University of Luxembourg

Multilingual. Personalised. Connected.

Datasets, AI, and Static analysis for Mobile App Analysis

PROMISE 2025, Trondheim, Norway

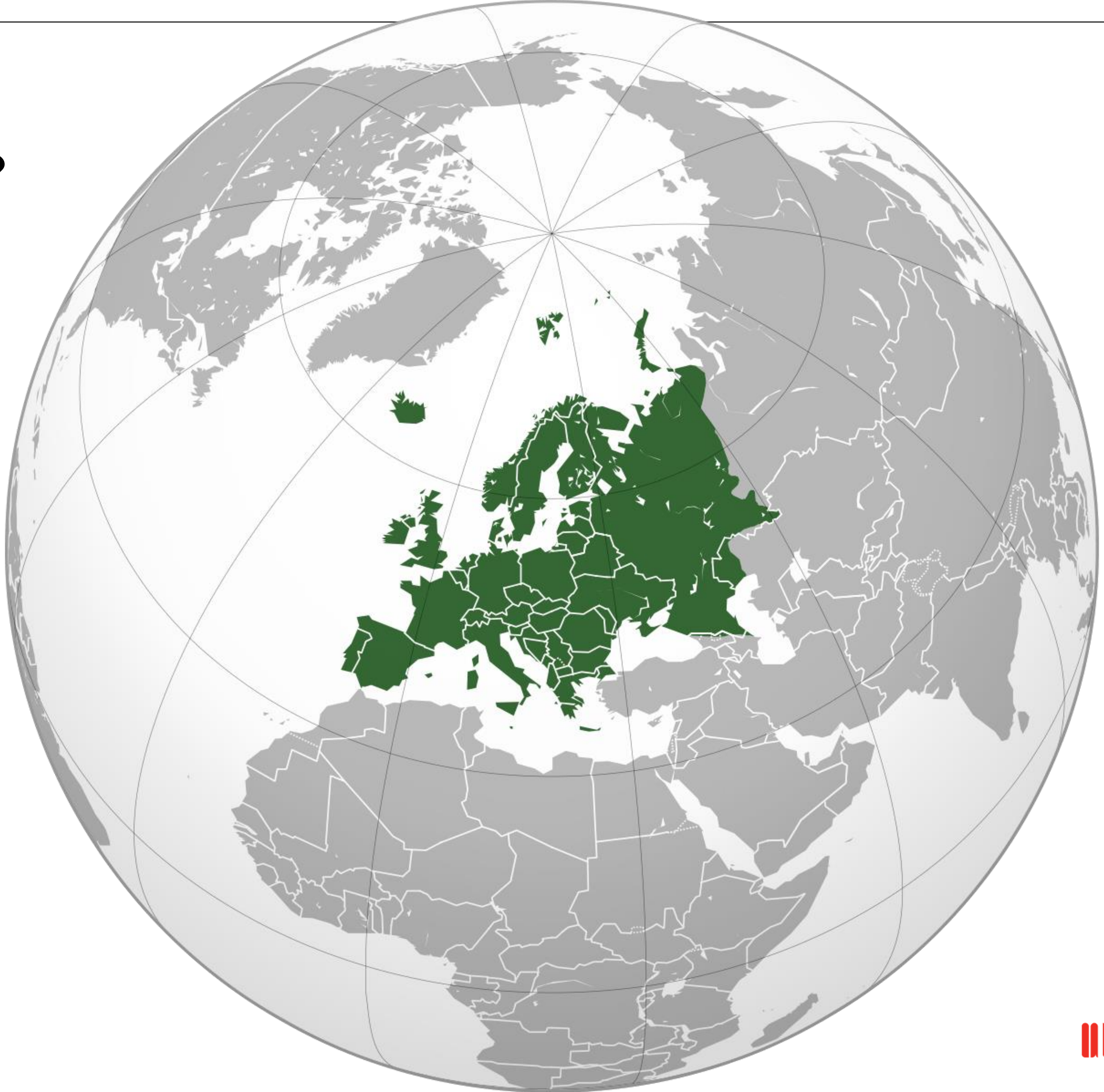
Prof. Dr. Jacques Klein, June 2025

UNIVERSITÉ DU LUXEMBOURG

Where is Luxembourg?

UNIVERSITY OF LUXEMBOURG

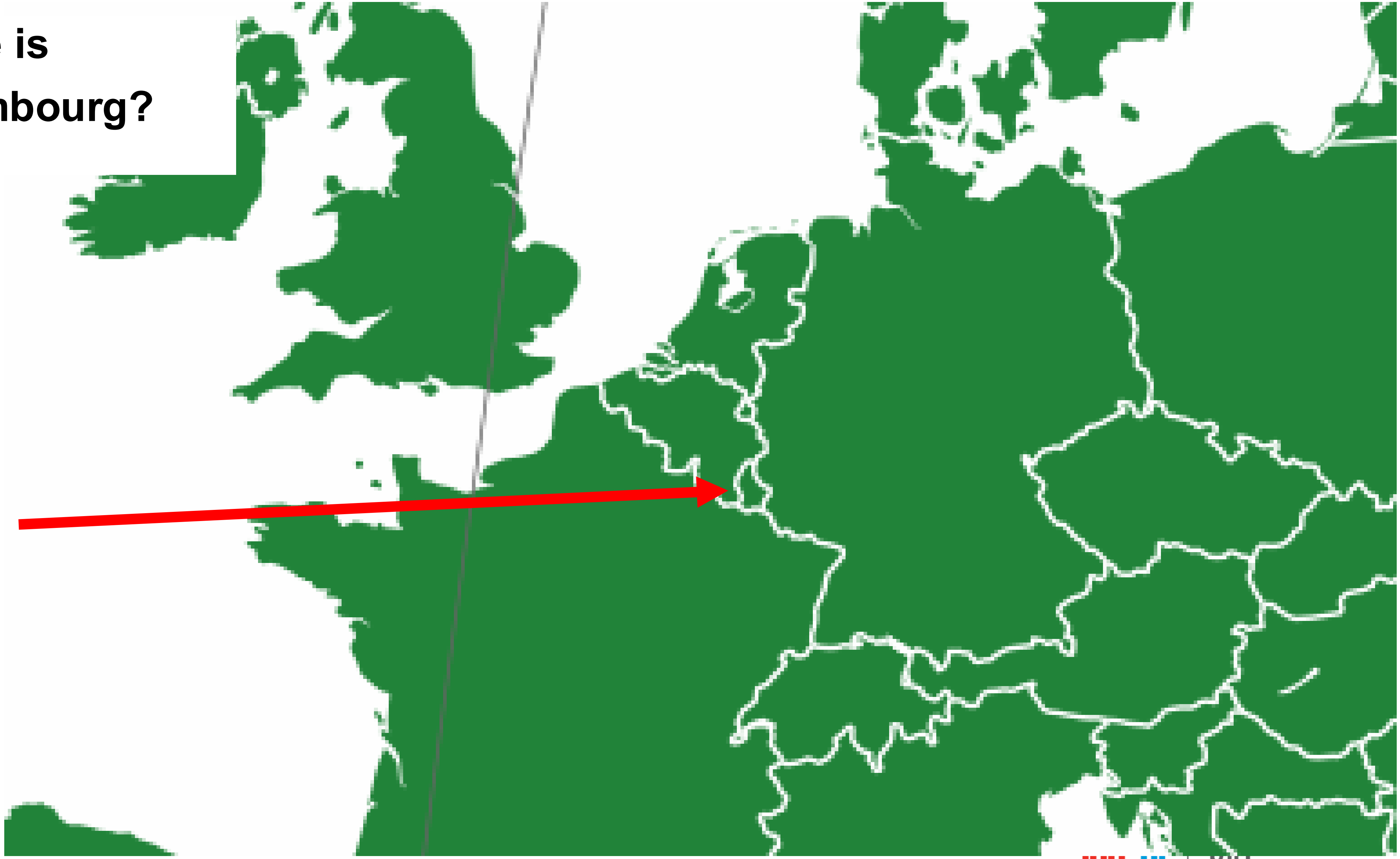
Where is Luxembourg?



Where is Luxembourg?

Where is Luxembourg?

Where is Luxembourg?



The University of Luxembourg

The University of Luxembourg is a research university with a distinctly **international**, **multilingual** and **interdisciplinary** character.

The University's ambition is to provide the **highest quality research** and teaching in its chosen fields and to generate a positive scientific, educational, social, cultural and societal impact in Luxembourg and the Greater Region.

UNIVERSITY OF LUXEMBOURG

Ranked

12^{th} Young University

worldwide and #1 worldwide for its "international outlook" in the Times Higher Education (THE) World University Rankings 2020

~7000 students

~1000 PhDs

270 faculty members

129 nationalities

56% international students

Trustworthy Software Engineering
TruX Research Group

Prof. Tegawendé F. BISSYANDE

Prof. Jacques KLEIN

UNIVERSITY OF LUXEMBOURG

TruX

SNT

Trustworthy Software Engineering
TruX Research Group

Prof. Tegawendé F. BISSYANDE

Prof. Jacques KLEIN

Dr. Jordan SAMHI

UNIVERSITY OF LUXEMBOURG

TruX People

Professors

- Tegawendé F. BISSYANDE (head)
- Jacques KLEIN (co-head)

R&D Specialists

1. Laura Bernardy

Research Scientist

1. Jordan SAMHI

Assistant

- Fiona LEVASSEUR

Research Associates

1. Yinghua LI
2. Tiezhu SUN
3. Aleksandr PILGUN
4. Olatunji IYIOLA (Emmanuel)
5. Navid KHALEDIAN
6. Tialia MALLOY

Coming Soon

1. El-Hacen DIALLO

PhD Students

1. Fatou Ndiaye MBODJI (Apr. 2021)
2. Xunzhu TANG (Oct. 2021)
3. Damien FRANCOIS (Nov. 2021)
4. Weiguo PIAN (Jan 2022)
5. Alioune DIALLO (Feb. 2022)
6. Christian OUEDRAOGO (Apr. 2022)
7. Aicha WAR (May 2022)
8. Yewei SONG (Jun. 2022)
9. Despoina GIARIMPAMPA (Sep. 2022)
10. Marco ALECCI (Oct. 2022)
11. Fred PHILIPPY (Mar. 2023)
12. Jules WAX (Mar. 2023)
13. Moustapha DIOUF (Apr. 2023)
14. Micheline MOUMOULA (Oct. 2023)
15. Pedro RUIZ JIMÉNEZ (Nov. 2023)
16. Omar EL BACHYR (Feb. 2024)
17. Prateek RAJPUT (Mar. 2024)
18. Albérick DJIRE (Mar. 2024)
19. Maimouna Tamah DIAO (Apr. 2024)
20. Maimouna OUATTARA (May 2024)
21. Aziz BONKOUNGOU (Jul. 2024)
22. Serge Lionel NIKIEMA (Jul. 2024)
23. Loic TALEB (Dec, 2024)
24. Pawel BORSUKIEWICZ (Dec. 2024)

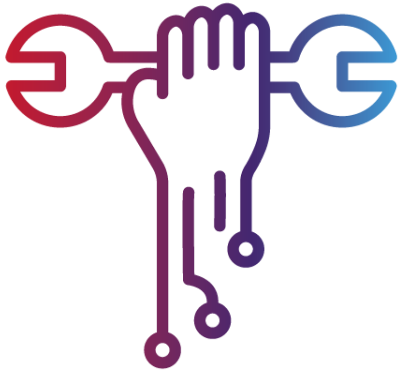
We specialize in Software Research

TruX

Software
Security

TruX

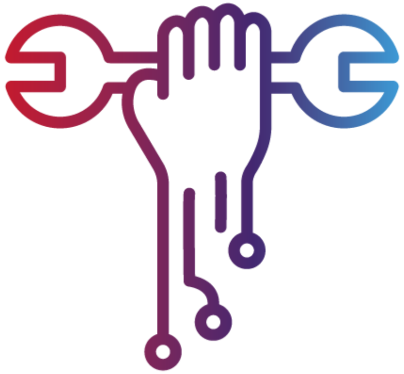
Software Security



Software Debugging

TruX

Software
Security



Software
Debugging

Software
Analytics

Mobile App Analysis

SnT

~~Mobile~~ App Analysis

UNIVERSITY OF LUXEMBOURG

Android App Analysis

Why Android App Analysis is important?

More than 6 billion **people** own a smartphone

Almost three-quarters are Android-based

We manipulate a lot of sensitive data

Just a fraction of 2024!

1

Large Set of Mobile Apps

2

Malware Detection

3

Mobile App Static Analysis

1

Large Set of Mobile Apps

2

Malware Detection

3

Mobile App Static Analysis

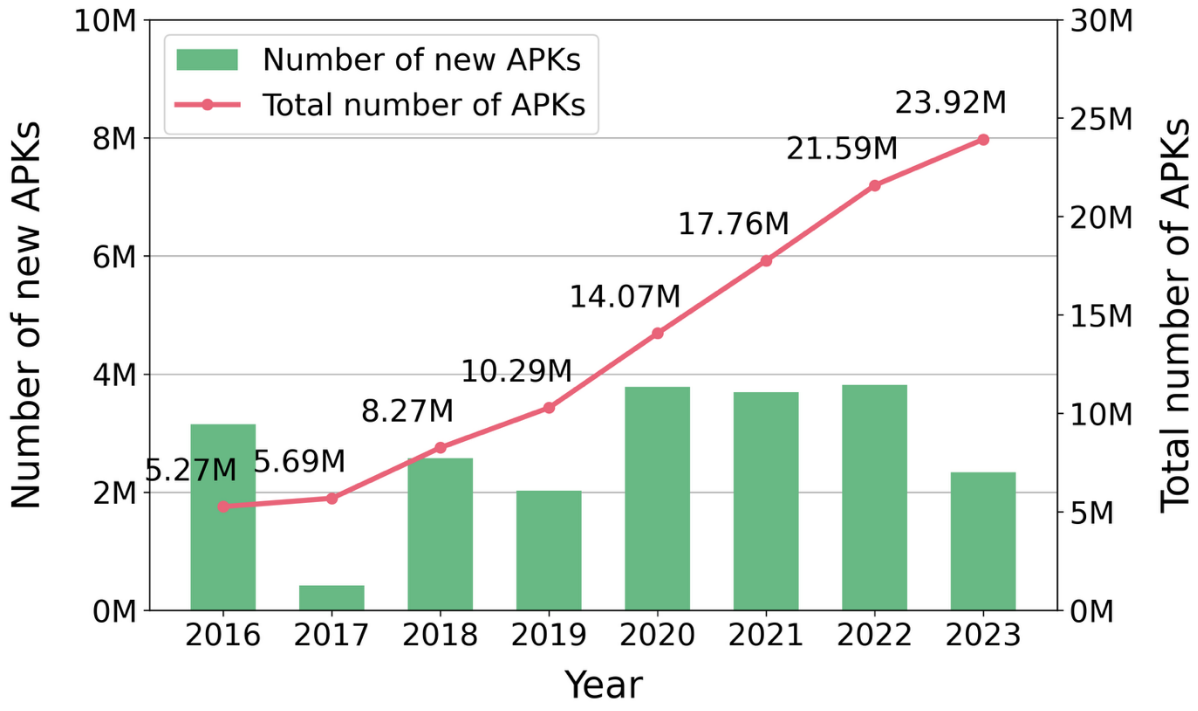
AndroZoo
A repository of Android Apps

[MSR 2016] AndroZoo: Collecting Millions of Android Apps for the Research Community

AndroZoo: A Retrospective

AndroZoo is currently the **biggest dataset of Android apps, with 24 million entries.**
It was created in **2016 at the University of Luxembourg.**

Constantly growing



[MSR 2024]: AndroZoo: A Retrospective with a Glimpse into the Future

AndroZoo: A Retrospective

App ≠ Apk

24 million apks, but 8 708 304 apps (average of 2.7 apks for each app)

Table 1: Top 10 apps by number of APKs

Package Name	#APKs
com.chrome.canary	1986
org.mozilla.fenix	1811
wp.wpbeta	910
dating.app.chat.flirt.wgbcv	826
com.blackforestapppaid.blackforest	822
com.brave.browser_nightly	787
com.topwar.gp	728
com.opodo.reisen	688
com.edreams.travel	679
com.styleseat.promobile	675

Table 2: Lifespan of apps in AndroZoo

#Years	#Apps	#Years	#Apps	#Years	#Apps
10	9347	6	37 099	2	315 206
9	20 072	5	84 931	1	432 536
8	20 171	4	108 962	0	2 732 016
7	37 378	3	186 800		

[MSR 2024]: AndroZoo: A Retrospective with a Glimpse into the Future

AndroZoo: A Retrospective

From November 2021 to November 2023:

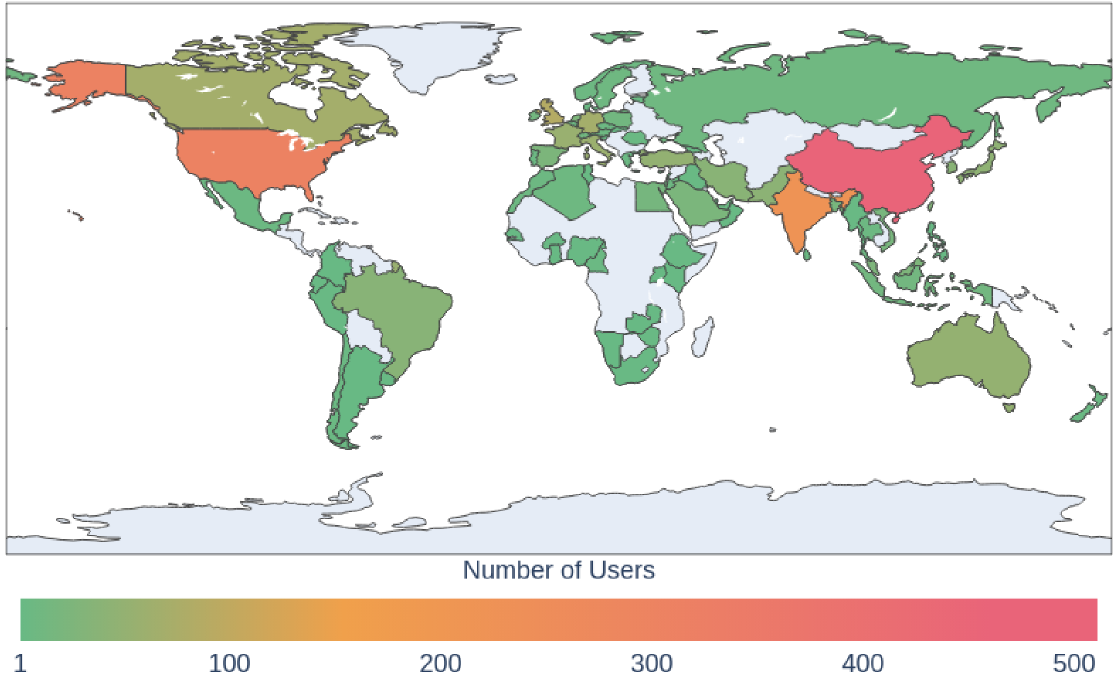
365 604 948 download requests from 692 different users

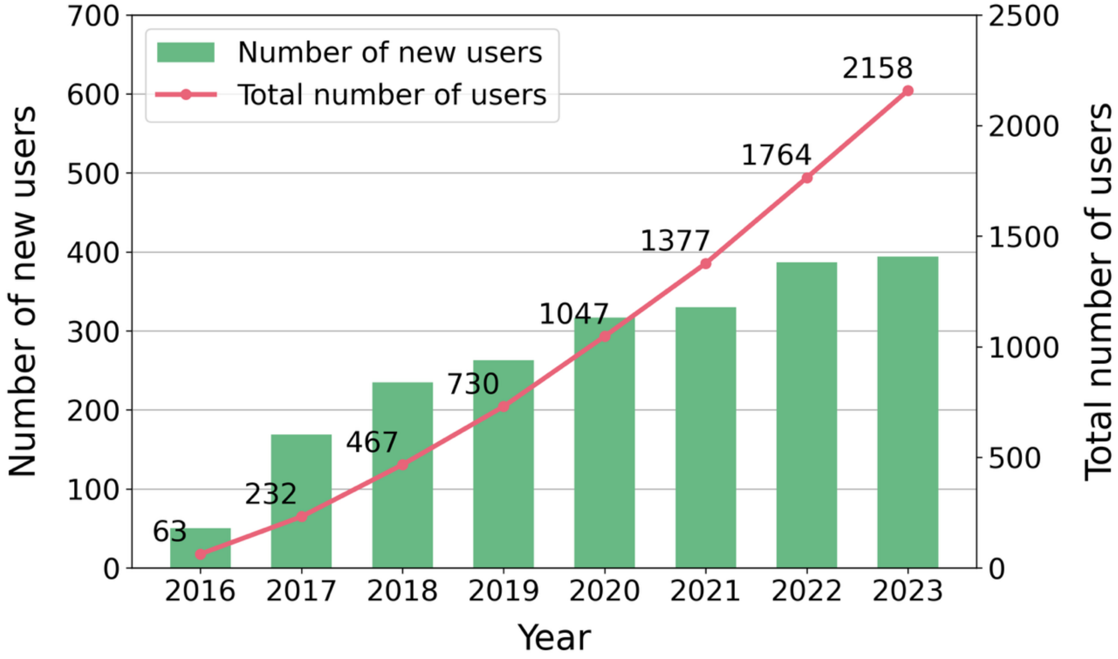
=> 4 PiB of data sent

[MSR 2024]: AndroZoo: A Retrospective with a Glimpse into the Future

AndroZoo: A Retrospective

AndroZoo is currently used by **more then 2000 users worldwide.**





AndroZoo: A Glimpse into the Future

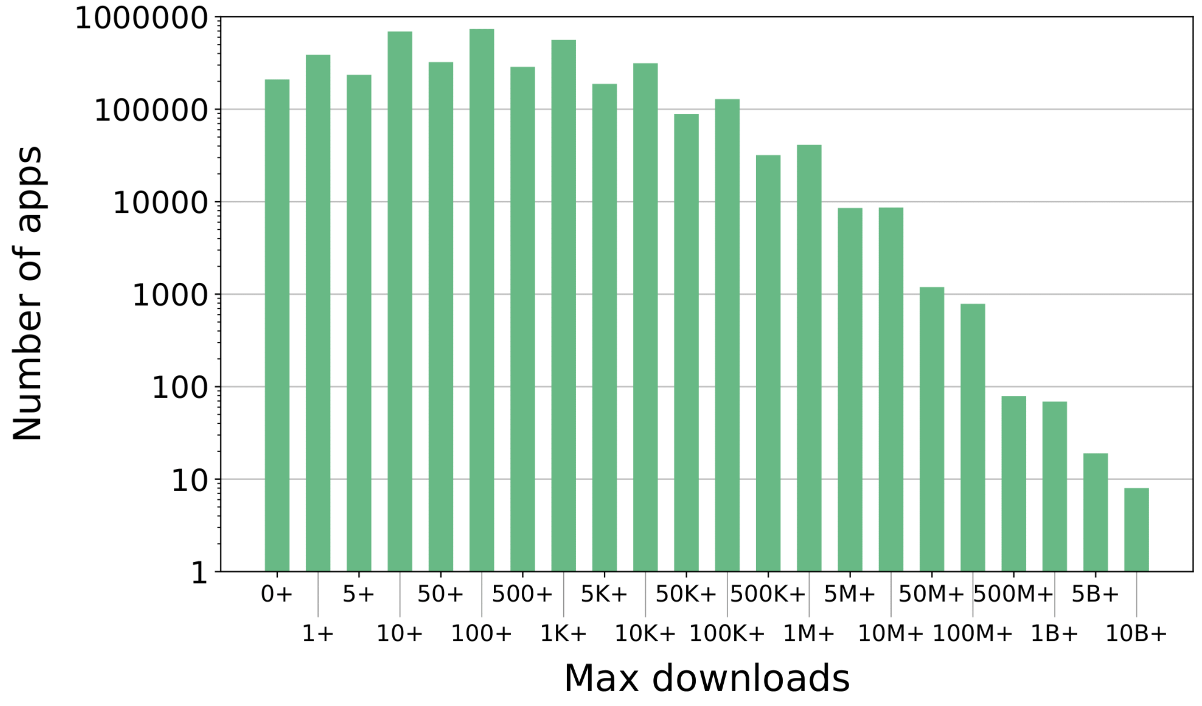
NEW

We started **collecting metadata since 2020,** and we are now **releasing them in AndroZoo** together with the apps.

EXAMPLE

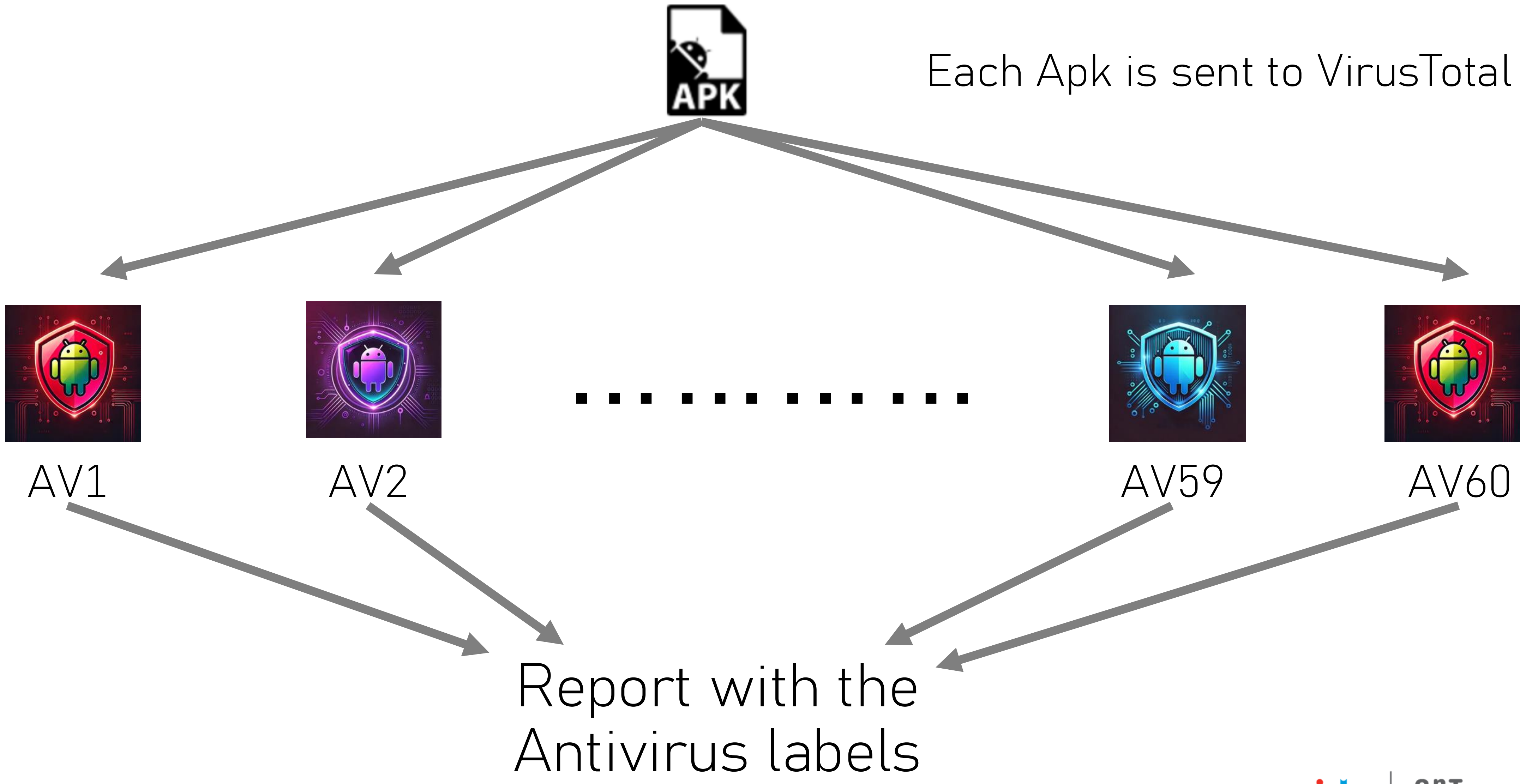
A few examples:

- Description
- Number of Downloads
- Ratings
- Permissions
- Upload Date
- Privacy Policy Link
- …. many others ….



What can you do with AndroZoo?

AndroZoo for Malware Investigation



AndroZoo for Malware Investigation

On 21,570,017 apks from Google Play
sent to VirusTotal,
85,782 have been tagged
by at least 10 Antivirus products

What can you do with AndroZoo?

Another Example

AndroZoo for Large Scale Empirical Studies

Let's start with a simple question

AndroZoo for Large Scale Empirical Studies

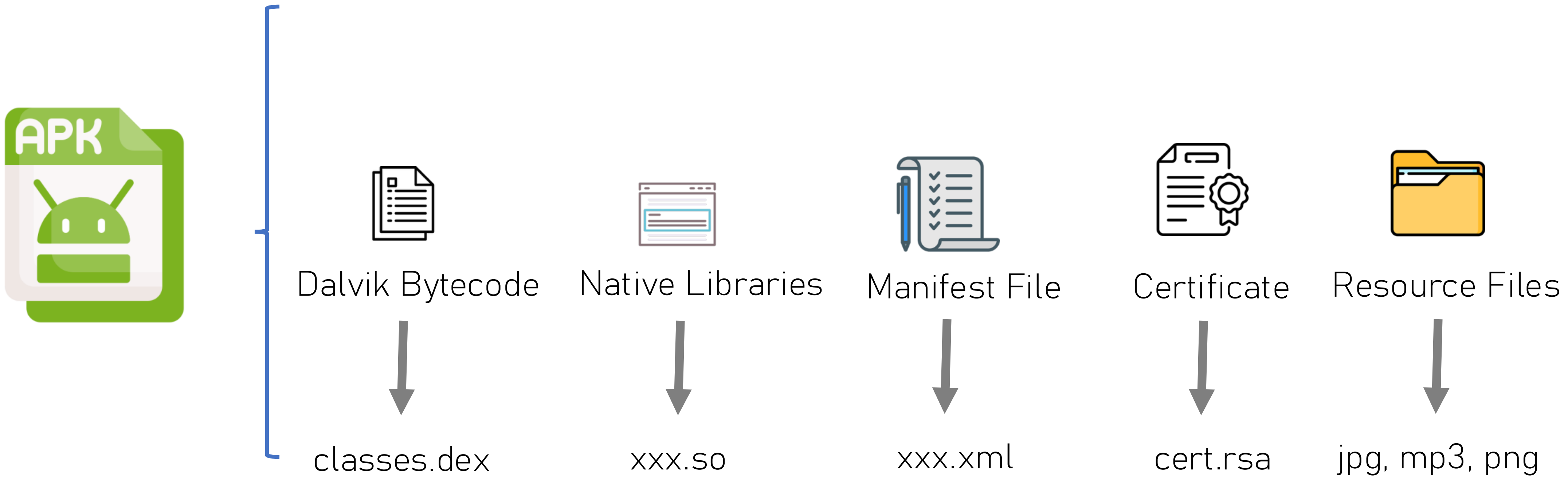
Let's start with a simple question

Do you know what is inside an Android App?

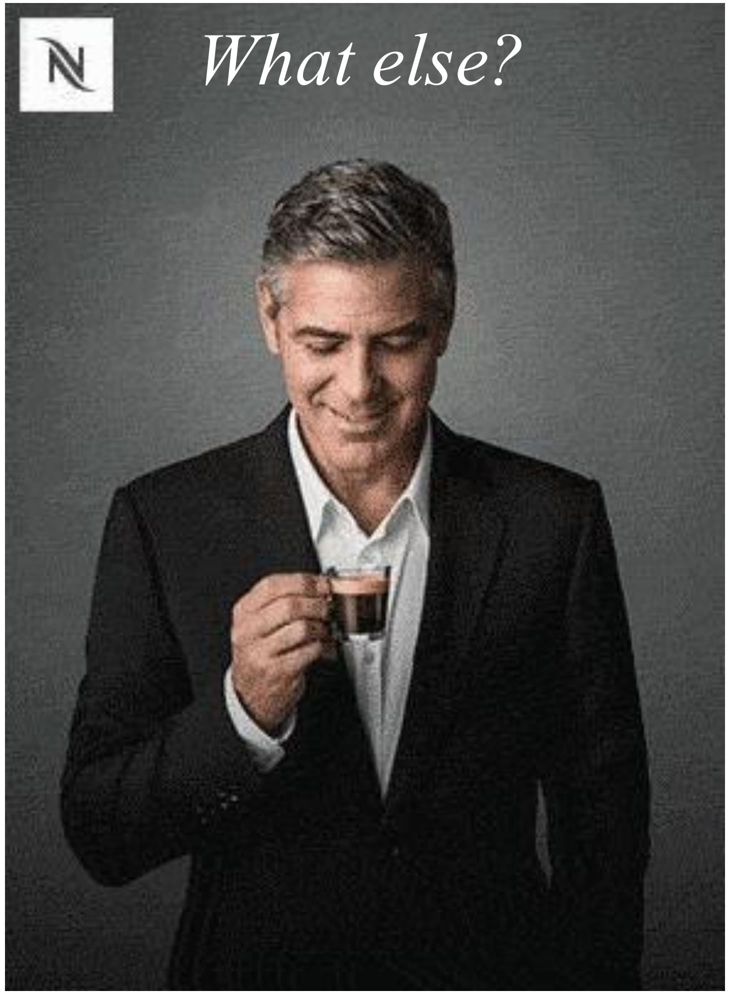
AndroZoo for Large Scale Empirical Studies

Let's start with a simple question

Do you know what is inside an Android App?



AndroZoo for Large Scale Empirical Studies



AndroZoo for Large Scale Empirical Studies

We dissected 410 125 apks

How many files?

270 million files
661 files on average

How many file extensions (.dex,.jpg, .png)?

Over 15,000 file extensions

How many file types?

1000 file types

Other interesting facts

- Several apks embed another apk file
- 10% of apks contain compressed files

1

Large Set of Mobile Apps

2

Malware Detection

3

Mobile App Static Analysis

Malware Detection

Performance Assessment Issues

App Code Representation

Temporal-Incremental Learning

Malware Detection

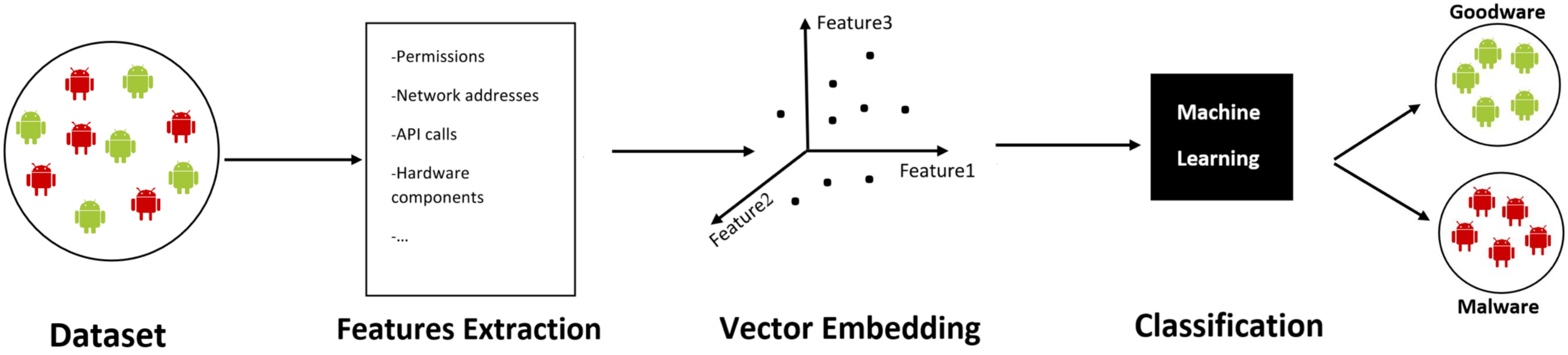
Performance Assessment Issues

App Code Representation

Temporal-Incremental Learning

On the difficulty of Assessing Machine- learning- based Android Malware Detection Approaches

Classical ML-based Android malware detection



Building Blocks of Machine Learning-based Android malware detection

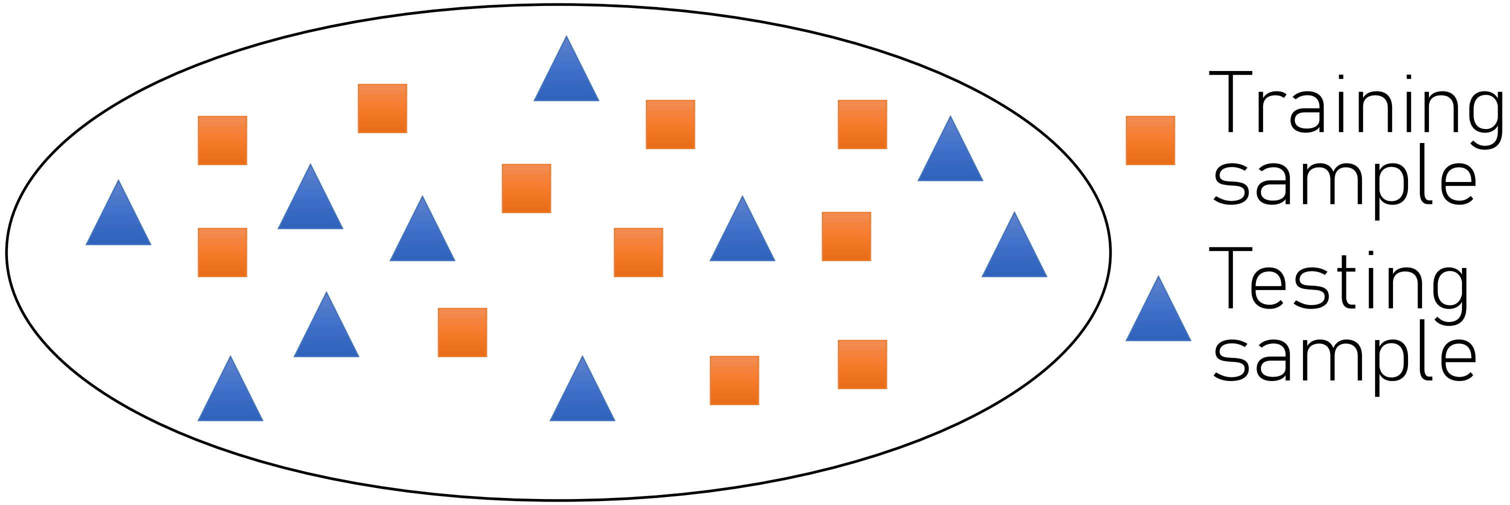
[EMSE2021] "Lessons learnt on reproducibility in machine learning based android malware detection"

Outstanding malware detection score of existing approaches

F1 score = 0.99

Machine Learning to detect Android Malware: main Outcomes

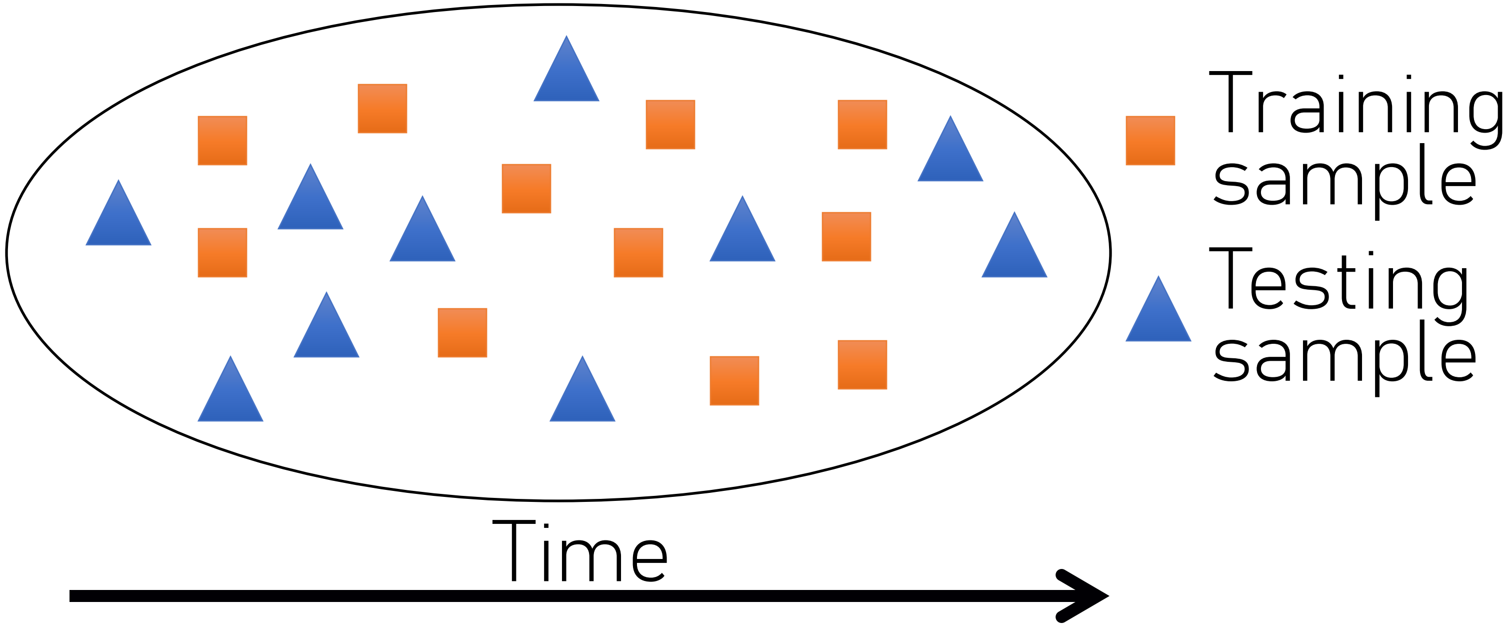
- Be careful about TIME! We don't know the future yet...



[1] Are Your Training Datasets Yet Relevant? - An Investigation into the Importance of Timeline in Machine Learning-Based Malware Detection

Machine Learning to detect Android Malware: main Outcomes

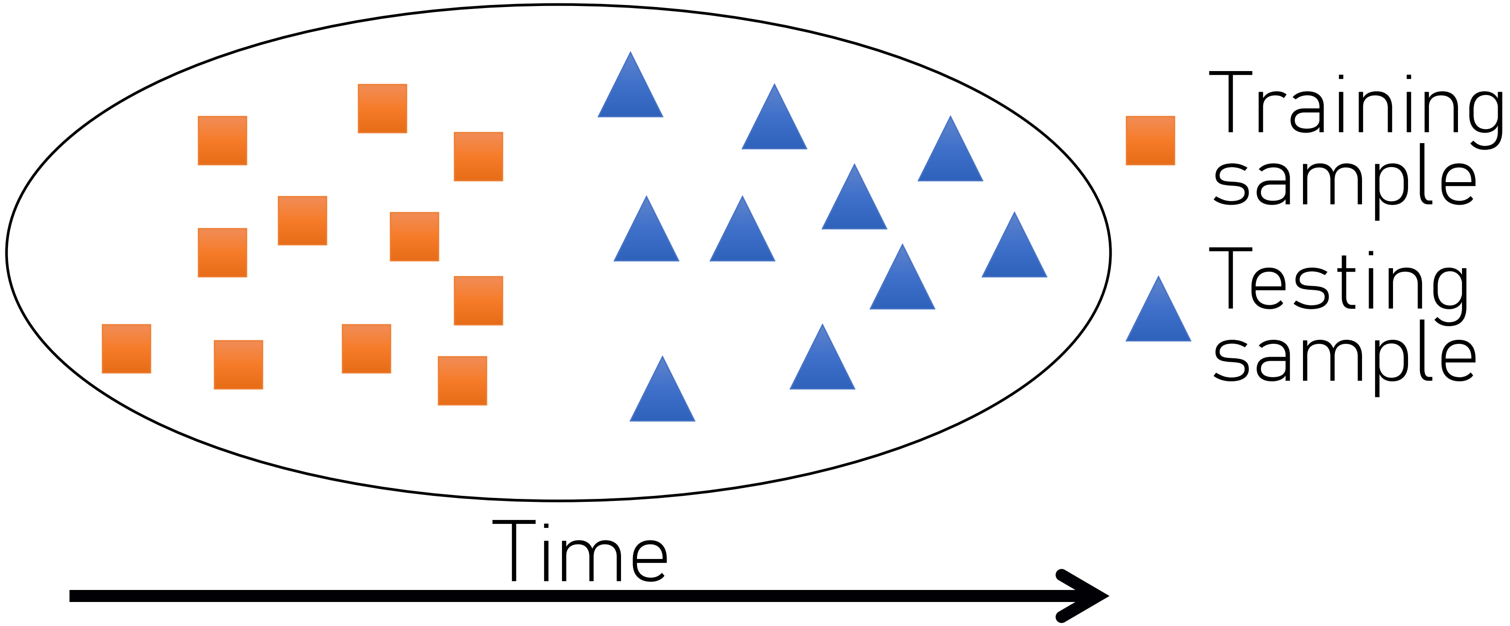
- Be careful about TIME! We don't know the future yet...



[1] Are Your Training Datasets Yet Relevant? - An Investigation into the Importance of Timeline in Machine Learning-Based Malware Detection

Machine Learning to detect Android Malware: main Outcomes

- Be careful about TIME! We don't know the future yet…



[1] Are Your Training Datasets Yet Relevant? - An Investigation into the Importance of Timeline in Machine Learning-Based Malware Detection

Machine Learning to detect Android Malware: main Outcomes

Ten-fold cross validation is not appropriated to assess machine learning-based malware detectors (paper at EMSE [2])

- Very good results "in the lab"
- Very poor results "in the wild"

[EMSE2014] Empirical Assessment of Machine Learning-Based Malware Detectors for Android: Measuring the Gap between In-the-Lab and In-the-Wild Validation Scenarios

Malware Detection

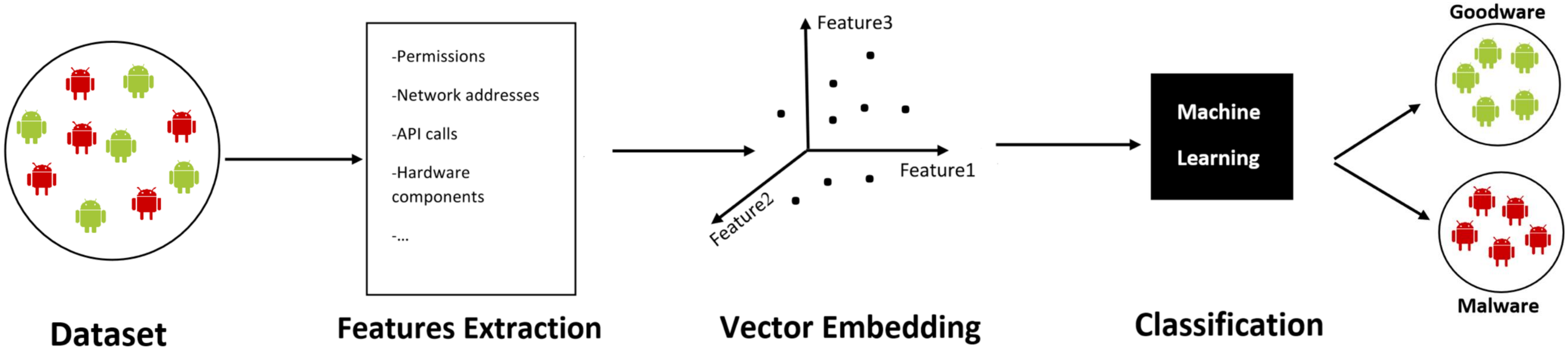
Performance Assessment Issues

App Code Representation

Temporal-Incremental Learning

App Code Representation

Classical ML-based Android malware detection

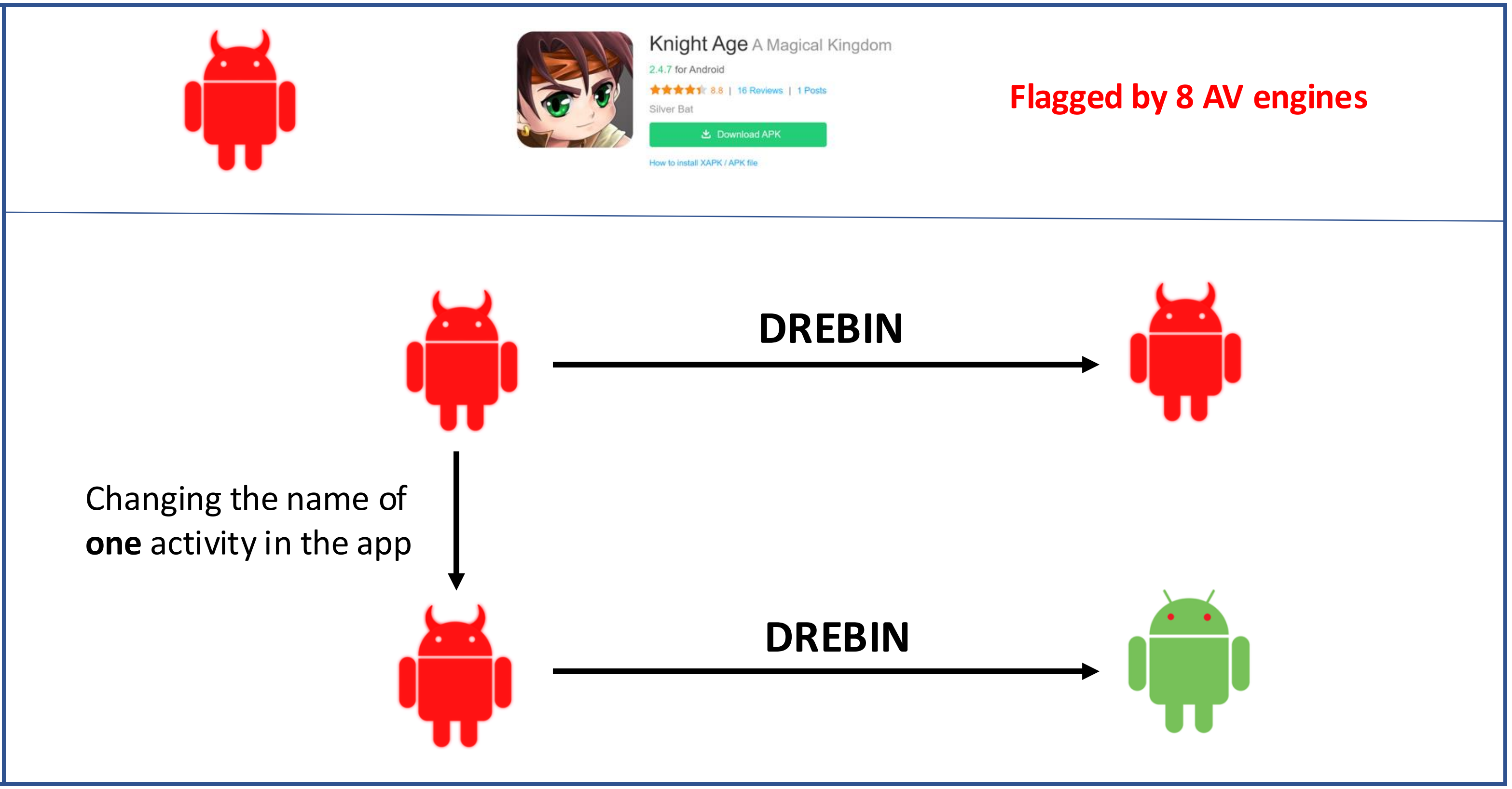


Building Blocks of Machine Learning-based Android malware detection

[EMSE2021] “Lessons learnt on reproducibility in machine learning based android malware detection”

Issues with Robustness: The discriminatory power of DREBIN’s features set

# of features	F1-score
1 230 854	0.98
1	**0.80**



Findings:

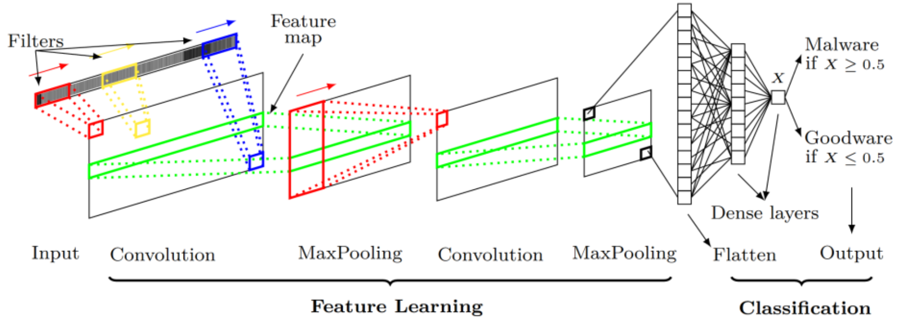
- A single feature can offer a surprisingly high detection rate.
- DREBIN’s most relevant features contain id-features.

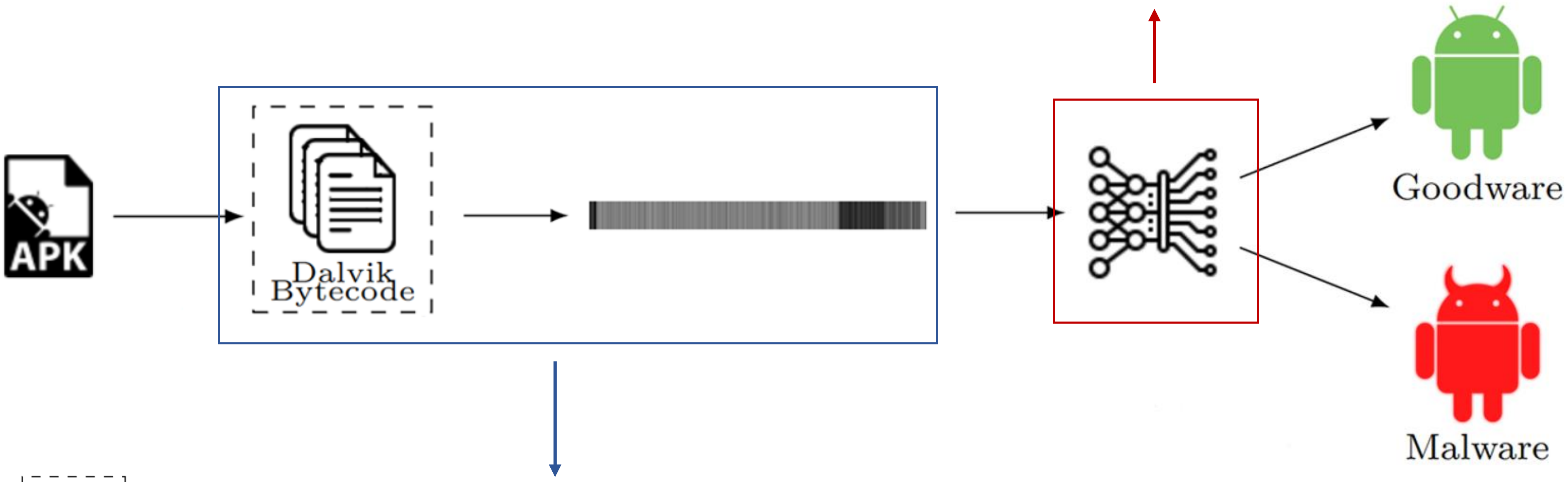
[TOPS2022] “A Deep Dive Inside DREBIN: An Explorative Analysis beyond Android Malware Detection Scores”

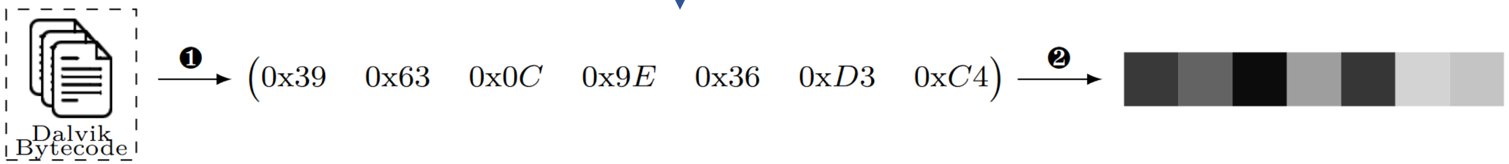
Let’s start simple

DexRay: An app as an Image

Approach







Process of image generation from dalvik bytecode. ❶: bytecode bytes' vectorisation; ❷: Mapping bytes to pixels

[MLHat2021] "DexRay: A Simple, yet Effective Deep Learning Approach to Android Malware Detection based on Image Representation of Bytecode"

Effectiveness of DexRay

Dataset and experimental setup

- 96 994 benign + 61 809 malware = 158 803 apps
- Apps with compilation dates from 2019 and 2020
- Dataset split: 80% training, 10% validation, and 10% test
- Experiments are repeated 10 times

Performance of DexRay against SotA malware detection approaches

	Accuracy	Precision	Recall	F1-score
DexRay	0.97	0.97	0.95	0.96
Drebin	0.97	0.97	0.94	0.96
R2-D2	0.97	0.96	0.97	0.97
Ding et al.-Model 1	0.94	-	0.93	-
Ding et al.-Model 2	0.95	-	0.94	-
DexRay (Temporally Consistent)	0.97	0.97	0.98	0.98

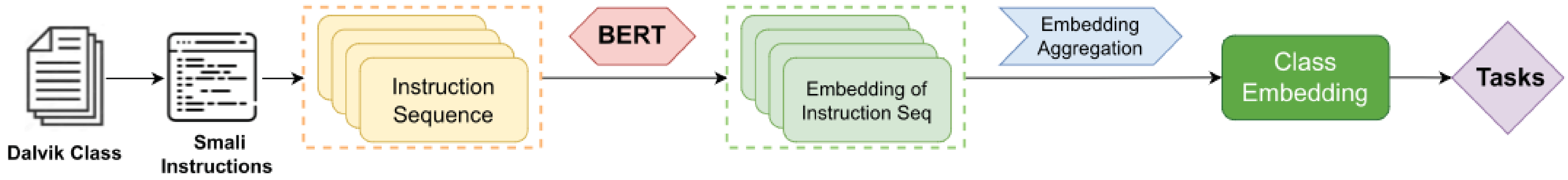
Findings:

- DexRay yields performance metrics that are comparable to the state of the art.
- Its simplicity has not hindered its performance when compared to similar works presenting sophisticated configurations.

A little bit better…

DexBERT: Class level Representation

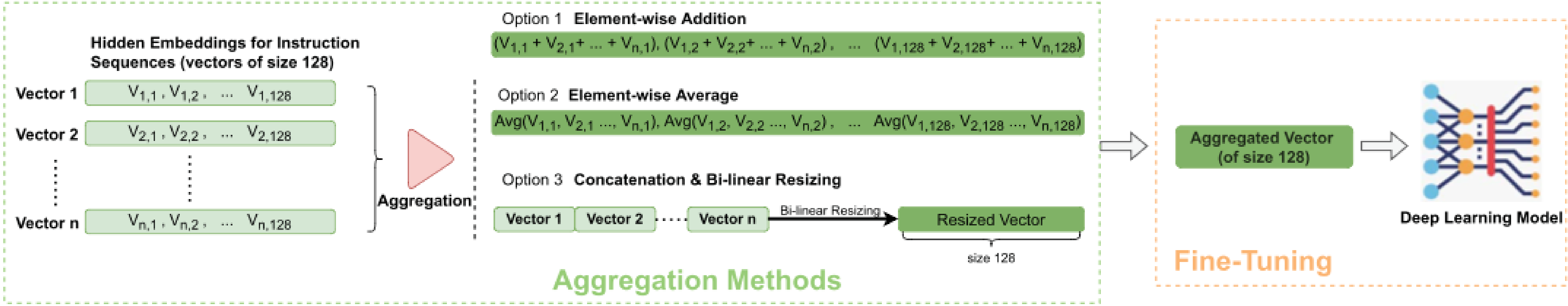
DexBERT: Effective, Task-Agnostic and Fine-Grained Representation Learning of Android Bytecode



DexBERT class embedding

[TSE2023] "DexBERT: Effective, Task-Agnostic and Fine-grained Representation Learning of Android Bytecode"

DexBERT: Effective, Task-Agnostic and Fine-Grained Representation Learning of Android Bytecode

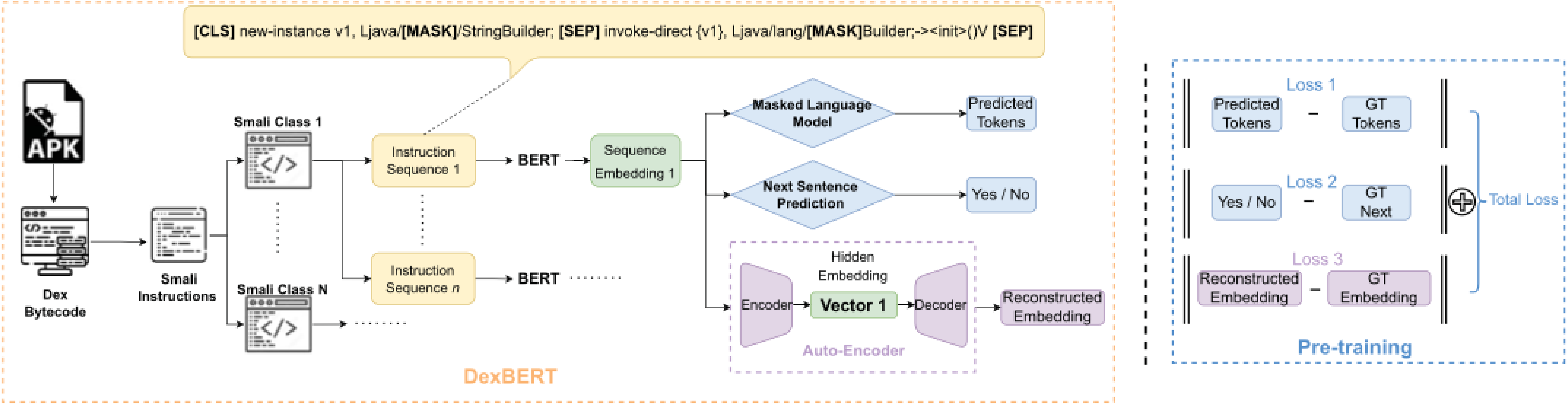


Three embedding aggregation methods and fine-tuning of downstream tasks.

(Addition is working the best)

DexBERT: Effective, Task-Agnostic and Fine-Grained Representation Learning of Android Bytecode

Pre-Training



Pre-training on 158 000 apps (556 millions tokens)

[TSE2023] “DexBERT: Effective, Task-Agnostic and Fine-grained Representation Learning of Android Bytecode”

DexBERT: Evaluation

Performance of Malicious Code localization on the MYST Dataset

Approach	F1 Score	Precision	Recall
MKLDroid	0.2488	0.1434	0.9400
smali2vec	0.9916	0.9880	0.9954
DexBERT-m	0.5749	0.4034	**1.0000**
DexBERT	**0.9981**	**0.9983**	0.9979

2000 apps for fine-tuning and 1000 for evaluation

DexBERT: Evaluation

Performance of Malicious Code localization on the MYST Dataset

Approach	F1 Score	Precision	Recall
MKLDroid	0.2488	0.1434	0.9400
smali2vec	0.9916	0.9880	0.9954
DexBERT-m	0.5749	0.4034	**1.0000**
DexBERT	**0.9981**	**0.9983**	0.9979

2000 apps for fine-tuning and 1000 for evaluation

Performance of Component Type Classification

Method	Activity	Service	BroadcastReceiver	ContentProvider	Average
BERT	0.8272	0.7642	0.5673	0.9091	0.7669
CodeBERT	0.917	0.5381	0.8756	0.8468	0.7943
DexBERT(woPT)	0.7402	0.5850	0.7660	0.8947	0.7465
DexBERT	**0.9780**	**0.9117**	**0.9600**	**0.9756**	**0.9563**

1000 real-world APKs (3406 components).

75% for training and 25% for testing.

[TSE2023] "DexBERT: Effective, Task-Agnostic and Fine-grained Representation Learning of Android Bytecode"

DexBERT: Evaluation

Performance of Malicious Code localization on the MYST Dataset

Approach	F1 Score	Precision	Recall
MKLDroid	0.2488	0.1434	0.9400
smali2vec	0.9916	0.9880	0.9954
DexBERT-m	0.5749	0.4034	**1.0000**
DexBERT	**0.9981**	**0.9983**	0.9979

2000 apps for fine-tuning and 1000 for evaluation

Performance of Component Type Classification

Method	Activity	Service	BroadcastReceiver	ContentProvider	Average
BERT	0.8272	0.7642	0.5673	0.9091	0.7669
CodeBERT	0.917	0.5381	0.8756	0.8468	0.7943
DexBERT(woPT)	0.7402	0.5850	0.7660	0.8947	0.7465
DexBERT	**0.9780**	**0.9117**	**0.9600**	**0.9756**	**0.9563**

1000 real-world APKs (3406 components).

75% for training and 25% for testing.

Performance of App Defect Detection

Project # of classes	AnkiDroid 14767	BankDroid 12372	BoardGame 1634	Chess 5005	ConnectBot 3865	Andlytics 5305	FBreader 9883	K9Mail 11857	Wikipedia 18883	Yaaic 974	Average Score	Weighted Average AUC Score
smali2vec	0.7914	0.7967	**0.8887**	0.8481	**0.9516**	0.834	0.8932	0.7655	**0.8922**	**0.9371**	0.8598	0.8399
DexBERT	**0.9572**	**0.9363**	0.7691	**0.9125**	0.8517	**0.9248**	**0.9378**	**0.8674**	0.8587	0.8764	**0.8892**	**0.9032**

92K smali classes labeled with Checkmarkx

[TSE2023] "DexBERT: Effective, Task-Agnostic and Fine-grained Representation Learning of Android Bytecode"

uni.lu | SnT

Full App-level Representation

DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware

DexBERT for Android class embedding

+

LaFiCMIL

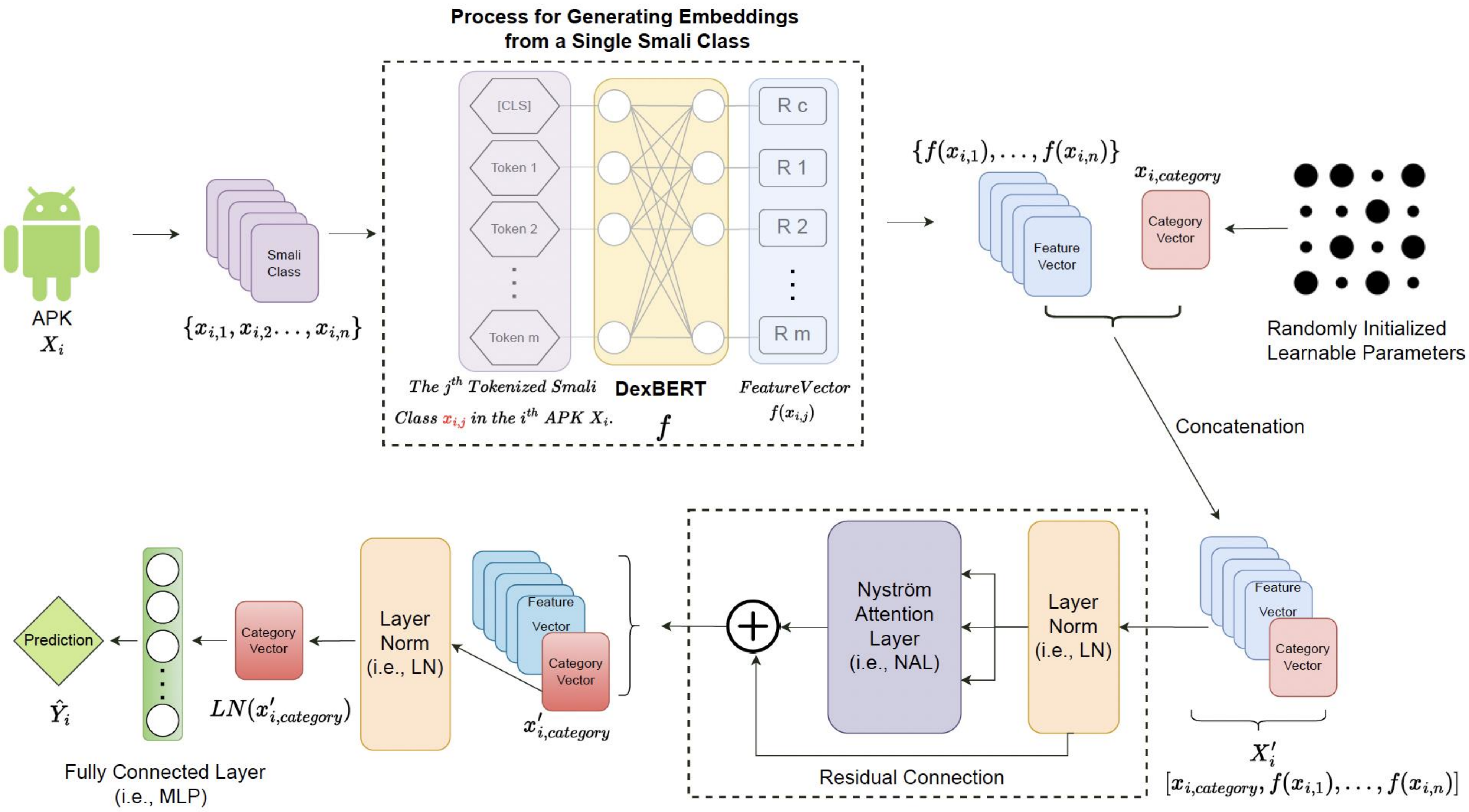
(Correlated Multiple Instance Learning)

[NLDB2024]: LaFiCMIL: Rethinking Large File Classification from the Perspective of Correlated Multiple Instance Learning

DetectBERT

[ESEM2024] "Towards Full App-Level Representation Learning to Detect Android Malware"

DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware



[ESEM2024] "Towards Full App-Level Representation Learning to Detect Android Malware"

DetectBERT: Evaluation

Table 2: Performance comparison with existing state-of-the-art approaches.

Model	Accuracy	Precision	Recall	F1 Score
Drebin	0.97	0.97	0.94	0.96
DexRay	0.97	0.97	0.95	0.96
DetectBERT	**0.97**	**0.98**	**0.95**	**0.97**

Table 3: Temporal consistency performance comparison with state-of-the-art approaches.

Model	Accuracy	Precision	Recall	F1 Score
Drebin	0.96	0.95	0.98	0.97
DexRay	0.97	0.97	0.98	0.98
DetectBERT	**0.99**	**0.99**	**0.99**	**0.99**

158 803 apks

(96 994 benign 61 809 malware)

80% training, 10% validation, 10% test

[ESEM2024] "Towards Full App-Level Representation Learning to Detect Android Malware"

Malware Detection

Performance Assessment Issues

App Code Representation

Temporal-Incremental Learning

SnT

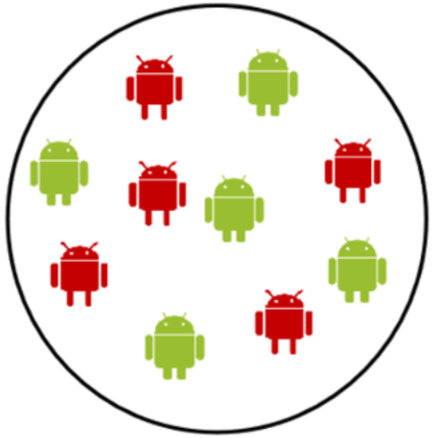
Temporal-incremental Learning for Android Malware Detection

Published at TOSEM in 2024

Presented at FSE 2025 by Tiezhu Sun

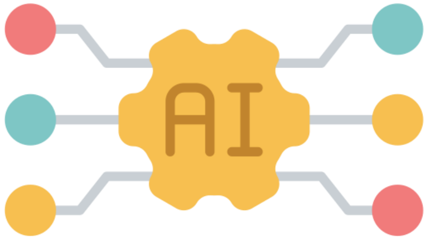
UNIVERSITY OF LUXEMBOURG

Android Malware Learning



Dataset

Data-driven and **automatically learned** features



Deep learning model

Classification

Detection

Localization

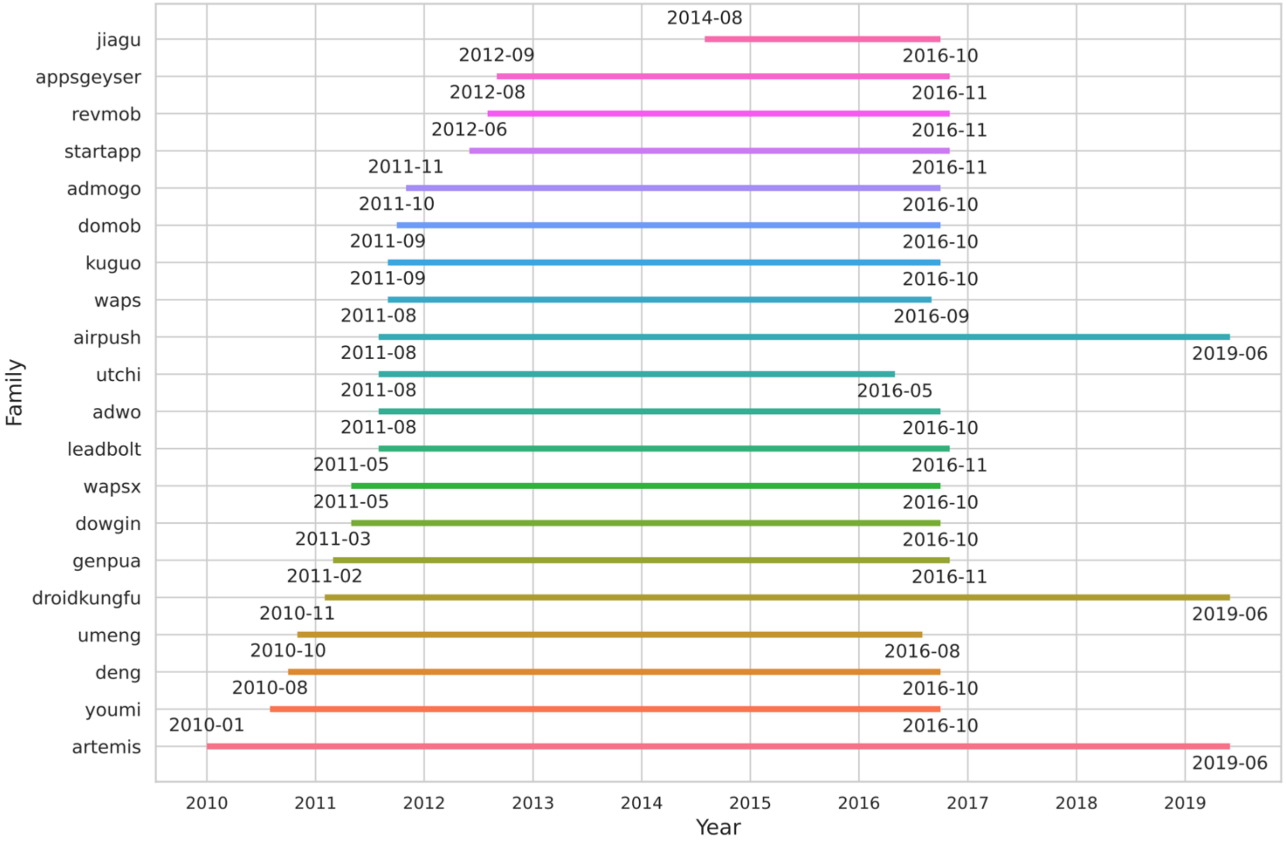
Android Malware Family Classification

Malicious Behaviors of Different Malware Families [1]

Family Name	Privacy Stealing	SMS/CALL	Remote Control	Bank Stealing	Ransom	Abusing Accessibility	Privilege Escalation	Stealthy Download	Ads	Miner	Tricky Behavior	Premium Service
RuMMS	√	√	√	√		○	○				√	
Xavier	√		√					√	√			
LIBSKIN	√	√	√				√	√	√			
HiddenAd	√								√		√	
GhostClicker	√		√				√		√			
MilkyDoor	√		√									
EventBot	√	√	√	√		√						
GhostCtrl	√	√	√		○							
Lucy			√		√	√	√	√			√	
FAKEBANK	√	√	√	√			√				√	
FakeSpy	√	√	√	√								
Joker	√	√	√					○	○			√
SpyNote	√	√	√								√	
solid	√								√		√	
ZNIU	√	√	√				√					

[1] Wang et al. 2022. MalRadar: MalRadar: Demystifying android malware in the new era. Proceedings of the ACM on Measurement and Analysis of Computing Systems

Motivation



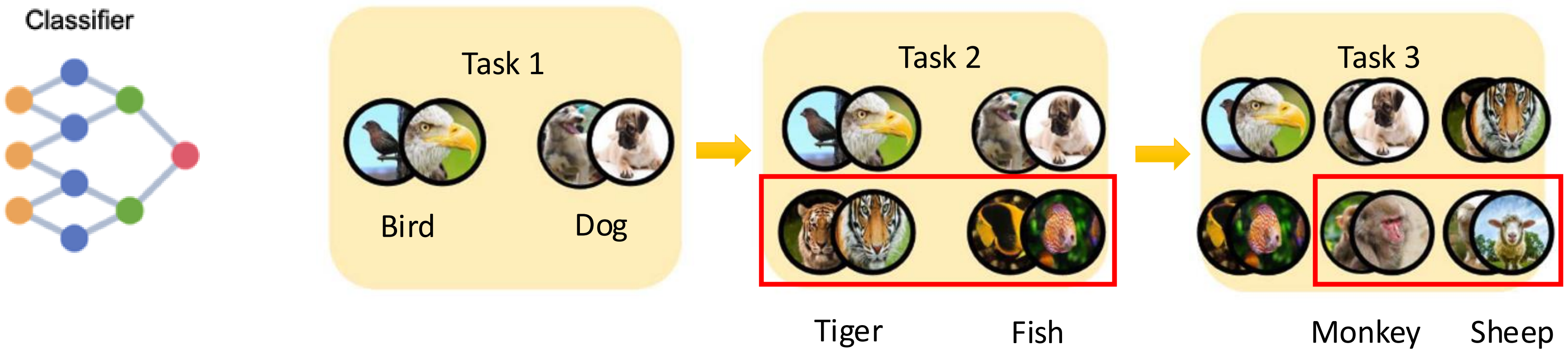
Life Span of 20 Malware Families

Concept Drift

1. The emergence of new malware families.

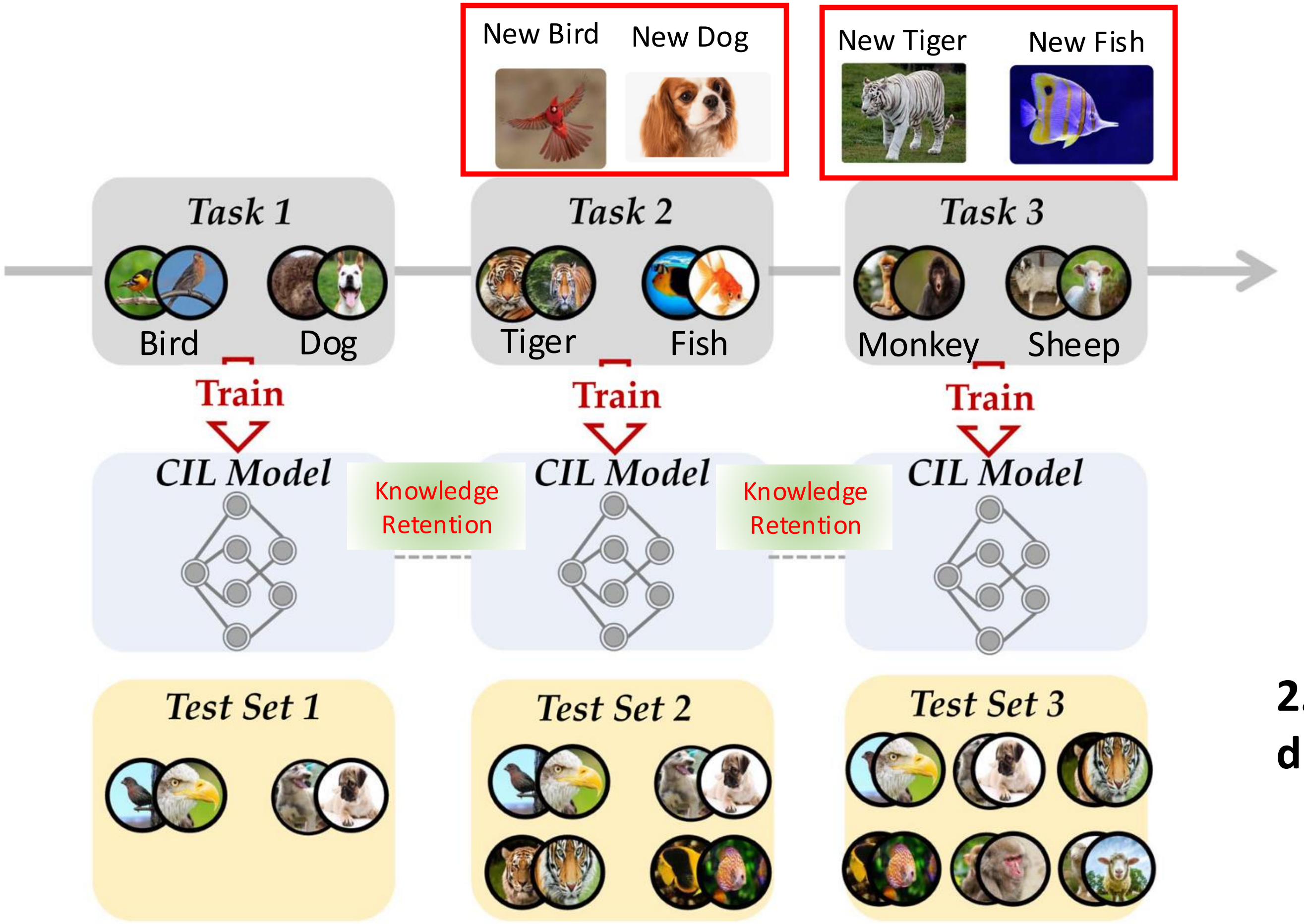
2. The shifts in the data distribution of old families.

Traditional Solution: Full Retraining



Drawbacks:
- Increasing resource demands for training time and data storage.
- Historical data might be unavailable due to privacy protection policies or security concerns.

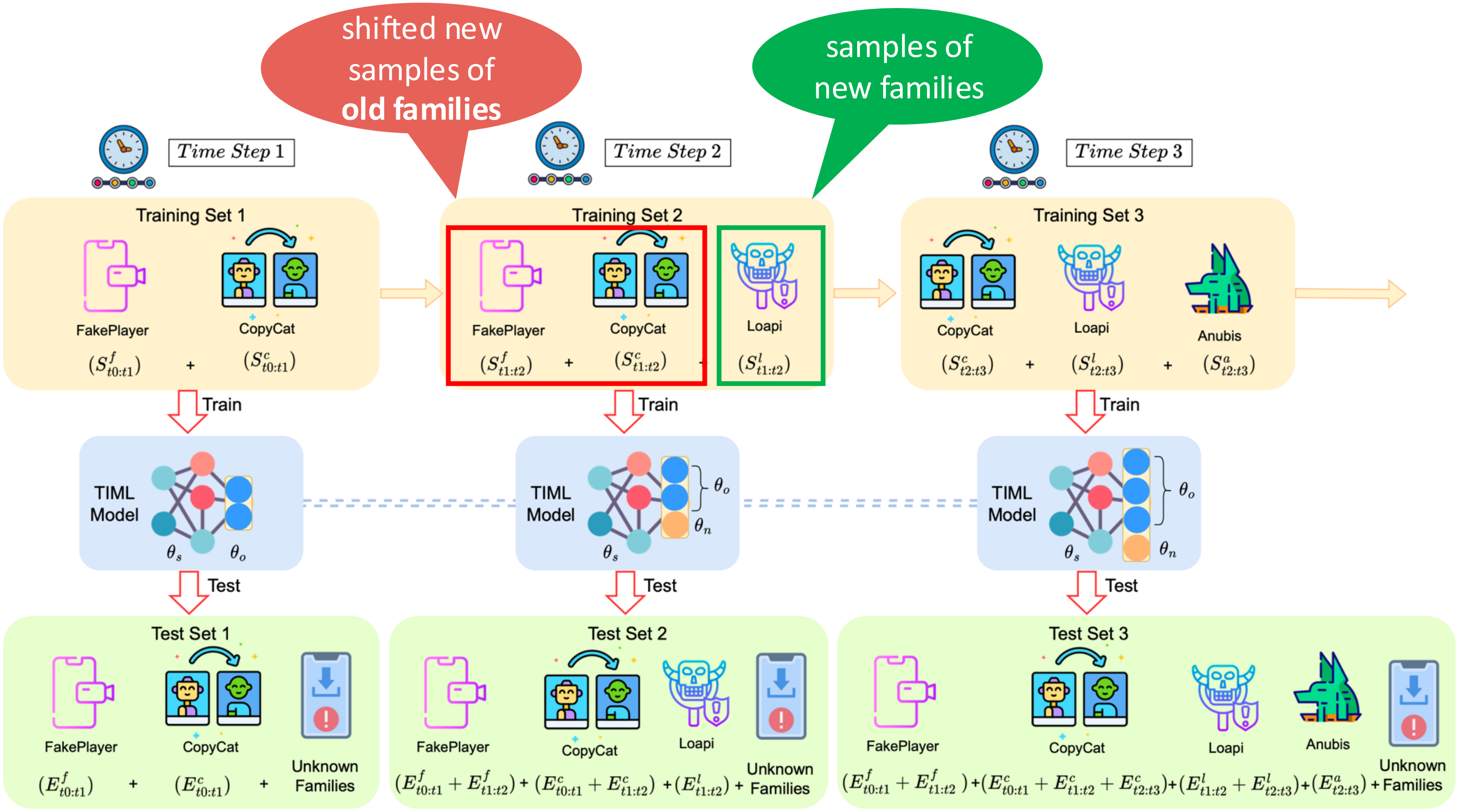
CIL: Class-Incremental Learning



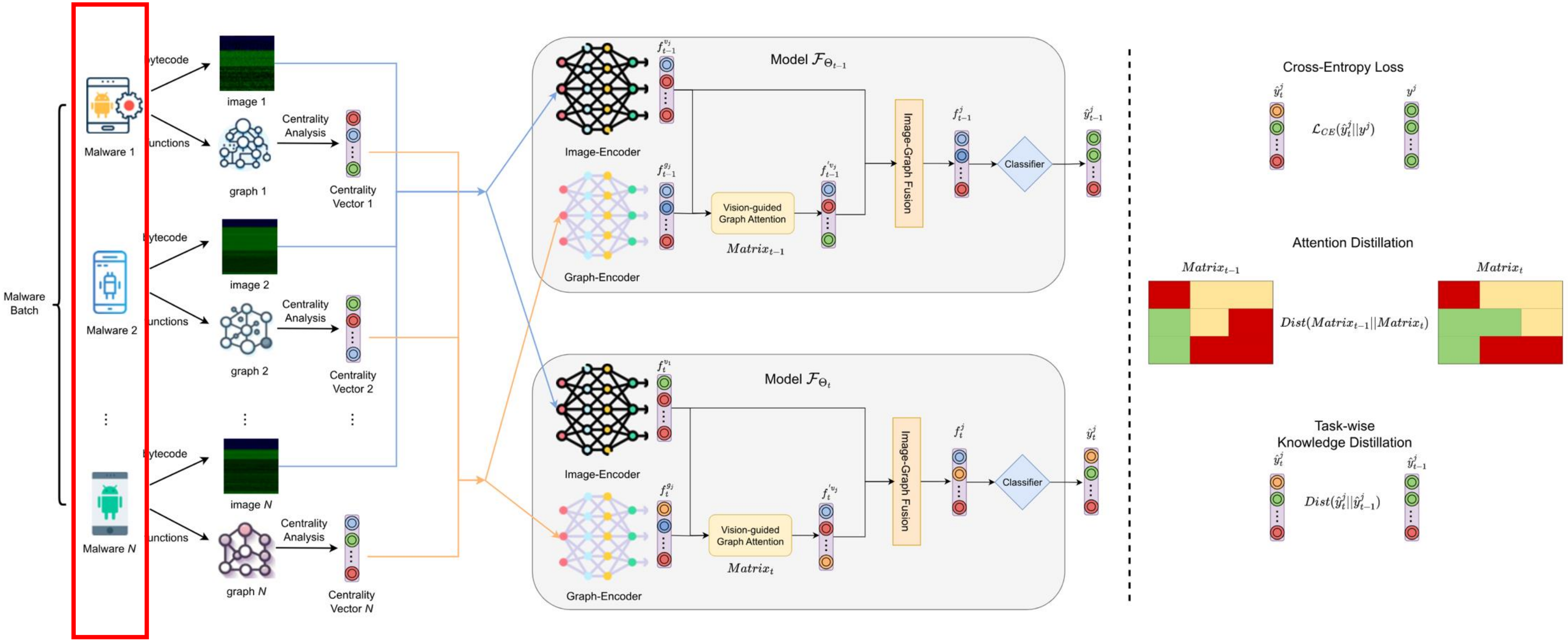
1. The emergence of new classes.

2. The shifts in the data distribution of old classes.

TIML: Temporal-Incremental Malware Learning



Multimodal TIML



Dataset

- **Size**: 1.2 million malware samples, categorized into 696 malware families, sourced from MalNet [1].
- **Time Span**: Covers a decade of malware evolution, with the “first-seen” timestamp obtained from AndroZoo [2].
- **Organization**: Samples are carefully organized in chronological order based on their emergence.

[1] S., Freitas, "MalNet: A Large-Scale Image Database of Malicious Software,” CIKM 2022.
[2] K., Allix, “Androzoo: Collecting millions of android apps for the research community”, MSR 2016.

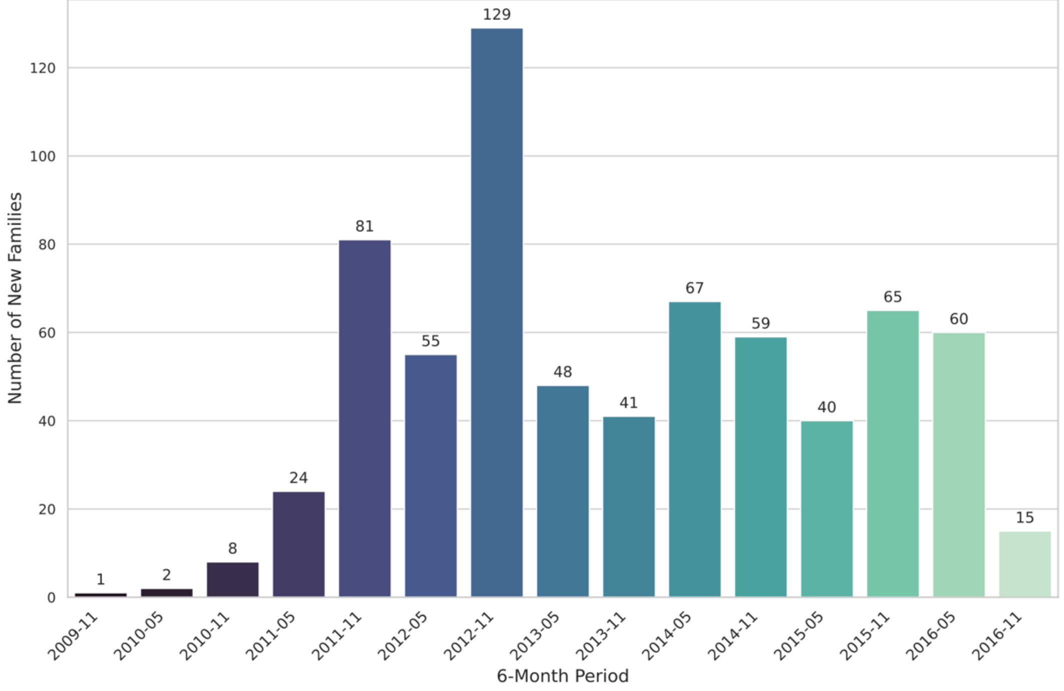
Preliminary Study: CIL vs TIML

Table 4.1 Accuracy comparison between adapted TIML approaches and their original CIL counterparts.

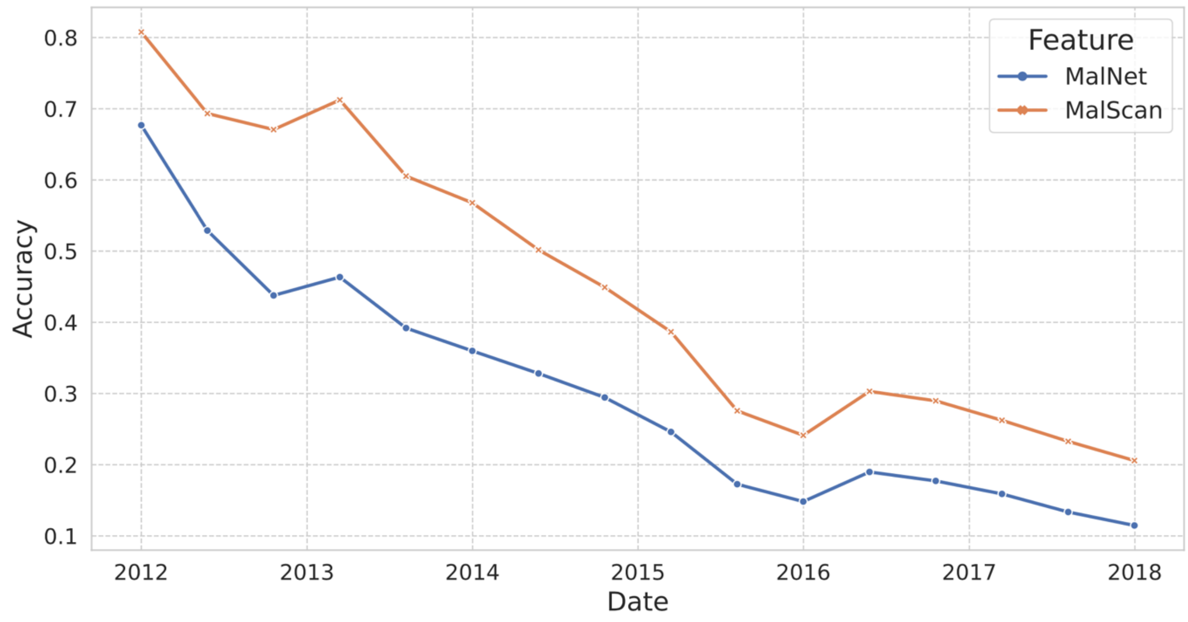
Method	Adapted TIML Accuracy	CIL Accuracy
LwF	49.68%	27.99%
iCaRL	58.15%	23.13%
SS-IL	54.58%	21.65%

Findings:
TIML methods demonstrate significant accuracy improvements.

RQ1: Is concept drift a significant factor affecting malware classification?



Distribution of new malware families – per 6-months time steps.



Performance drop curve of models trained on pre-2012 malware families and evaluated on post-2012 samples from the same families.

<u>Findings</u>:

- The two types of concept drift do exist.
- Concept drift degrade the performance of malware classifiers.

RQ2: How well do TIML approaches perform in malware classification?

MalNet Feature

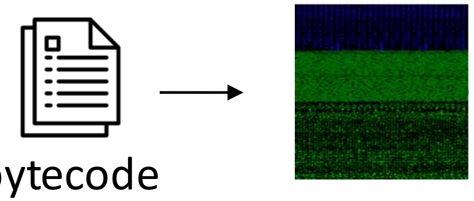
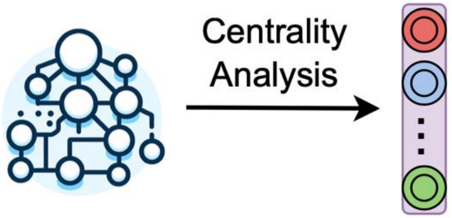


Table 4.2 Performance comparison of different approaches based on two input features: MalNet and MalScan.

Approach	Mean Accuracy (%)		Average Forgetting	
	MalNet	MalScan	MalNet	MalScan
Random Prediction	1.72	1.72	-	-
Fine-tuning *	51.63	64.66	13.25	18.59
LwF	52.82	65.69	12.48	18.09
SS-IL	51.73	67.14	8.24	8.73
iCaRL	53.57	68.57	8.79	8.57
LwF with Exemplars	56.28	69.74	8.07	8.13
Full Retraining *	63.23	75.08	-	-
MM-TIML	70.53		7.66	

MalScan Feature



Findings:

- TIML approaches achieve competitive performance compared to full retraining.
- The slight gap is due to TIML's limited access to historical data.

RQ3: How resilient are TIML approaches to catastrophic forgetting?

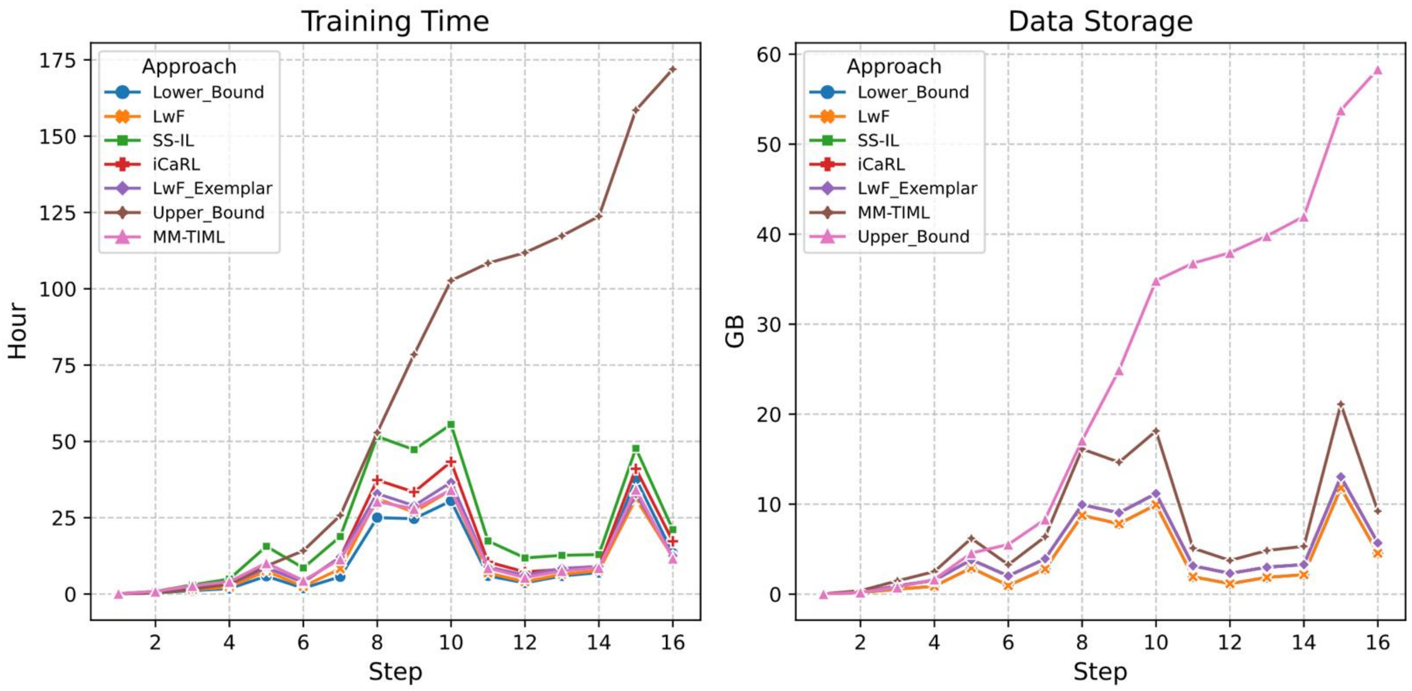
Table 4.2 Performance comparison of different approaches based on two input features: MalNet and MalScan.

	Mean Accuracy (%)		Average Forgetting	
Approach	MalNet	MalScan	MalNet	MalScan
Random Prediction	1.72	1.72	-	-
Fine-tuning *	51.63	64.66	13.25	18.59
LwF	52.82	65.69	12.48	18.09
SS-IL	51.73	67.14	8.24	8.73
iCaRL	53.57	68.57	8.79	8.57
LwF with Exemplars	56.28	69.74	8.07	8.13
Full Retraining *	63.23	75.08	-	-
MM-TIML	70.53		7.66	

Findings:

- TIML approaches exhibit signs of forgetting.
- MM-TIML demonstrates the strongest retention of previous knowledge.

RQ4: How effectively do TIML approaches optimize resource utilization?



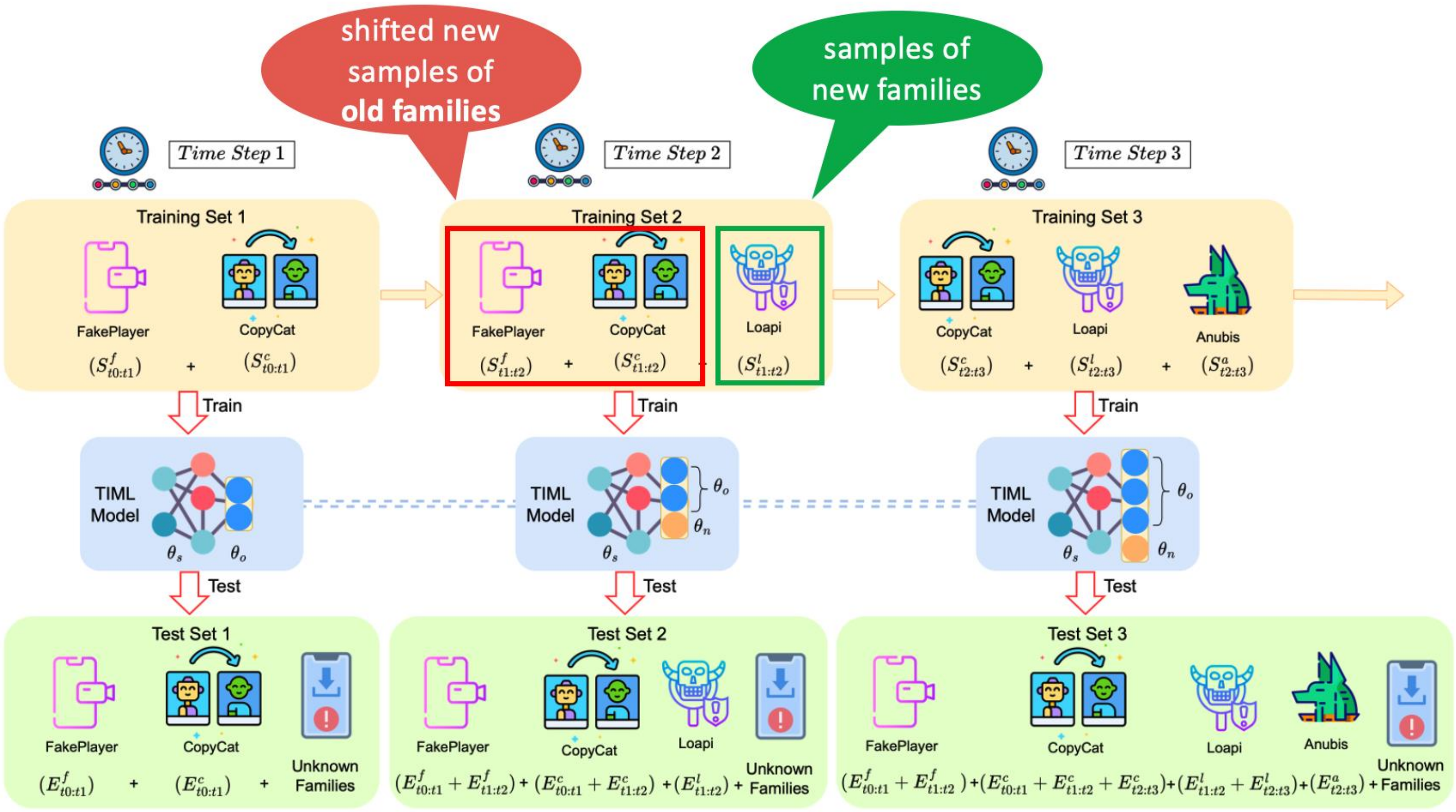
Training time and data storage comparison of different approaches, based on MalNet.

Findings:

- TIML approaches significantly reduce training time and data storage requirements.
- The advantage of TIML becomes more pronounced with increasing model updates.

Summary

TIML: Temporal-Incremental Malware Learning



Today: Android Malware & Dynamic Analysis

LLMs? | Malicious code localization

Malware Family Characterization | Ground Truth Creation LLMs?

LLMs? | Code Obfuscation

App Instrumentation | ACV Tool AndroLog

Code Coverage – Logic Bombs

1

Large Set of Mobile Apps

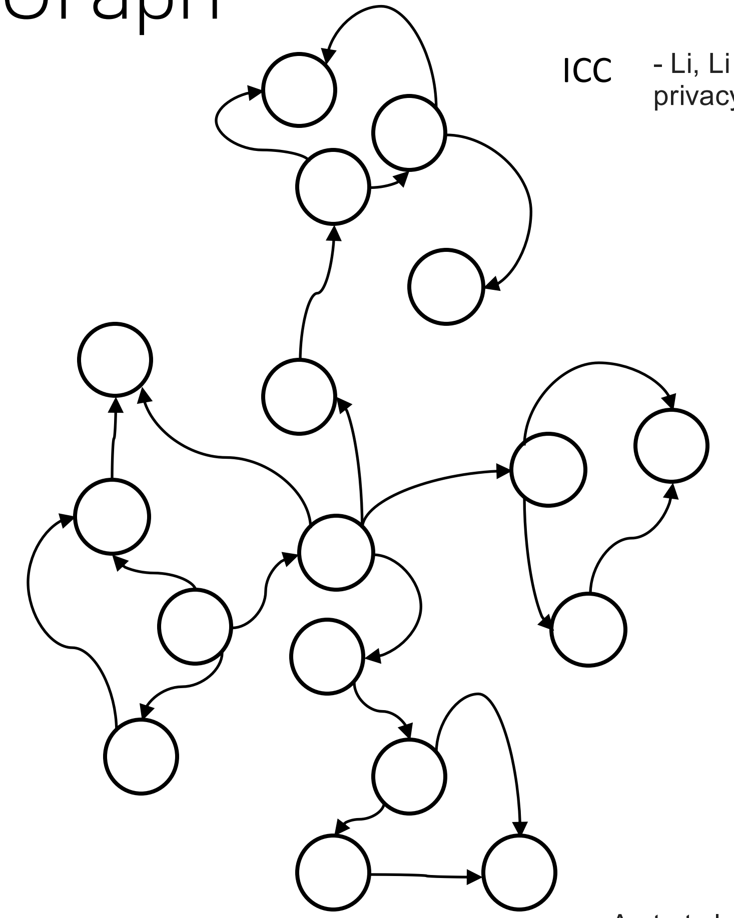
2

Malware Detection

3

Mobile App Analysis

Let's come back to Call Graph



ICC - Li, Li et al. Iccta: Detecting inter-component privacy leaks in android apps. ICSE 2015.

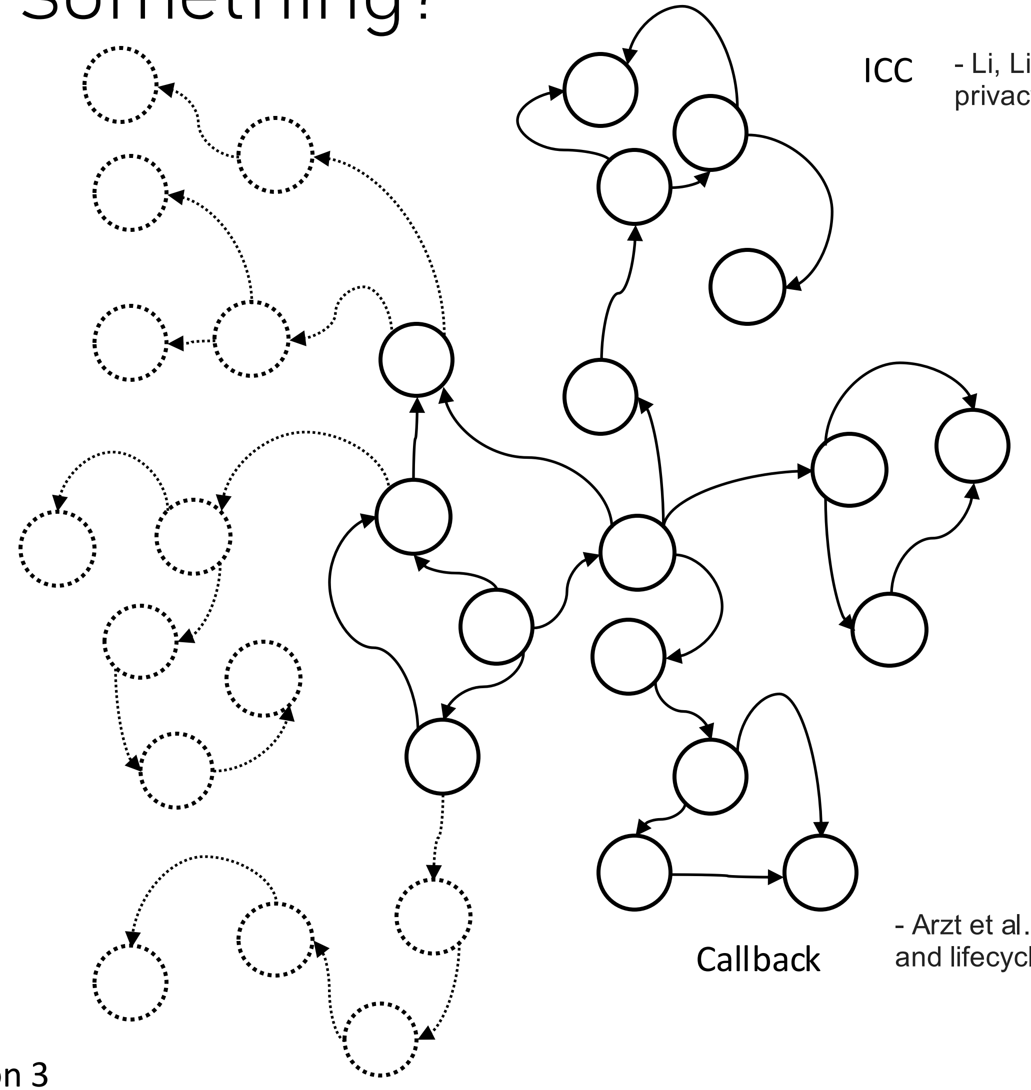
Reflection

- Li, Li et al., Droidra: Taming reflection to support whole-program analysis of android apps. ISSTA 2016.

Callback

- Arzt et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. PLDI 2014.

Are we missing something?



Contribution 1

Contribution 2

Contribution 3

ICC - Li, Li et al. Iccta: Detecting inter-component privacy leaks in android apps. ICSE 2015.

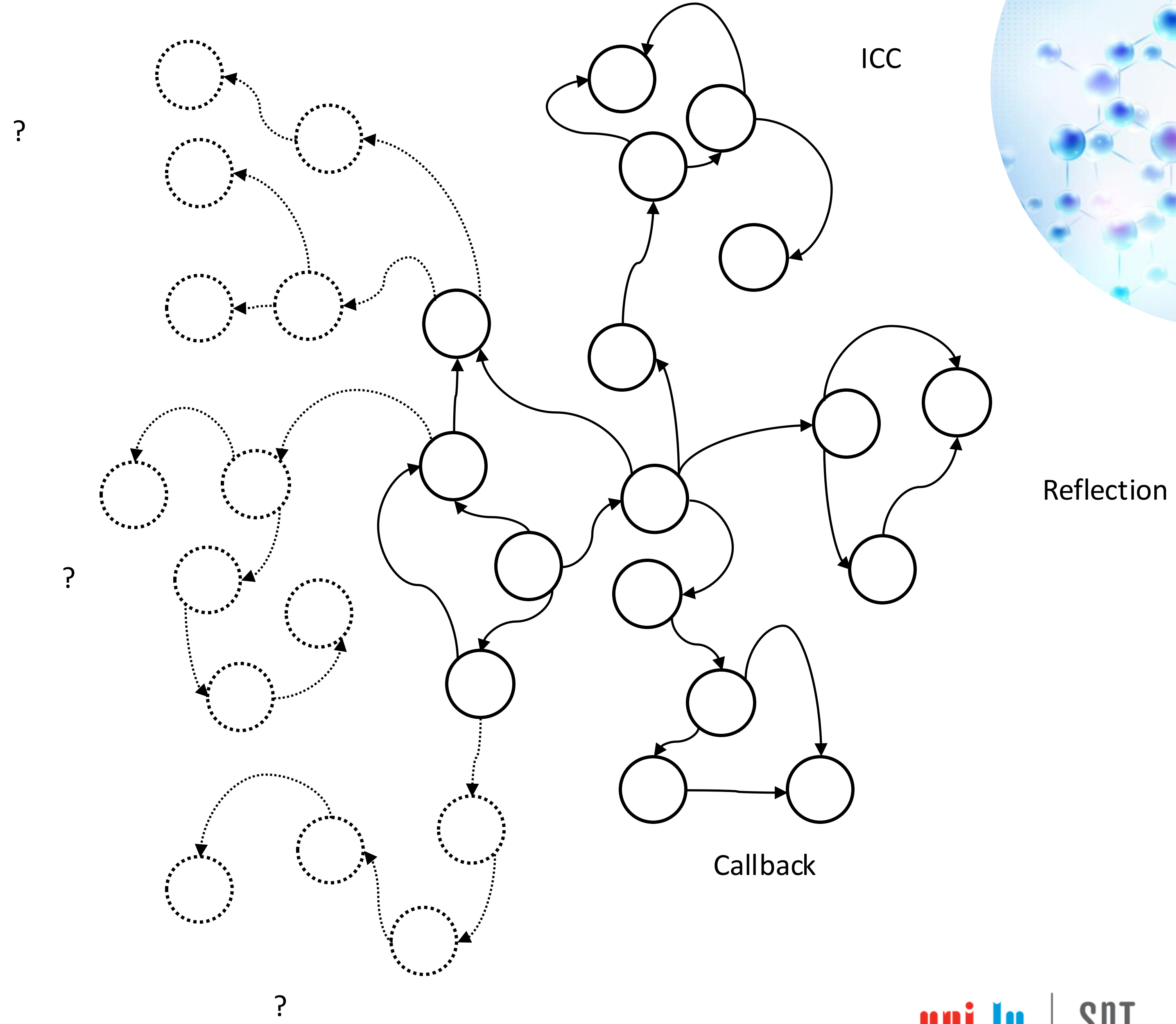
Reflection

- Li, Li et al., Droidra: Taming reflection to support whole-program analysis of android apps. ISSTA 2016.
.

Callback

- Arzt et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. PLDI 2014.

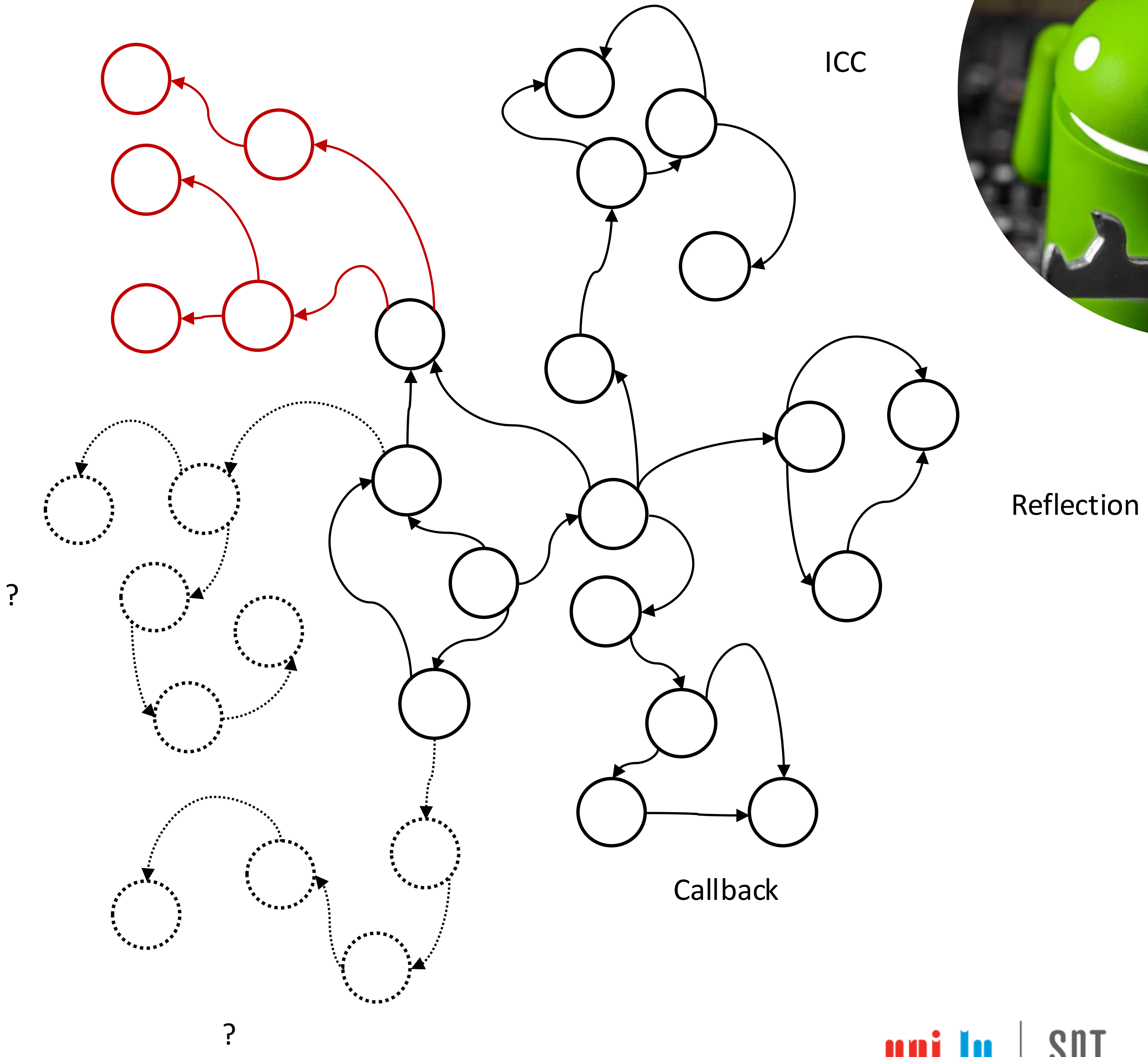
“Opportunistic” discoveries….



Contribution 1:
J. Samhi et al., "RAICC: Revealing Atypical Inter-Component Communication in Android apps", ICSE 2021.

- RAICC improves ICC modeling
- It is is already used by collaborators
- It is maintained
- Improvable on-demand
- RAICC and artifacts are available at:

https://github.com/JordanSamhi/RAICC

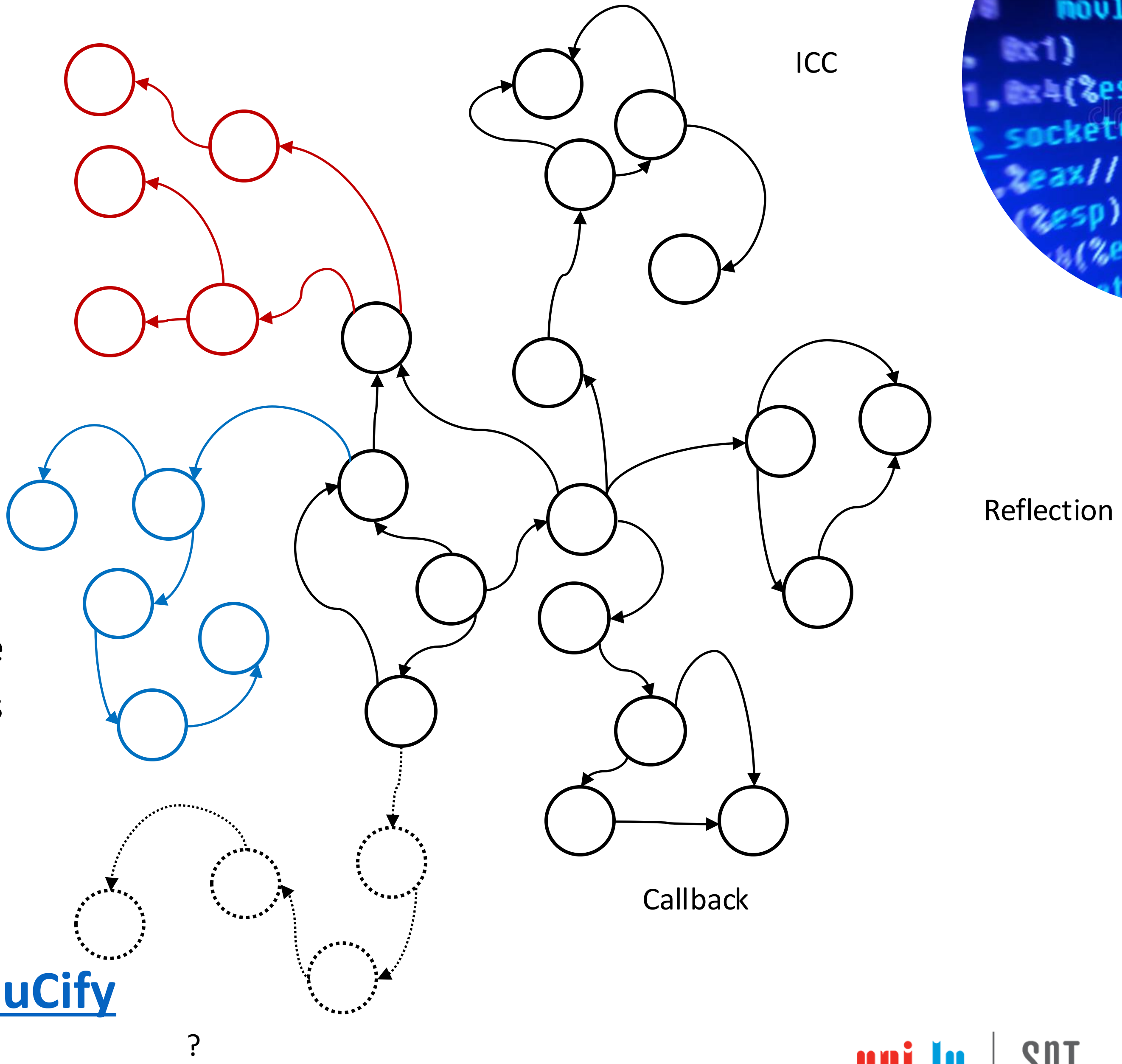


Contribution 1:
J. Samhi et al., “RAICC: Revealing Atypical Inter-Component Communication in Android apps”, ICSE 2021.

Contribution 2:
J. Samhi et al., “JuCify: A Step Towards Android Code Unification for Enhanced Static Analysis”, ICSE 2022.

- We proposed a new approach to unify the bytecode and native code representations
- We demonstrated how JuCify is a step toward code unification
- JuCify and artifacts are available at:

https://github.com/JordanSamhi/JuCify

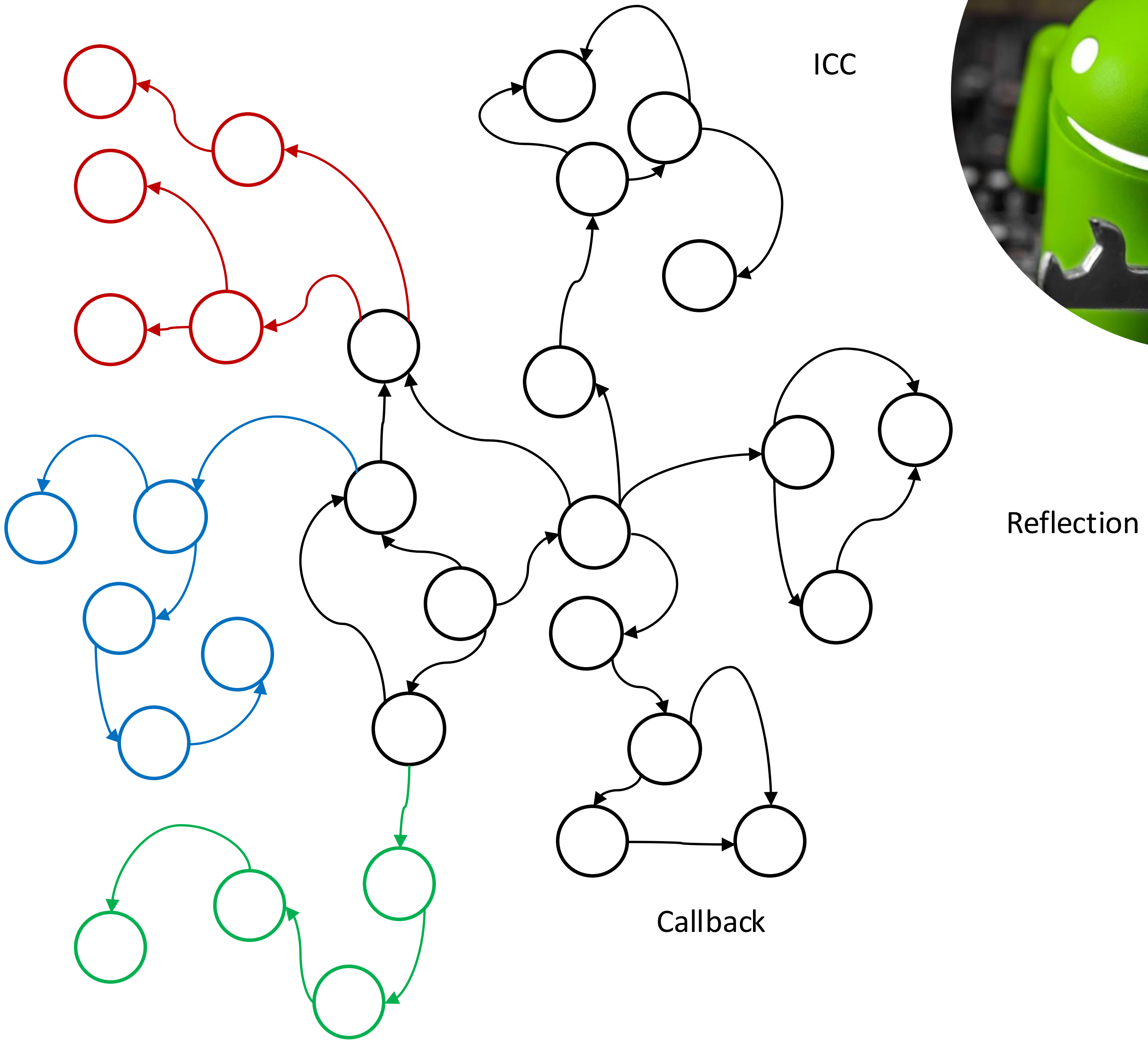


Contribution 1:
J. Samhi et al., "RAICC: Revealing Atypical Inter-Component Communication in Android apps", ICSE 2021.

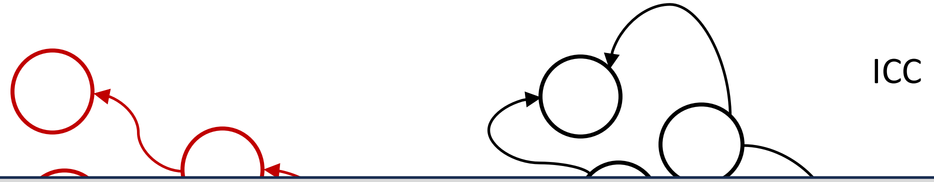
Contribution 2:
J. Samhi et al., "JuCify: A Step Towards Android Code Unification for Enhanced Static Analysis", ICSE 2022.

Contribution 3:
J. Samhi et al., "Resolving Conditional Implicit Calls to Improve Static and Dynamic Analysis in Android apps", TOSEM 2025

- We proposed a new approach for Conditional Implicit Calls
- We demonstrated how Archer improves static analysis
- We demonstrated how Archer aids dynamic analysis



Contribution 1:
J. Samhi et al., “RAICC: Revealing Atypical
Inter-Component C
apps”, ICSE 202



Contribution
J. Samhi et al.
Code Unificat
ICSE 2022.

tion

Contribution
J. Samhi et al.
Calls to Impro
Android apps

Is our call graph comprehensive/complete now?

Or are we still missing something?

- We propo
Implicit Ca
- We demor
analysis
- We demonstrated how Archer aids dynamic
analysis

Contribution 1:
J. Samhi et al., “RAICC: Revealing Atypical Inter-Component C
apps”, ICSE 202

Contribution
J. Samhi et al.
Code Unificat
ICSE 2022.

tion

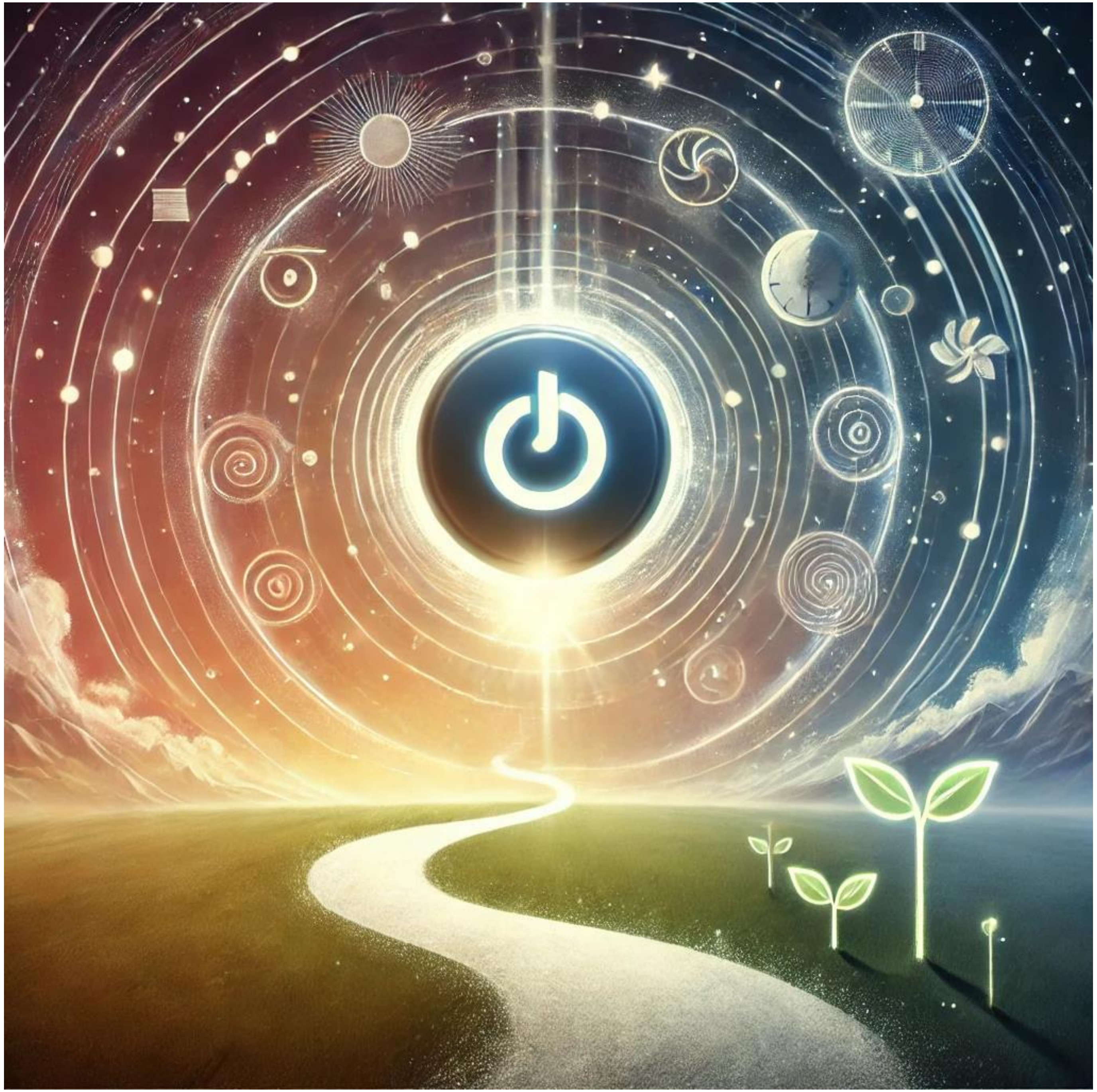
Contribution
J. Samhi et al.
Calls to Impro
Android apps

- We propo
Implicit Ca
- We demon
analysis
- We demonstrated how Archer aids dynamic analysis

Is our call graph comprehensive/complete now?

Or are we still missing something?

ISSTA24: *Call Graph Soundness in Android Static Analysis, Jordan Samhi, René Just, Tegawendé F. Bissyandé, Michael D. Ernst, Jacques Klein*



Let's restart from the beginning

Two main techniques to analyse a program

1

Dynamic Analysis

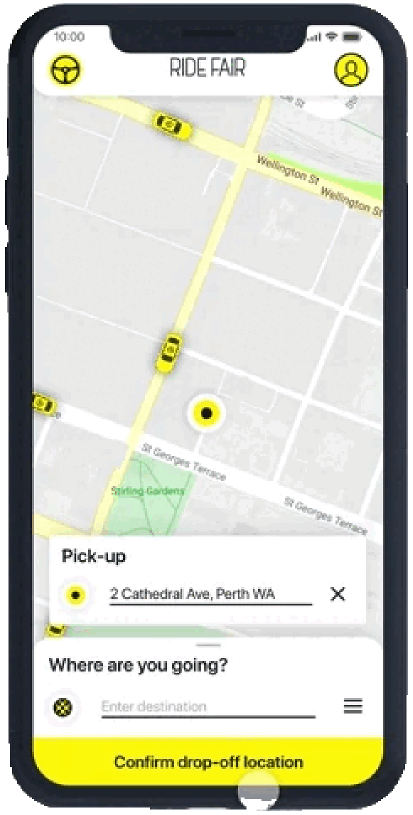
2

Static Analysis

Objective

Measure and understand the level of **unsoundness** in Android static analysis tools

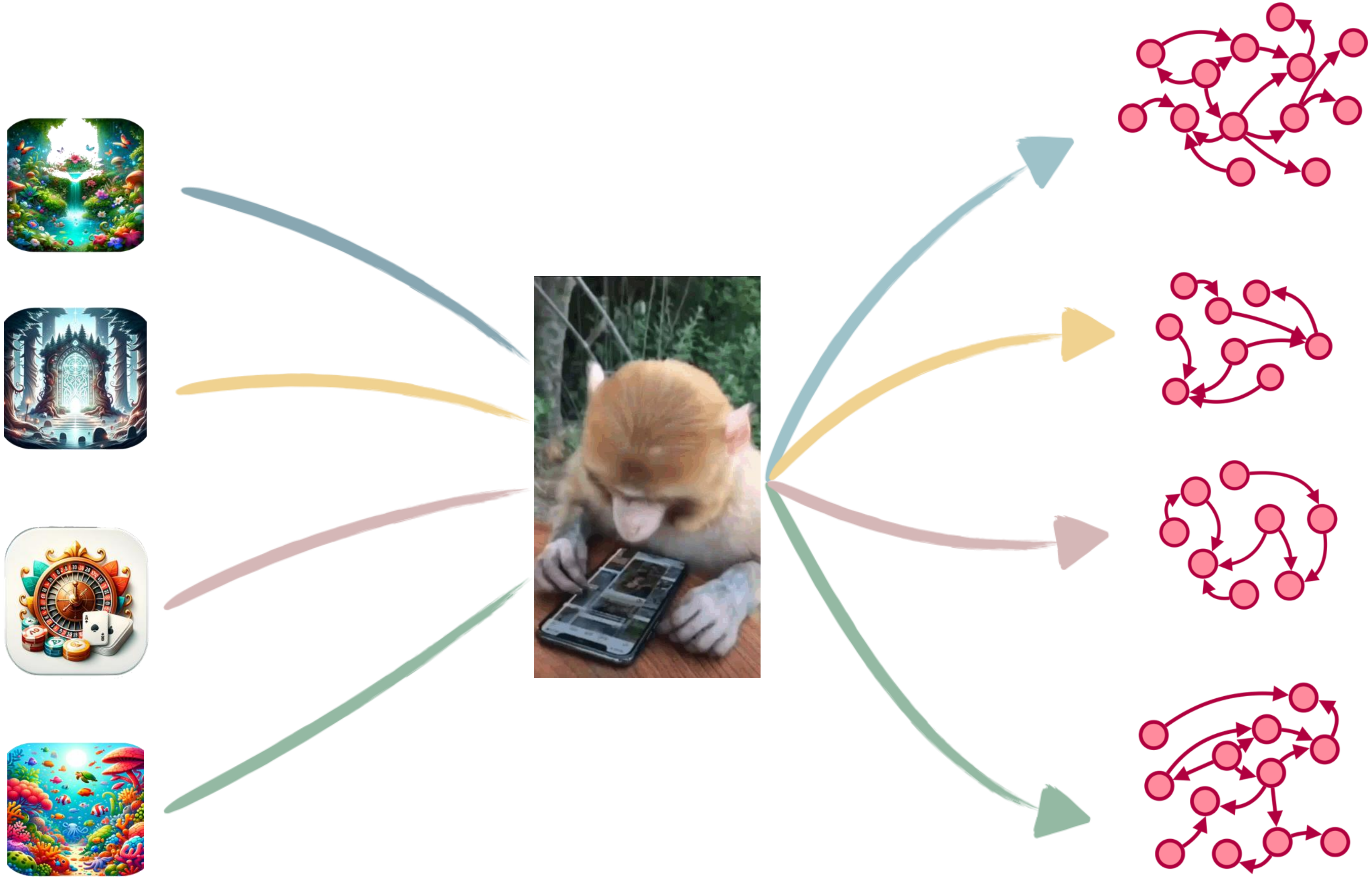
How?

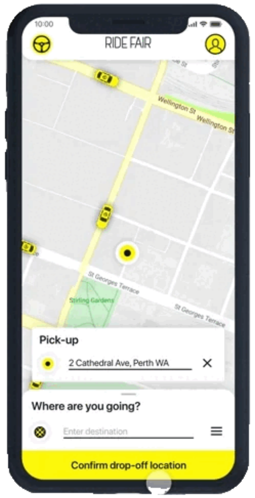


Dynamic Analysis

Static Analysis

Dynamic Analysis

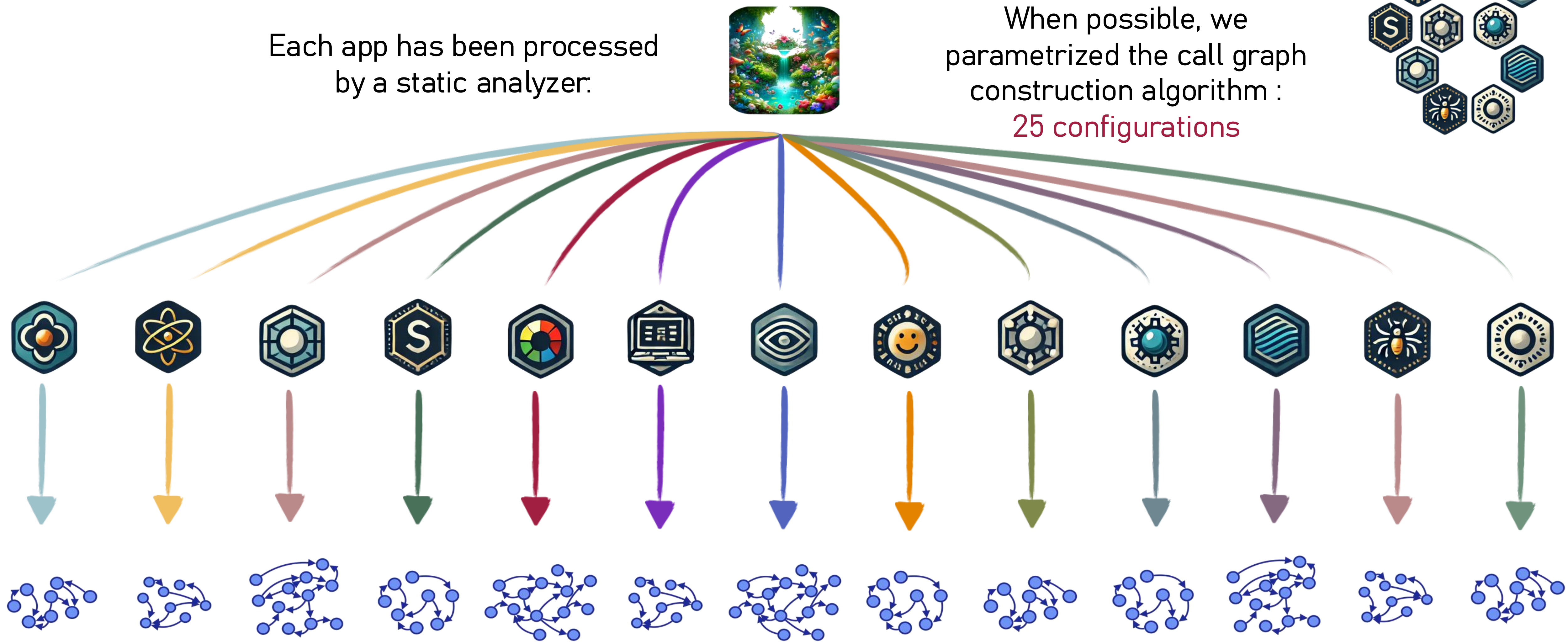




Static Analysis

Each app has been processed by a static analyzer.

When possible, we parametrized the call graph construction algorithm : 25 configurations

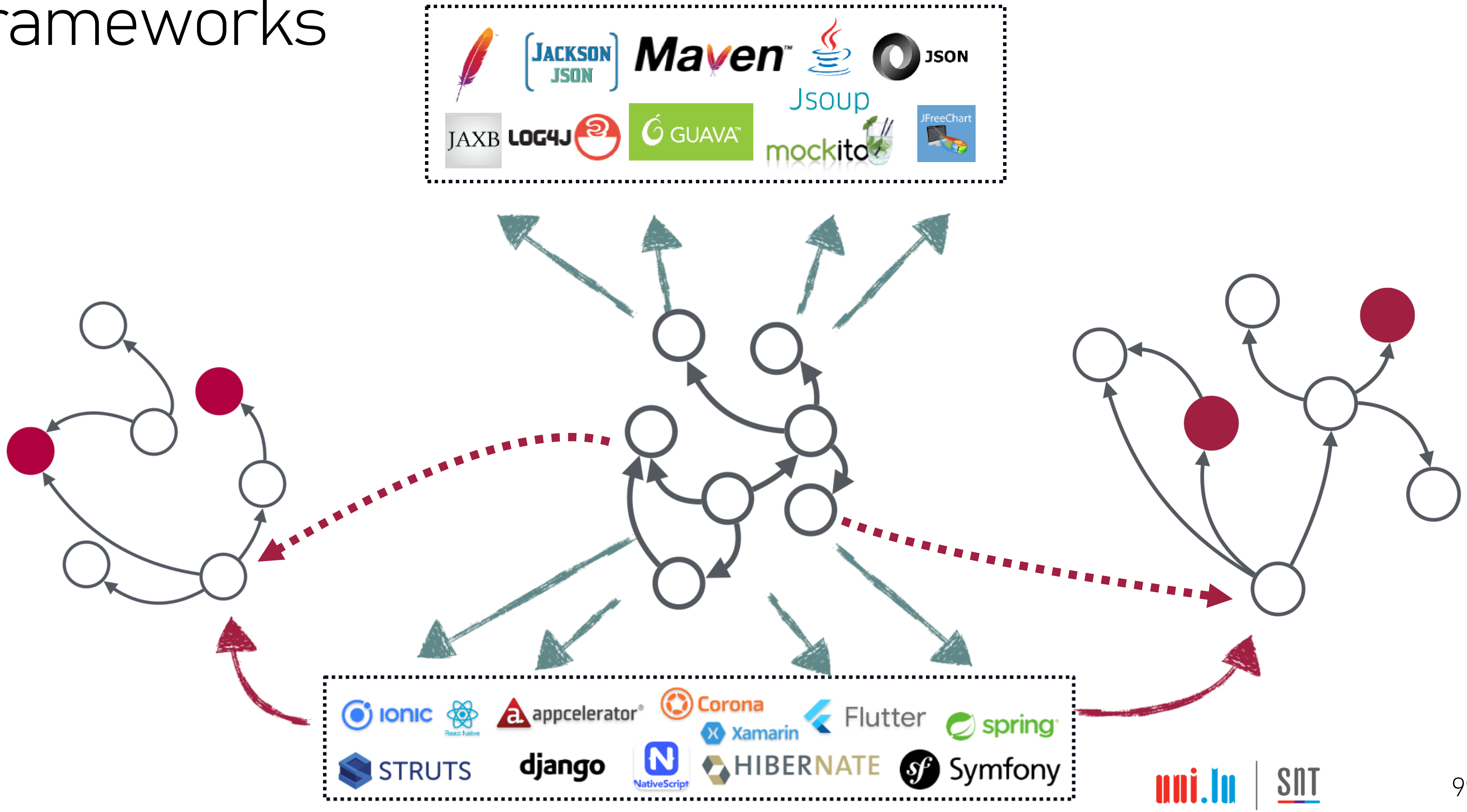


40%
methods missed with the biggest over-approximation

ISSTA 2024, Call Graph Soundness in Android Static Analysis, Jordan Samhi, René Just, Tegawendé F. Bissyandé, Michael D. Ernst, Jacques Klein

What is the cause of this **unsoundness**?

Frameworks


JACKSON JSON
Maven
JSON
Jsoup
JAXB
LOG4J
GUAVA
mockito
JFreeChart
ionic
React Native
appcelerator
Corona
Xamarin
Flutter
spring
STRUTS
django
NativeScript
HIBERNATE
Symfony

Using dynamic analysis to improve static analysis

Straightforward idea:

- Collect the entry point methods via dynamic analysis
- Feed these entry point methods to the static analyzer

Preliminary results:

- On 100 apps
- By dynamically analyzing the apps for 5 min each

	Average # of nodes	Median # of nodes
Without RD	50 626	25 899
With RD	65 534	46 307
	+29%	+79%

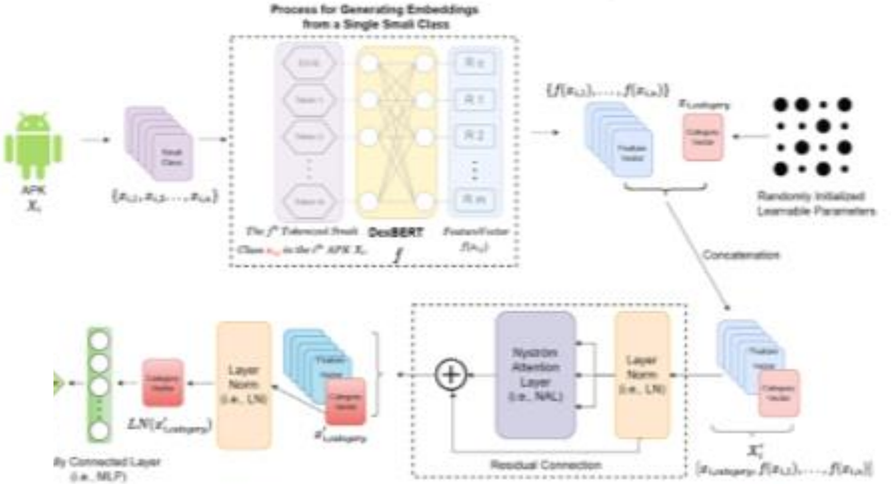
FSE IVR 2025, Do you have 5 min? Improving Call Graph Analysis with Runtime Information, Jordan Samhi, Marc Miltenberger, Marco Alecci, Steven Arzt, Tegawendé F. Bissyandé, Jacques Klein

AndroZoo for Large Scale Empirical Studies

Let's start with a simple question

Do you know what is inside an Android App?

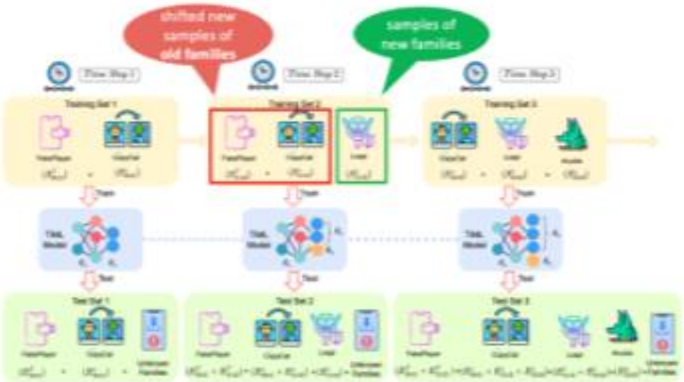
DetectBERT: Towards Full App-Level Representation Learning to Detect Android Malware



… Full App-Level Representation Learning to Detect Android Malware"

Summary

TIML: Temporal-Incremental Malware Learning



40%

methods missed with the biggest over-approximation

ISSTA 2024, Call Graph Soundness in Android Static Analysis, Jordan Samhi, René Just, Tegawendé F. Bissyandé, Michael D. Ernst, Jacques Klein