南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Taming Android Compatibility Issues with Software Big Data

Yepang LIU

CSE Department, SUSTech

liuyp1@sustc.edu.cn

Joint work with the Hong Kong University of Science and Technology and Peking University

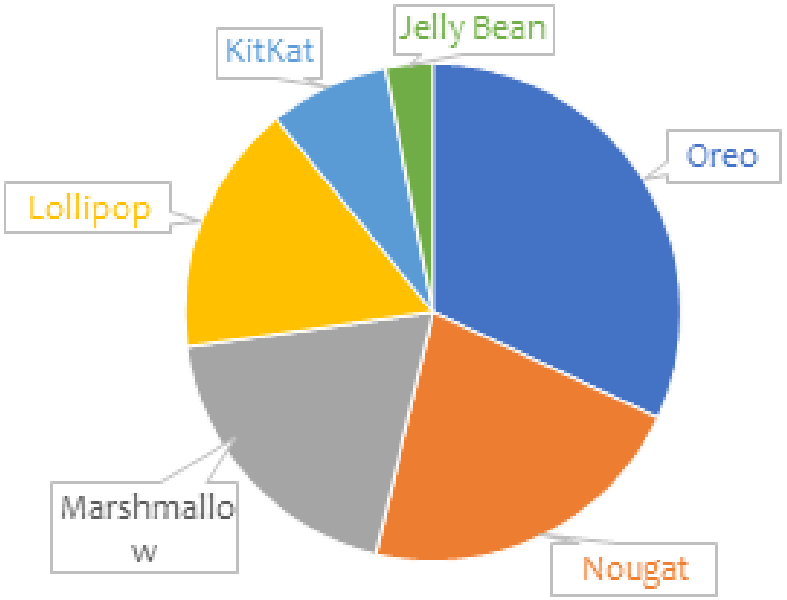
Android is the most popular mobile OS

Android	iOS	KaiOS
75.27%	22.74%	0.75%

Unknown	Windows	Samsung
0.32%	0.24%	0.22%

Mobile Operating System Market Share Worldwide - May 2019 (By StatCounter)

Fragmented Ecosystem



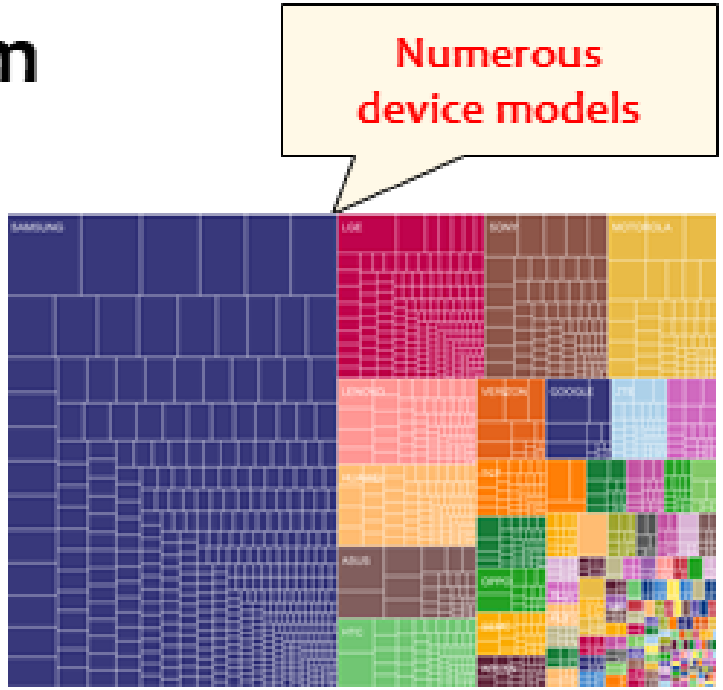
10+ widely-used API levels

Version	Code Name	API	Distribution
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%

Data collected during a 7 day period ending on May 7, 2019

Fragmented Ecosystem

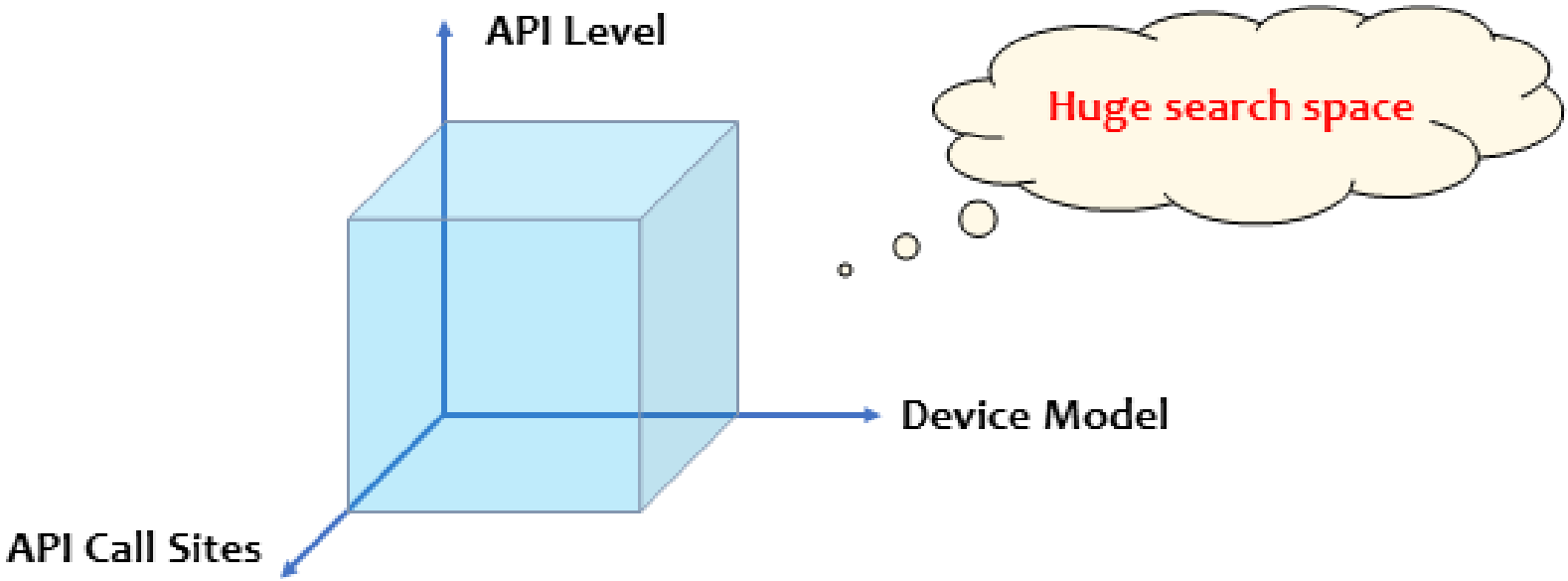
100+ Android certified vendors



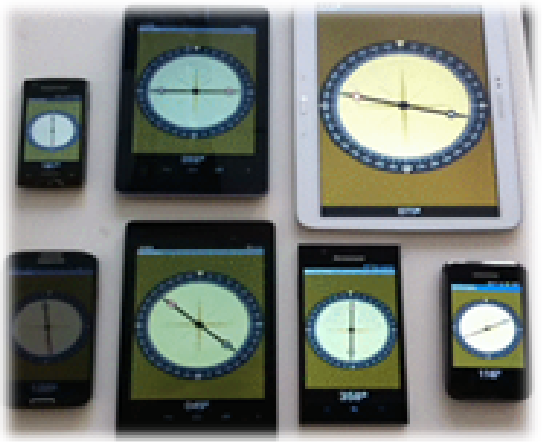
Over **24,000** distinct device models

(Aug 2015, Open Signal)

Compatibility testing is non-trivial



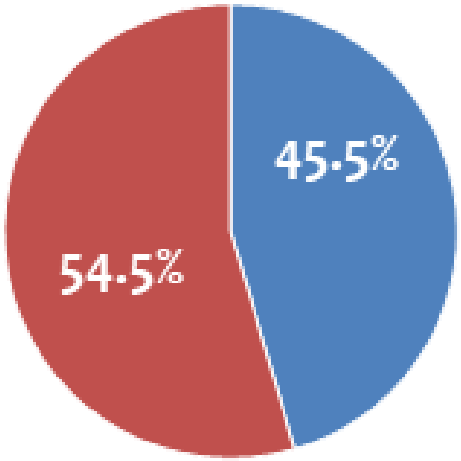
68.8% Android developers encountered various compatibility issues (Wei et al. 2018)



Annoying compatibility issues

Frustrated users and developers

Compatibility Issue Types (Wei et al. 2018)



Studied 220 bugs in total

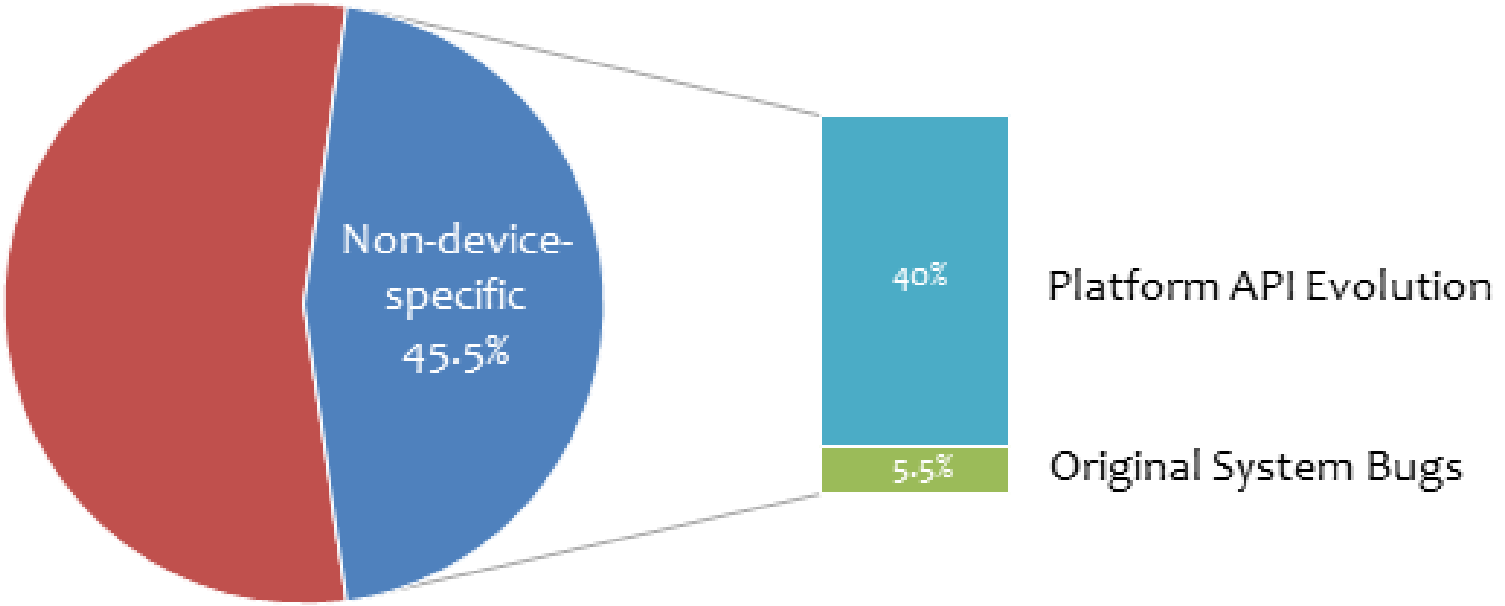
Non-device-specific issues (45.5%):

The issues can occur on different device models

Device-specific issues (54.5%):

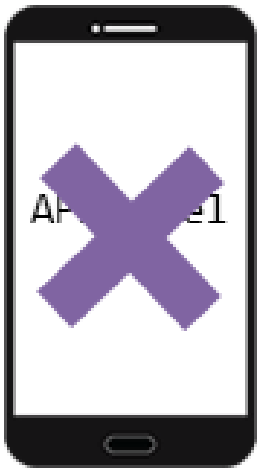
The issues can be triggered only on certain device models

Non-Device-Specific Compatibility Issues



Non-Device-Specific Issue Example

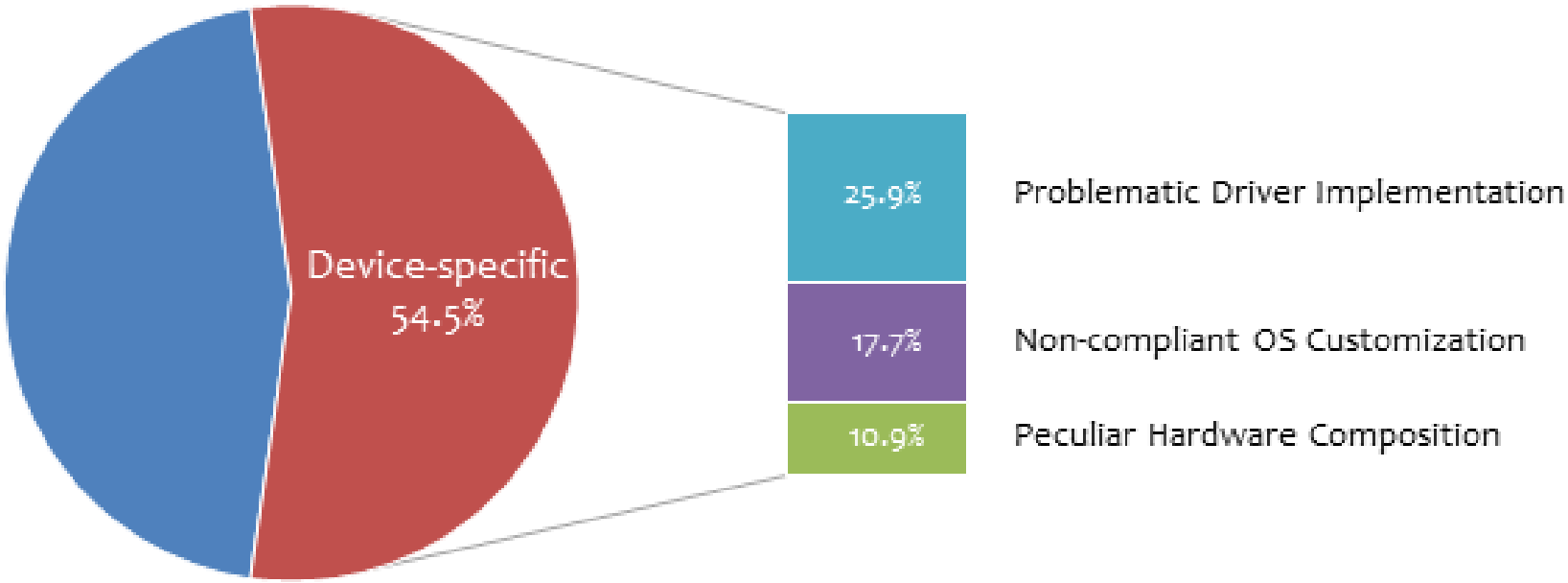
AnkiDroid pull request 130



API (Added in API Level 16):

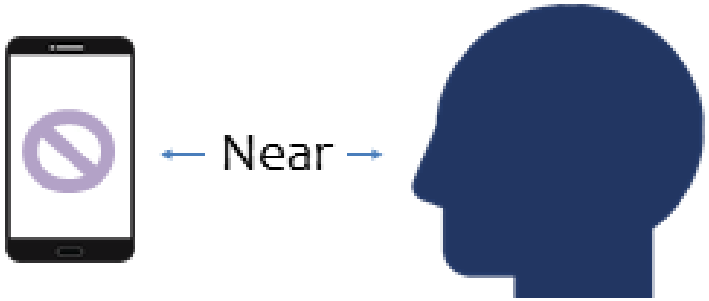
```
SQLiteDatabase.disableWriteAheadLogging()
```

Device-Specific Compatibility Issues



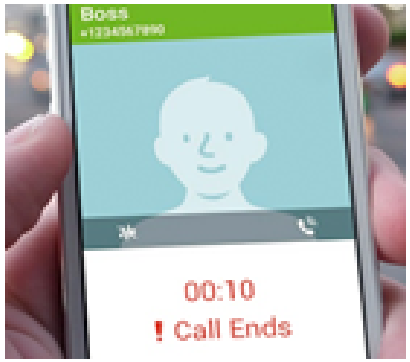
Problematic Driver Implementation Example

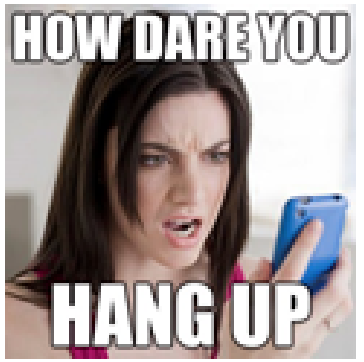
Proximity sensor: Report the distance between device and its surrounding objects



CSipSimple Issue 353

- Samsung SPH-M900's proximity sensor API reports a value **inversely** proportional to the real distance





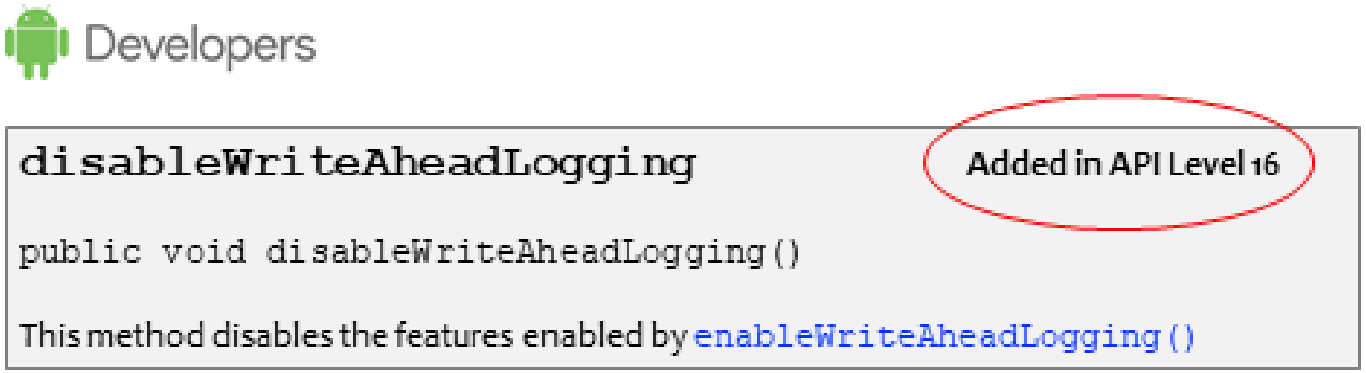
Observation: Knowing the correlations between (1) APIs and API levels, (2) APIs and device models help compatibility analysis and issue fixing

```
1.  + if (android.os.Build.VERSION.SDK_INT >= 16) {
2.        sqLiteDatabase.disableWriteAheadLogging();
3.  + } // invoke the API only on API level 16+
```

```
1.    boolean proximitySensorActive = getProximitySensorState();
2.  + boolean invertProximitySensor =
3.      android.os.Build.PRODUCT.equalsIgnoreCase("SPH-M900");
4.  + if (invertProximitySensor) {
5.        proximitySensorActive = !proximitySensorActive;
6.  + } // reverse distance detection result on SAMSUNG SPH-M900
```

Correlation Mining

- Mining the correlations between APIs and API levels is not difficult.



Correlation Mining

- Mining the correlations between APIs and API levels is not difficult.
- However, mining API-device correlations is hard since Android system customizations are often closed-source.

Would the system customizations affect the behavior of an Android API ?

PIVOT: API-Device Correlator

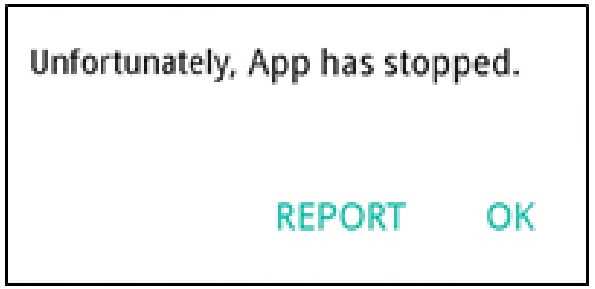
- The first fully automated solution for learning API-device correlations
- Does not require analyzing customized system code. Instead, it leverages Android apps, which are easily available.
- Satisfactory precision and can find useful API-device correlations

Observation

- Compatibility issues are often handled by **exercising an alternative execution path** if the running device matches certain issue-triggering models (Wei et al. 2016)

An Infamous Issue on LG Devices

- Pressing the menu button on some LG devices will lead to **app crashes** if the app customizes the options menu.



The Patched Code

```
public boolean onKeyUp(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_MENU) &&
            isLGE()) {
        Log.i(TAG, "Applying LG workaround");
        openOptionsMenu();
        return true;
    }
    //App-specific code cloned from VLC
    View v = getCurrentFocus();
    …
    return super.onKeyUp(keyCode, event);
}
public boolean isLGE() {
    return Build.MANUFACTURER.compareTo("LGE") == 0;
}
```

Execute on LG devices

Execute on other devices

Solution: Invoking openOptionsMenu() instead of onKeyUp() helps avoid crashes

The Patched Code

```
public boolean onKeyUp(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_MENU) &&
            isLGE()) {
        Log.i(TAG, "Applying LG workaround");
        openOptionsMenu();
        return true;
    }
    //App-specific code cloned from VLC
    View v = getCurrentFocus();
    …
    return super.onKeyUp(keyCode, event);
}
public boolean isLGE() {
    return Build.MANUFACTURER.compareTo("LGE") == 0;
}
```

API

Device condition

<openOptionsMenu, LGE>

APIs and devices can be correlated via analyzing patches to help avoid compatibility analysis

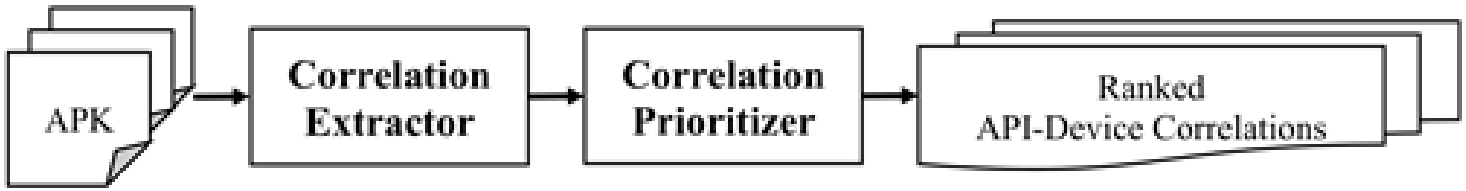
Solution: Invoking openOptionsMenu() instead of onKeyUp() helps avoid crashes

Feasibility Analysis

Learning API-device correlations from patches seems possible, but where to find such patches?

Companies of mature apps are likely to notice and patch issues early because of their large user base and rich QA resources

Workflow of PIVOT



- PIVOT learns API-device correlations via analyzing a large number of popular and highly-rated apps such as Facebook
- It prioritizes the correlations to suppress noises, which can be massive

Challenge 1

- Existing API precondition mining techniques (such as Nguyen et al. 2014) based on **frequent pattern mining algorithms** cannot effectively mine API-device correlations.

- **Assumption:** In an app corpus, the majority of API usages are correct. 

Unfortunately, compatibility issues are commonly left undetected in released apps (i.e., **majority of uses are incorrect**).

Challenge 2

- Noises caused by **irrelevant APIs**

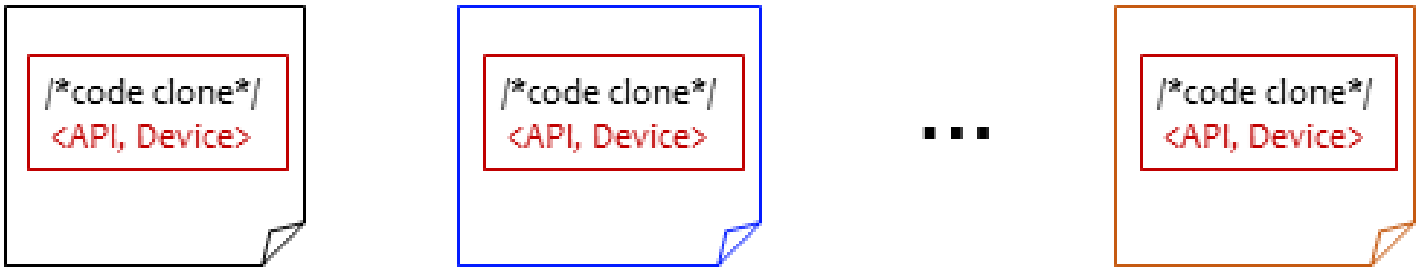
```
if ((keyCode == KeyEvent.KEYCODE_MENU) &&
        isLGE()) {
    Log.i(TAG, "Applying LG workaround");
    openOptionsMenu();
    return true;
}
```

Up to **97%** correlations learned in the first step are noises!!!

APIs irrelevant to compatibility issues may be called for app-specific purposes

Challenge 3

- Due to **code clones** and **library usages (e.g., Ad SDK)**, noisy API-device correlations can recur in different apps (high support values)



The In-App-Confidence Metric

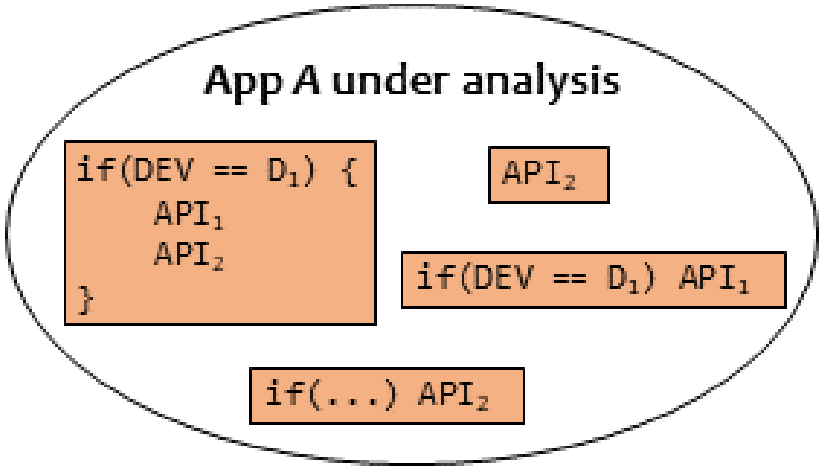
- Observation 1: Although compatibility issues may not be commonly fixed, once developers identify an issue, they tend to fix all problematic API callsites.
- Observation 2: The callsites of irrelevant APIs are often not guarded by any device conditions.

```
protected void onCreate (Bundle savedInstanceState) {
    Log.i(TAG, "Activity created");
    …
}
```

The In-App-Confidence Metric

$$IAC(A, c) = \frac{\#\ occurrences\ of\ c\ in\ A}{\#\ callsites\ of\ c.api\ in\ A}$$

A: an app
C: an API-device correlation

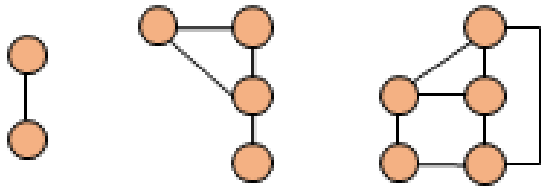


$$IAC(A, < API_1, D_1 >) = \frac{2}{2} = 1.00$$

$$IAC(A, < API_2, D_1 >) = \frac{1}{3} = 0.33$$

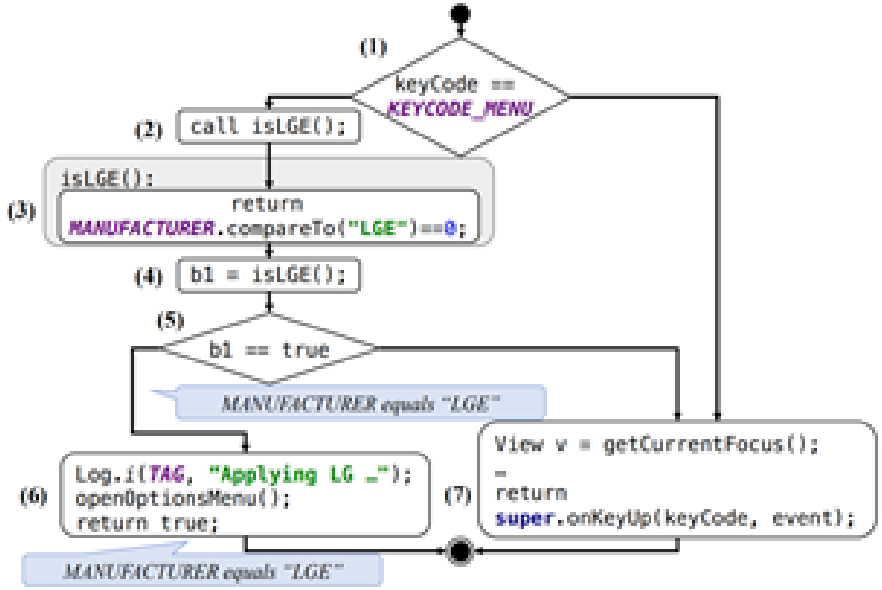
The Occurrence Diversity Metric

- Real API-device correlations should be observed in different apps/methods
- Measure occurrence diversity of a mined correlation c using Shannon Index (Shannon 1949, for measuring the diversity of characters in a string) at different granularities
  - App level: c should originate from apps of different companies
  - Method level: c should originate from different methods (analyzing control flow structures)



Illustrative Example

```
public boolean onKeyUp(int keyCode, KeyEvent event) {
    if ((keyCode == KeyEvent.KEYCODE_MENU) &&
            isLGE()) {
        Log.i(TAG, "Applying LG workaround");
        openOptionsMenu();
        return true;
    }
    //App-specific code cloned from VLC
    View v = getCurrentFocus();
    …
    return super.onKeyUp(keyCode, event);
}
public boolean isLGE() {
    return Build.MANUFACTURER.compareTo("LGE") == 0;
}
protected void onCreate (Bundle savedInstanceState) {
    Log.i(TAG, "Activity created");
    …
}
```



<Log.i(String, String), "LGE"> ✗ Low in-app-confidence value
(not guarded by device condition in onCreate)

<Activity.getCurrentFocus(), "LGE"> ✗ Low occurrence diversity value
(originate from many clone code snippets)

<Activity.openOptionsMenu(), "LGE"> ✓ Highly ranked
(related to real compatibility issues)

Evaluation

- **RQ1 (Effectiveness):** Can PIVOT effectively identify API-device correlations from large-scale Android app corpuses?

- **RQ2 (Usefulness):** Can the API-device correlations learned by PIVOT help compatibility issue detection?

Experiment 1

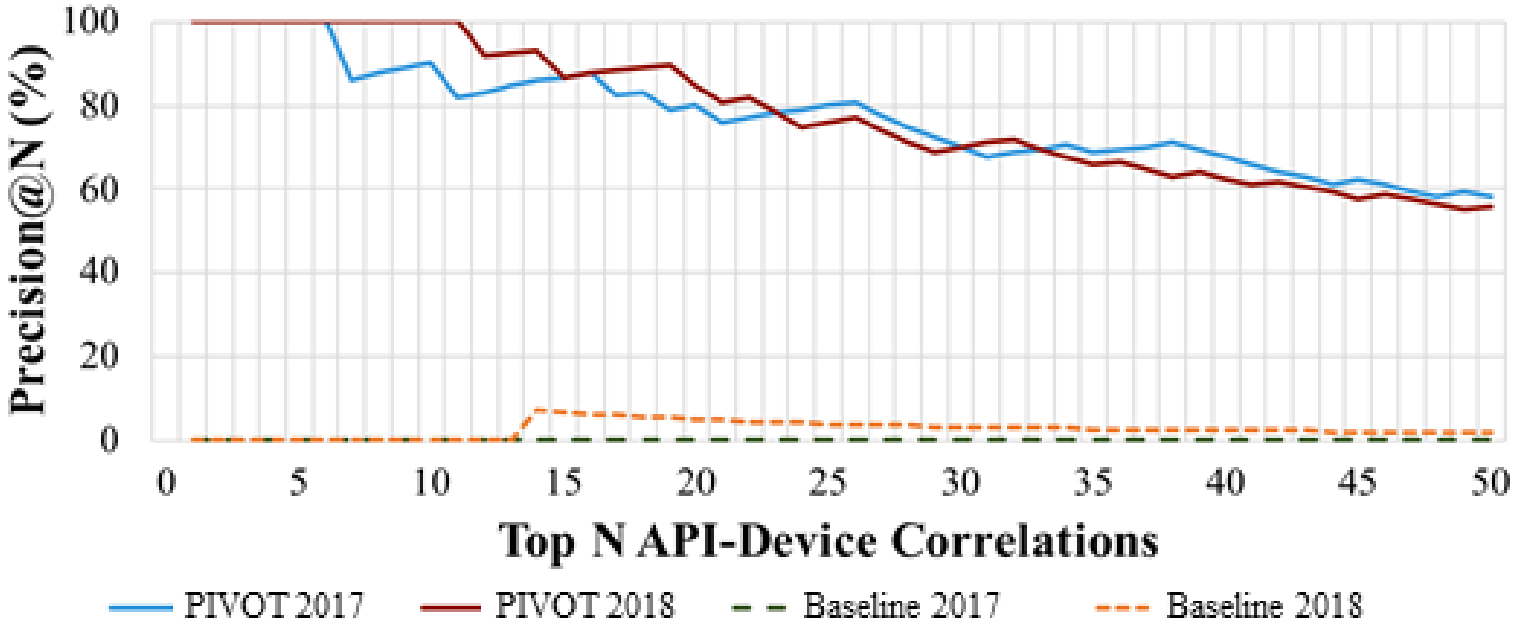
- **Two app corpuses** of top 100 apps in each category on Google Play
  - Nov 2017 (2524 apps): 14.3 million classes, 112.2 million methods
  - June 2018 (3145 apps): 20.0 million classes, 108.9 million methods
- **Baseline:** A representative API precondition miner (Nguyen et al. 2014)
- **Ground truth:** manually established ground truth by checking the top 50 API-device correlations mined by each method
- **Evaluation metric:** Precision@N

RQ1: Effectiveness

Precision@5 = 100%

Precision@10 = 90%+

Precision@50 = 56%+

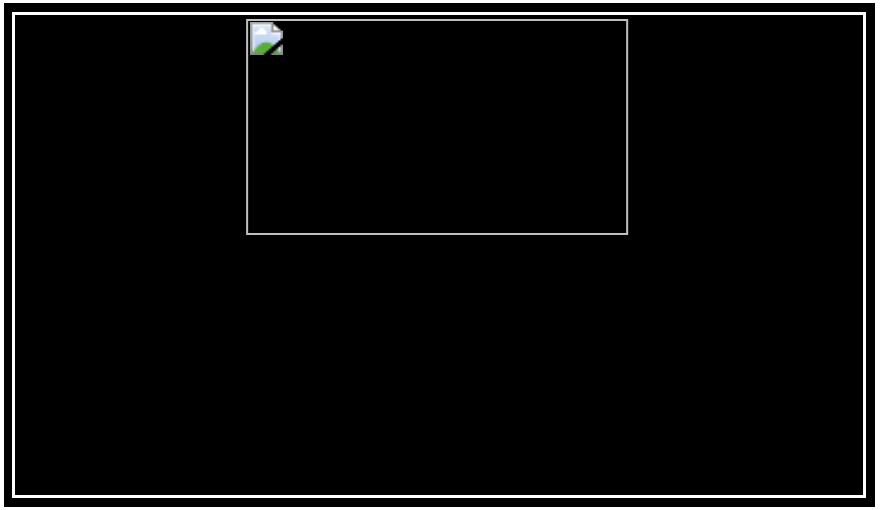


Experiment 2

- The 49 API-device correlations in top-50 correlations mined from the two corpuses helped build an archive of 11 compatibility issues
- Selected 5 issues whose concerned devices are available on Amazon Device Farm (https://aws.amazon.com/device-farm/) or WeTest (https://wetest.qq.com/) and incurred clearly inconsistent app behaviors (e.g., crash)
- Used FicFinder (Wei et al., 2016) to detect new issue instances in 44 app subjects randomly selected from top apps indexed by F-Droid

RQ2: Usefulness

- Found 19 compatibility issues in 10 apps, reproduced 10 issues
- Reported the 10 issues to devs (7 confirmed, 4 fixed)



Conclusion

- PIVOT is the first technique that automatically learns API-device correlations via code analysis
- Two sets of novel metrics for correlation prioritization
- Evaluation results show that PIVOT achieves satisfactory precision
- The API-device correlations learned by PIVOT are useful in detecting real compatibility issues

Related Publications

- L. Wei, Y. Liu, S.C. Cheung. **Taming Android Fragmentation: Characterizing and Detecting Compatibility Issues for Android Apps.** In Proceedings of ASE 2016. ACM SIGSOFT Distinguished Paper Award.
- L. Wei, Y. Liu, S.C. Cheung, H. Huang, X. Lu, X. Liu. **Understanding and Detecting Fragmentation-Induced Compatibility Issues for Android Apps.** In TSE 2018.
- L. Wei, Y. Liu, S.C. Cheung. **PIVOT: Learning API-Device Correlations to Facilitate Android Compatibility Issue Detection.** In Proceedings of ICSE 2019. ACM SIGSOFT Distinguished Artifact Award.