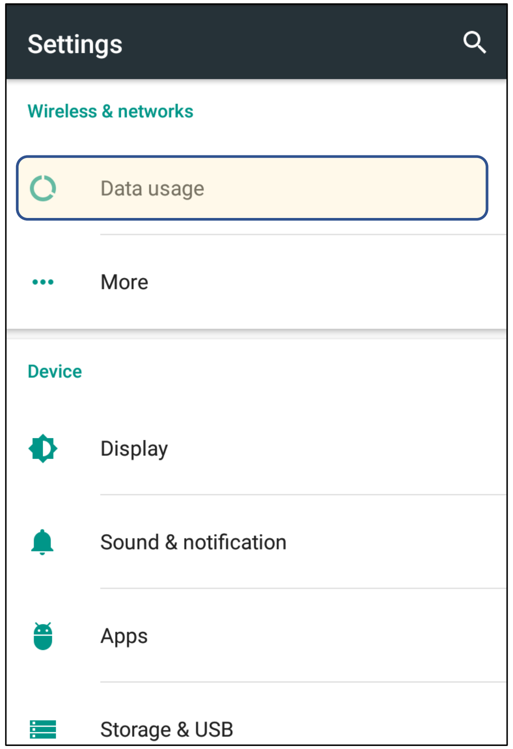
APK

Test device

UI Event Execution

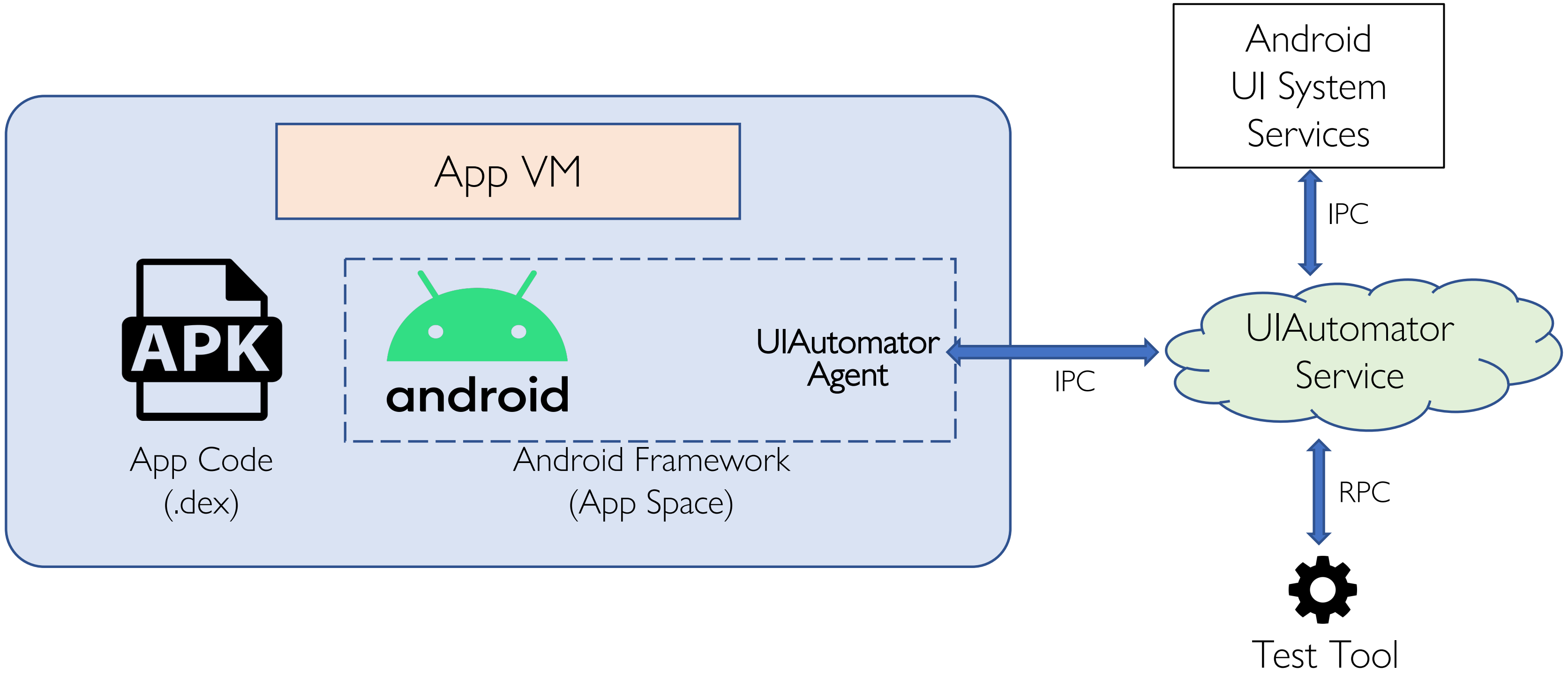
Test tool

Obtain structured on-screen contents from the test device

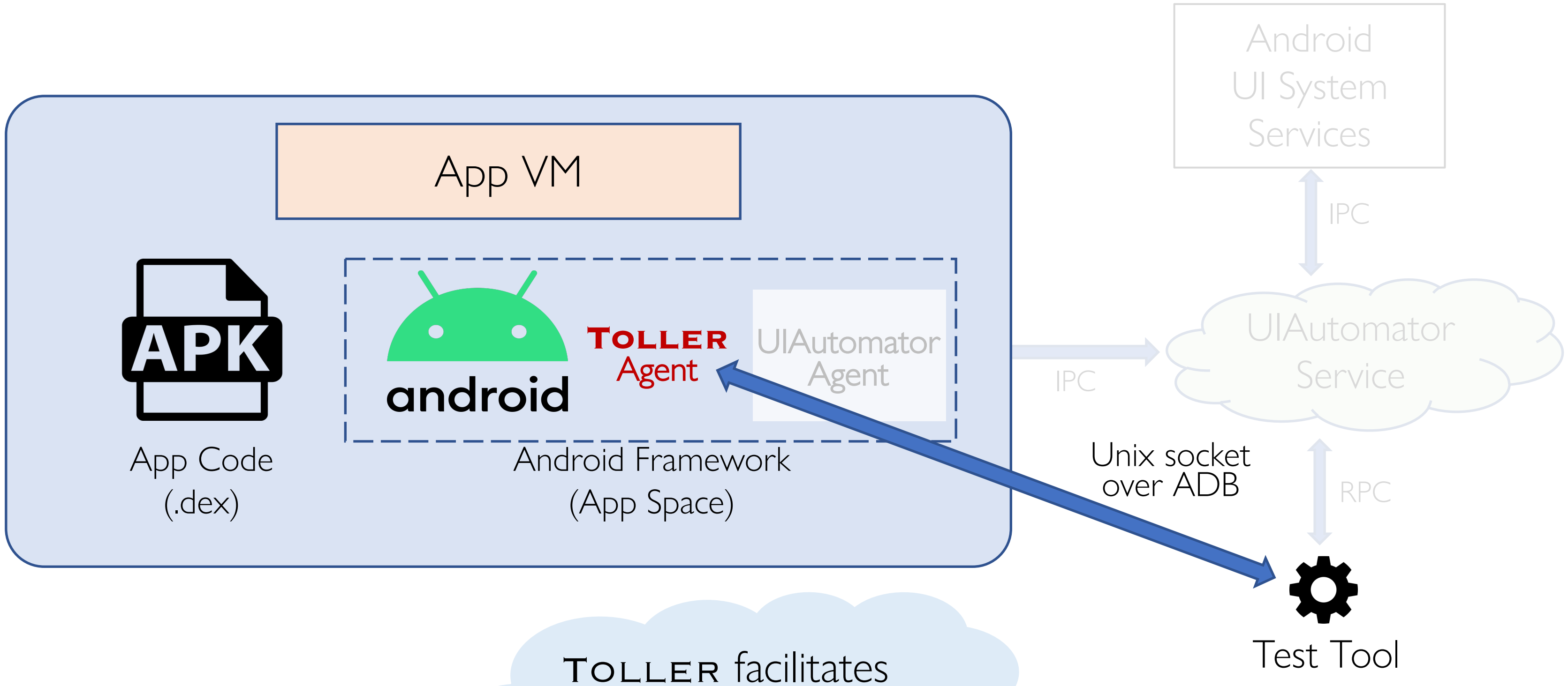


```
..
<node
 text="Settings”
 class="android.widget.TextView” />
..
..
<node
 class="android.widget.LinearLayout”>
  <node
   resource-id="icon”
   class="android.widget.ImageView” />
  <node
   text="Data usage”
   resource-id="title”
   class="android.widget.TextView” />
  ..
</node>
<node
 class="android.widget.LinearLayout”>
  ..
  <node
   text="Display”
   resource-id="title”
   class="android.widget.TextView” />
  ..
</node>
```

Mechanism Of UI Hierarchy Capturing (UIAutomator)

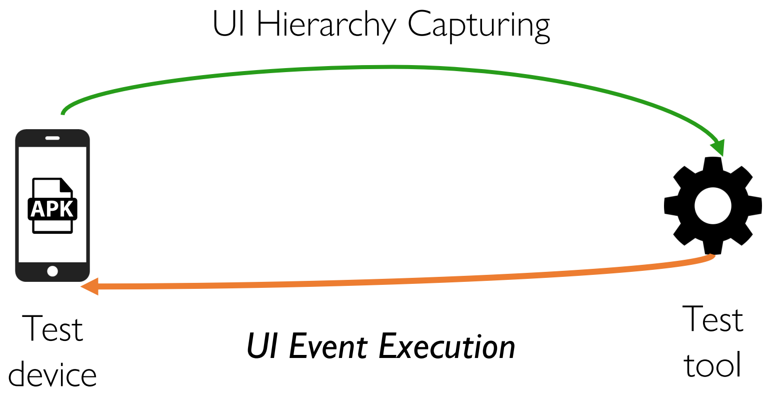


Mechanism Of UI Hierarchy Capturing (Toller)



Toller facilitates direct access to UI data structures!

UI Event Execution



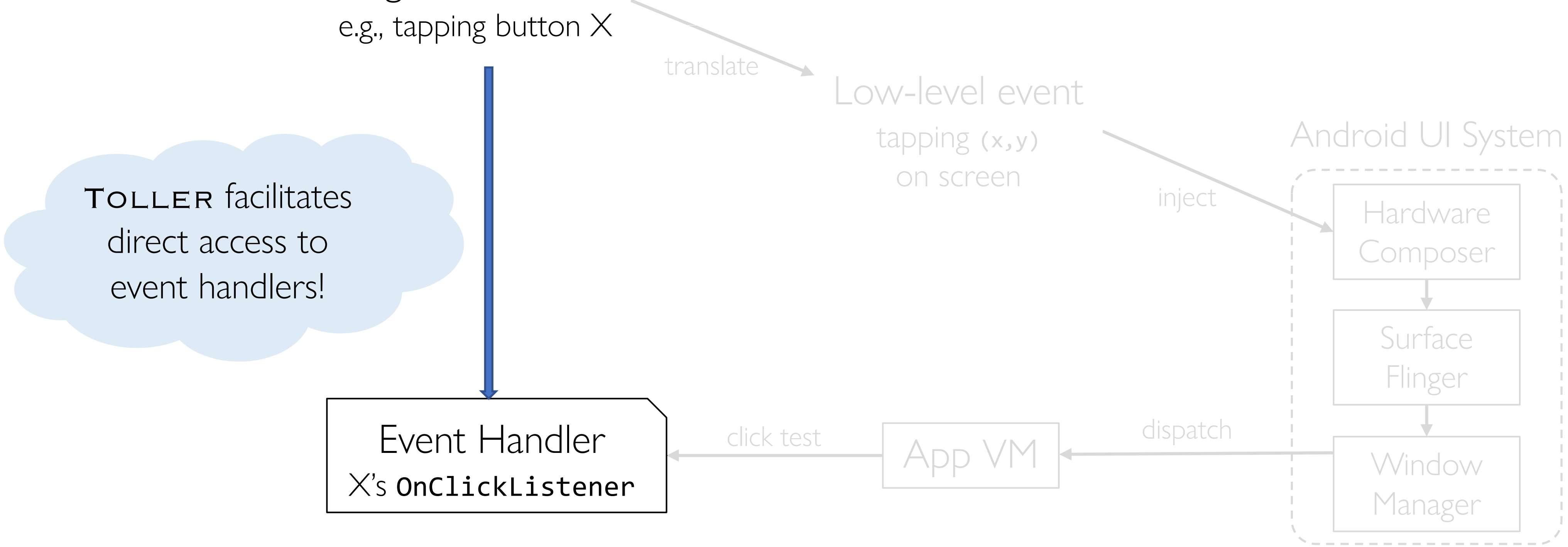
UI Hierarchy Capturing

Test device

UI Event Execution

Test tool

High-level event
e.g., tapping button X



TOLLER facilitates direct access to event handlers!

translate

Low-level event
tapping (x,y)
on screen

inject

Android UI System

Hardware Composer

Surface Flinger

Window Manager

dispatch

App VM

click test

Event Handler
X's `OnClickListener`

Evaluation Outline

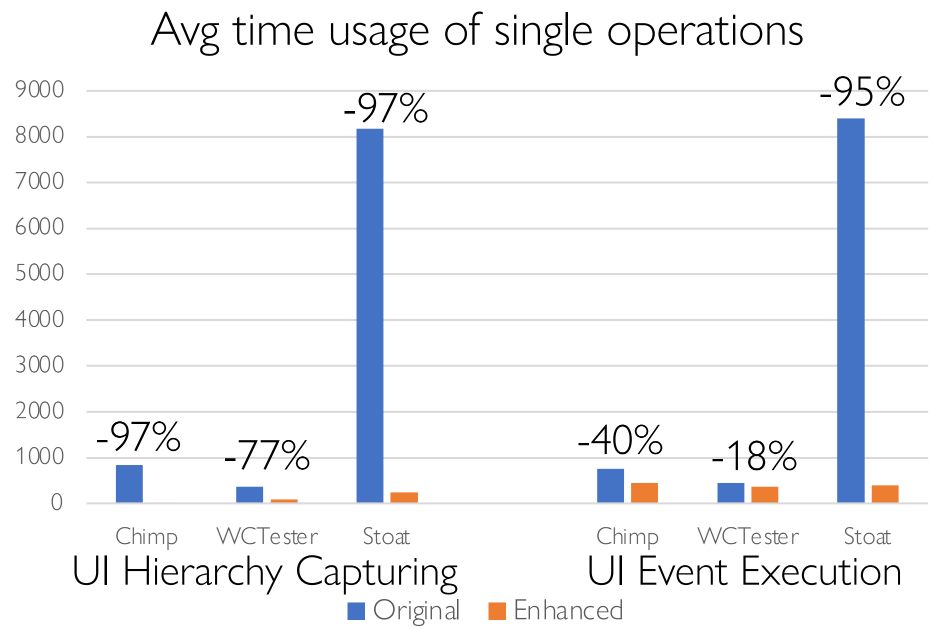
- RQ1: Efficiency of two types of operations
- RQ2: Code coverage improvement
- RQ3: Crash triggering ability improvement
- RQ4: Code/crash overlap with and without TOLLER
  - Please see paper[1] for details
- RQ5: Breakdown of improvements by enhancing types of two operations
  - Please see paper[1] for details

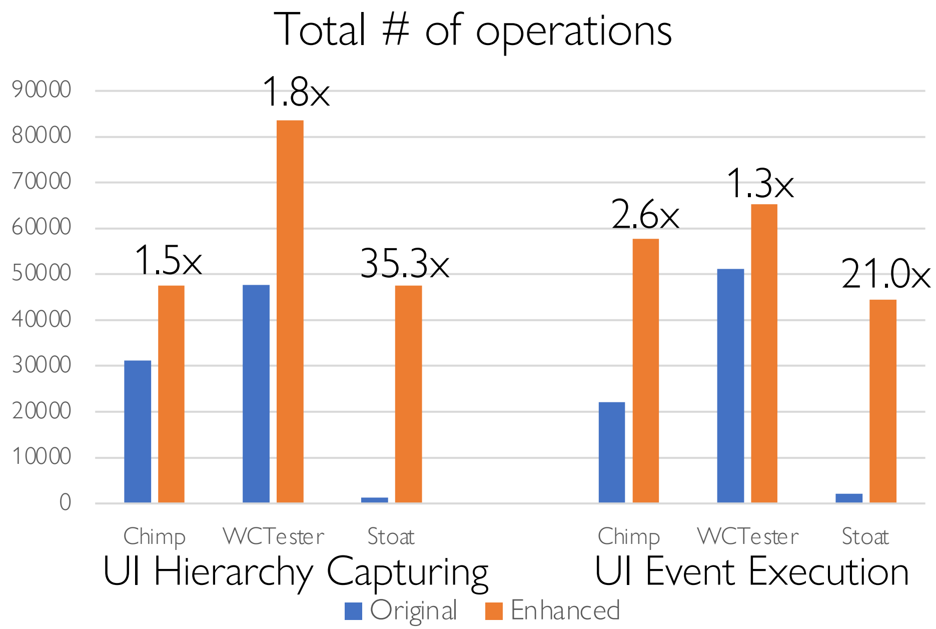
[1] https://wenyu.io/pub/issta21-toller.pdf

RQ1: Efficiency Comparison

- Same testing time, with and without **Toller**
- Fallback to UIAutomator on unhandled cases

Toller *substantially accelerates two types of operations*





RQ2: Code Coverage Improvement

- 3 one-hour runs for each (tool, app)
- Average # of Java methods covered after testing starts

TOLLER-induced coverage improvements are substantial enough to change relative tool competitiveness

+11.8%, 10.4%, 70.1% on CH, WT, and ST

#apps with highest coverage:

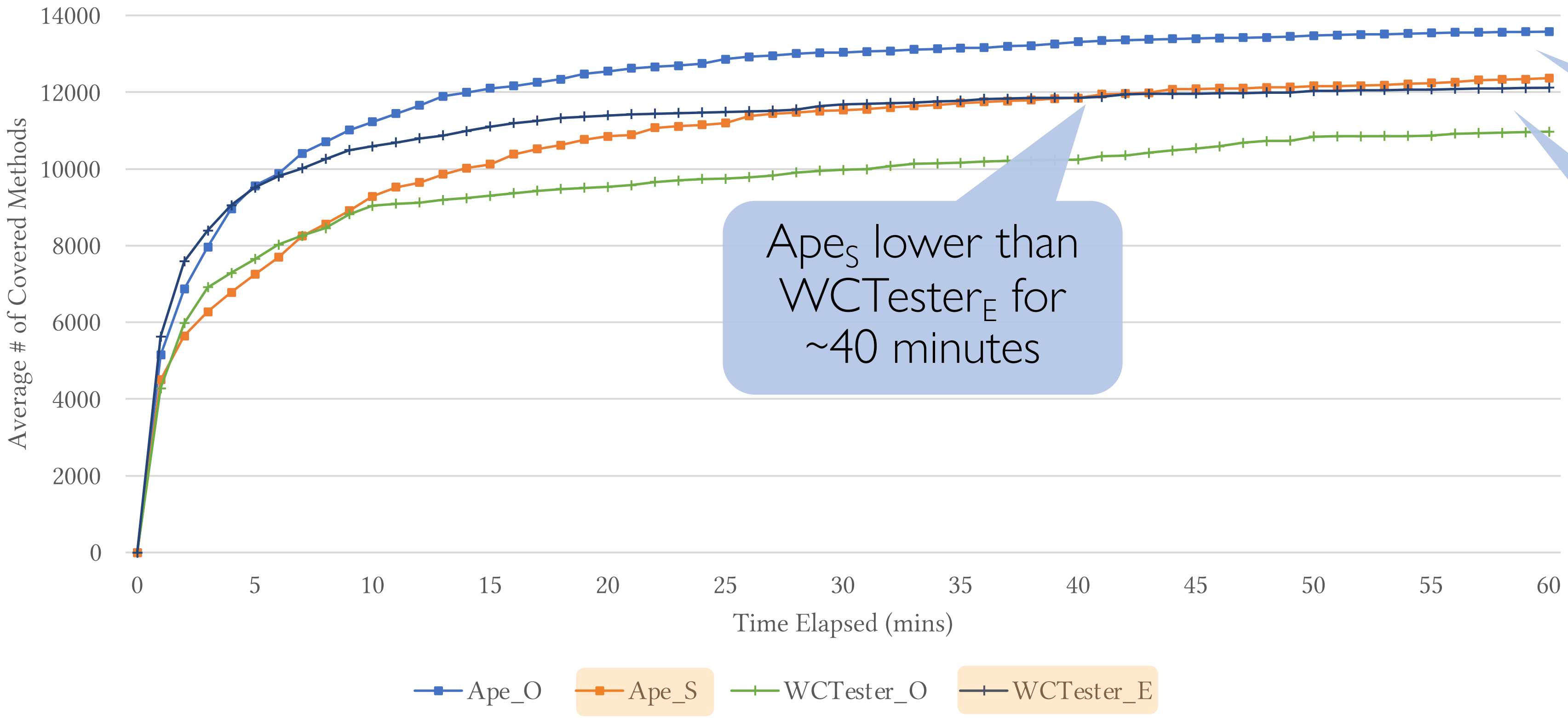
MK	CH	WT	ST
Without TOLLER			
4	8	3	0
With TOLLER			
2	5	5	3

MK = Monkey, CH = Chimp (re-implemented Monkey), WT = WCTester, ST = Stoat

RQ2: Code Coverage Improvement

- Additionally evaluate on *Ape*
  - More advanced algorithm than tools from the 2018 study
  - No mention of leveraging private APIs for UI Hierarchy Capturing
  - **S**low (no efficient infra. support) vs. **O**riginal (with infra. support)

Tools with less advanced algorithm but efficient infra. support could outperform tools with more advanced algorithm but no efficient infra. support



Ape_S lower than WCTester_E for ~40 minutes

9.7% improvement from Ape_S to Ape_O

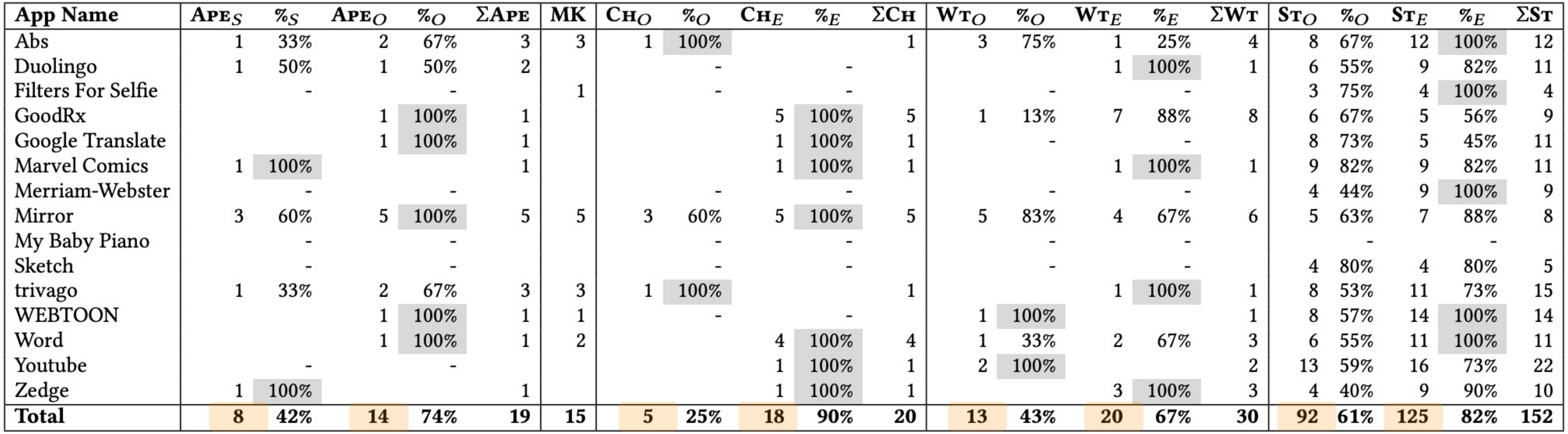
10.4% improvement from WCTester_O to WCTester_E

RQ3: Crash Triggering Improvement

Efficient infrastructure helps tools trigger substantially more crashes

- Cumulative # of distinct crashes, identified by stacktraces
- 3.6x, 1.5x, 1.4x for three enhanced tools; 1.8x for Ape
  - For the majority of (tool, app) pairs, more crashes are found by enhanced tool versions

MK = Monkey, CH = Chimp (re-implemented Monkey), WT = WCTester, ST = Stoat



App Name	Ape$_S$	%$_S$	Ape$_O$	%$_O$	ΣApe	MK	Ch$_O$	%$_O$	Ch$_E$	%$_E$	ΣCh	Wt$_O$	%$_O$	Wt$_E$	%$_E$	ΣWt	St$_O$	%$_O$	St$_E$	%$_E$	ΣSt
Abs	1	33%	2	67%	3	3	1	100%			1	3	75%	1	25%	4	8	67%	12	100%	12
Duolingo	1	50%	1	50%	2			-		-				1	100%	1	6	55%	9	82%	11
Filters For Selfie		-		-		1		-		-			-		-		3	75%	4	100%	4
GoodRx			1	100%	1				5	100%	5	1	13%	7	88%	8	6	67%	5	56%	9
Google Translate			1	100%	1				1	100%	1		-		-		8	73%	5	45%	11
Marvel Comics	1	100%			1				1	100%	1			1	100%	1	9	82%	9	82%	11
Merriam-Webster		-		-				-		-			-		-		4	44%	9	100%	9
Mirror	3	60%	5	100%	5	5	3	60%	5	100%	5	5	83%	4	67%	6	5	63%	7	88%	8
My Baby Piano		-		-				-		-			-		-			-		-	
Sketch		-		-				-		-			-		-		4	80%	4	80%	5
trivago	1	33%	2	67%	3	3	1	100%			1			1	100%	1	8	53%	11	73%	15
WEBTOON			1	100%	1	1		-		-		1	100%			1	8	57%	14	100%	14
Word			1	100%	1	2			4	100%	4	1	33%	2	67%	3	6	55%	11	100%	11
Youtube		-		-					1	100%	1	2	100%			2	13	59%	16	73%	22
Zedge	1	100%			1				1	100%	1			3	100%	3	4	40%	9	90%	10
Total	**8**	**42%**	**14**	**74%**	**19**	**15**	**5**	**25%**	**18**	**90%**	**20**	**13**	**43%**	**20**	**67%**	**30**	**92**	**61%**	**125**	**82%**	**152**

Recap & Conclusion

- Over 70% of testing time budget is for Android testing tools' use of test infrastructure
- Use of test infrastructure can be made much more efficient with TOLLER
  - 10.4% - 70.1% code coverage improvement, 1.4x - 3.6x unique crashes detected depending on tool
- Code and data available at https://github.com/TOLLER-Android/main

Efficient infrastructure support is useful for effective Android UI testing tools, complementary with existing algorithmic advances