University of Luxembourg

Multilingual. Personalised. Connected.

AI for Software Vulnerabilities and Android Malware Detection

Prof. Dr. Jacques Klein

NLPAICS, Lancaster, UK, July 2024

UNIVERSITÉ DU LUXEMBOURG

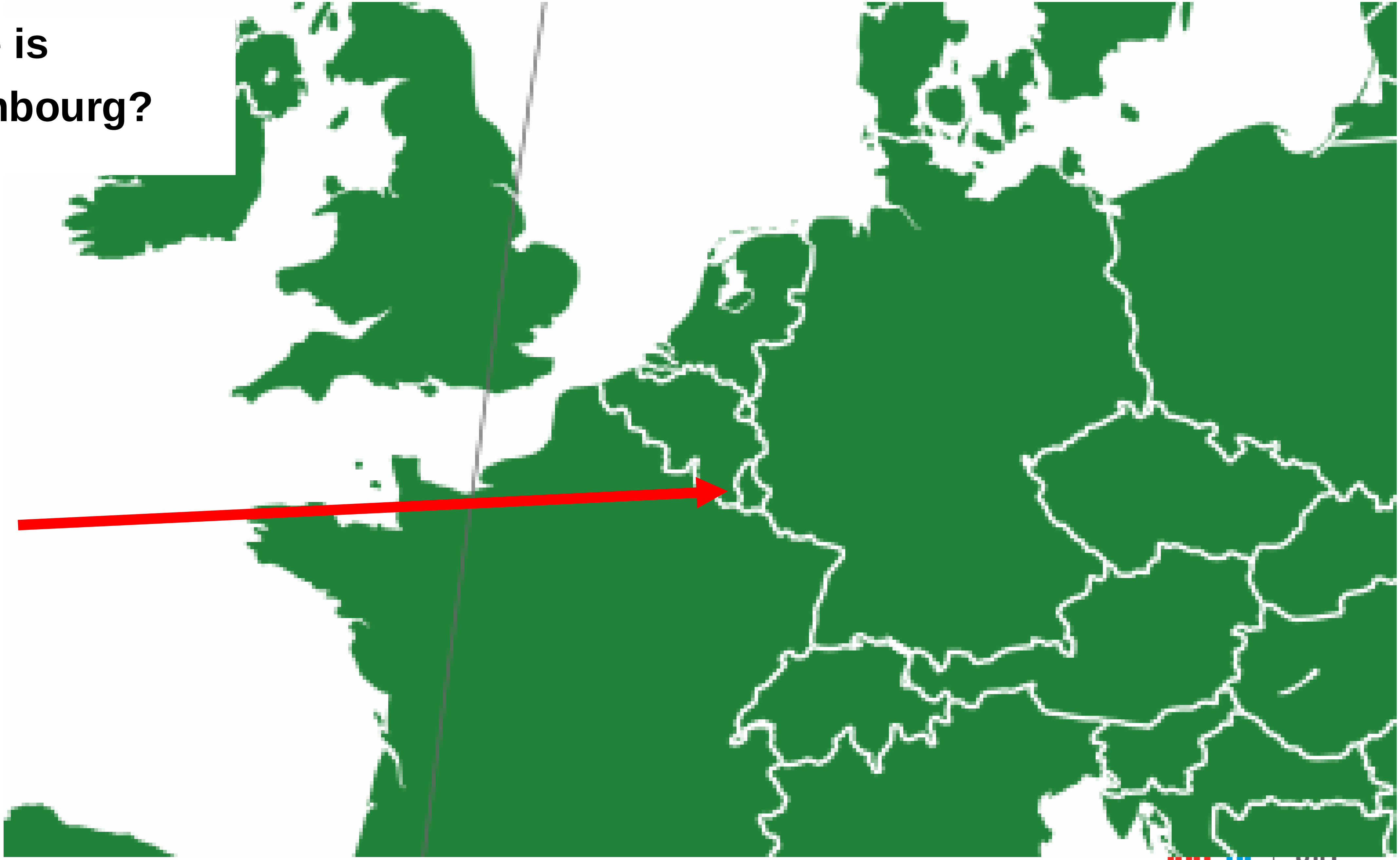
Who am I?

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

25th Young University

worldwide and #4 worldwide for its "international outlook" in the Times Higher Education (THE) World University Rankings 2023

7000
students

300
faculty members

60%
international students

1000+
PhDs

130
nationalities

The University of Luxembourg

Research Focus Areas

- Computer Science & ICT Security
- Finance and Financial Innovation
- Education
- Materials Science
- Contemporary and Digital History
- Interdisciplinary theme: Health and Systems Biomedicine
- Interdisciplinary theme: Data Modelling and Simulation

3 Faculties

Faculty of Science, Technology and Medicine

Faculty of Law, Economics and Finance

Faculty of Humanities, Education and Social Sciences

4 Interdisciplinary Centres

SnT

LCSB

C²DH

LCEL

The University of Luxembourg

Research Focus Areas

- Computer Science & ICT Security
- Finance and Financial Innovation
- Education
- Materials Science
- Contemporary and Digital History
- Interdisciplinary theme: Health and Systems Biomedicine
- Interdisciplinary theme: Data Modelling and Simulation

3 Faculties

Faculty of Science, Technology and Medicine

Faculty of Law, Economics and Finance

Faculty of Humanities, Education and Social Sciences

4 Interdisciplinary Centres

SnT

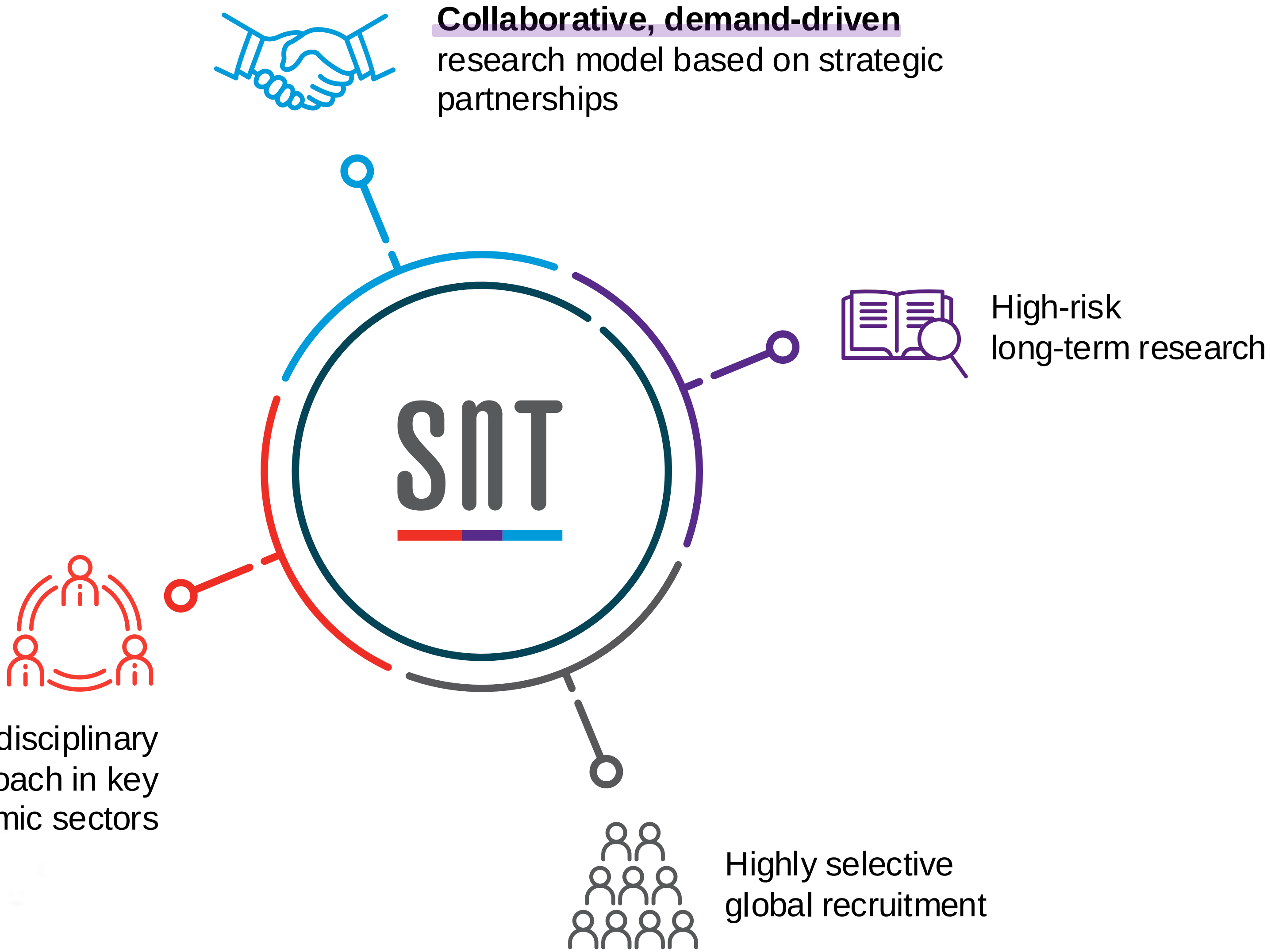
LCSB

C²DH

LCEL

Our vision

A leading international **research and innovation centre** in secure, reliable and trustworthy ICT systems and services. We play an instrumental role in Luxembourg by boosting R&D investments leading to economic growth and highly qualified talent.



Key Figures

PEOPLE

500+
Workforce

65+
Nationalities

31%
Alumni who stay in Luxembourg

50%
Doctoral Candidates on Industrial Projects

PARTNERSHIPS & INNOVATION

65+
Partners

8M
Partners annual contribution in Euros

70%
External project funding

6
Spin-offs

TruX

SnT

Trustworthy Software Engineering
TruX Research Group

Prof. Tegawendé F. BISSYANDE

Prof. Jacques KLEIN

UNIVERSITY OF LUXEMBOURG

TruX People

Professors

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

Visitors & Interns

1. Hocine REBATCHI
2. Yonghui LIU
3. Mohammad ANSARI

Research Associates

1. Abdoul Kader KABORE

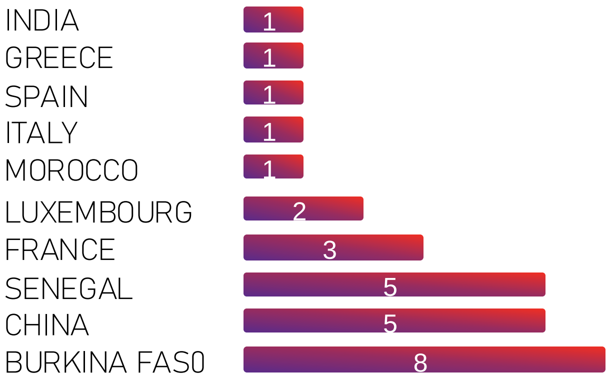
Assistant

- Fiona LEVASSEUR

Coming Soon

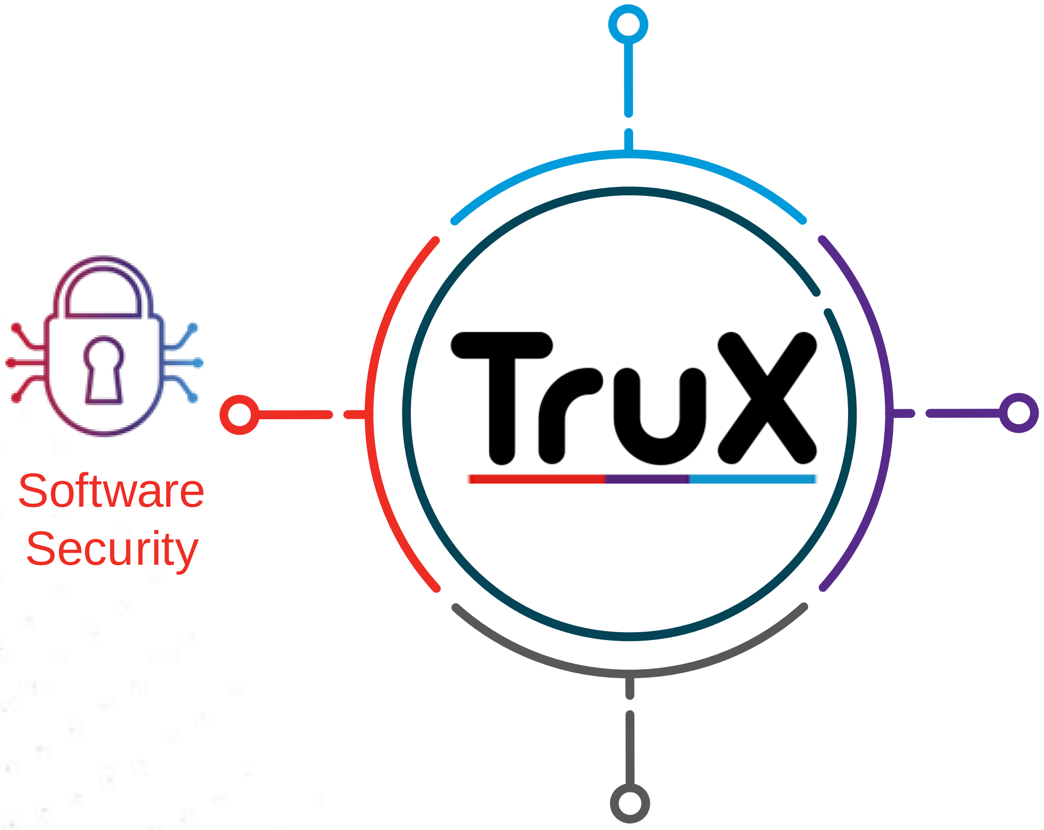
PhD Students

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



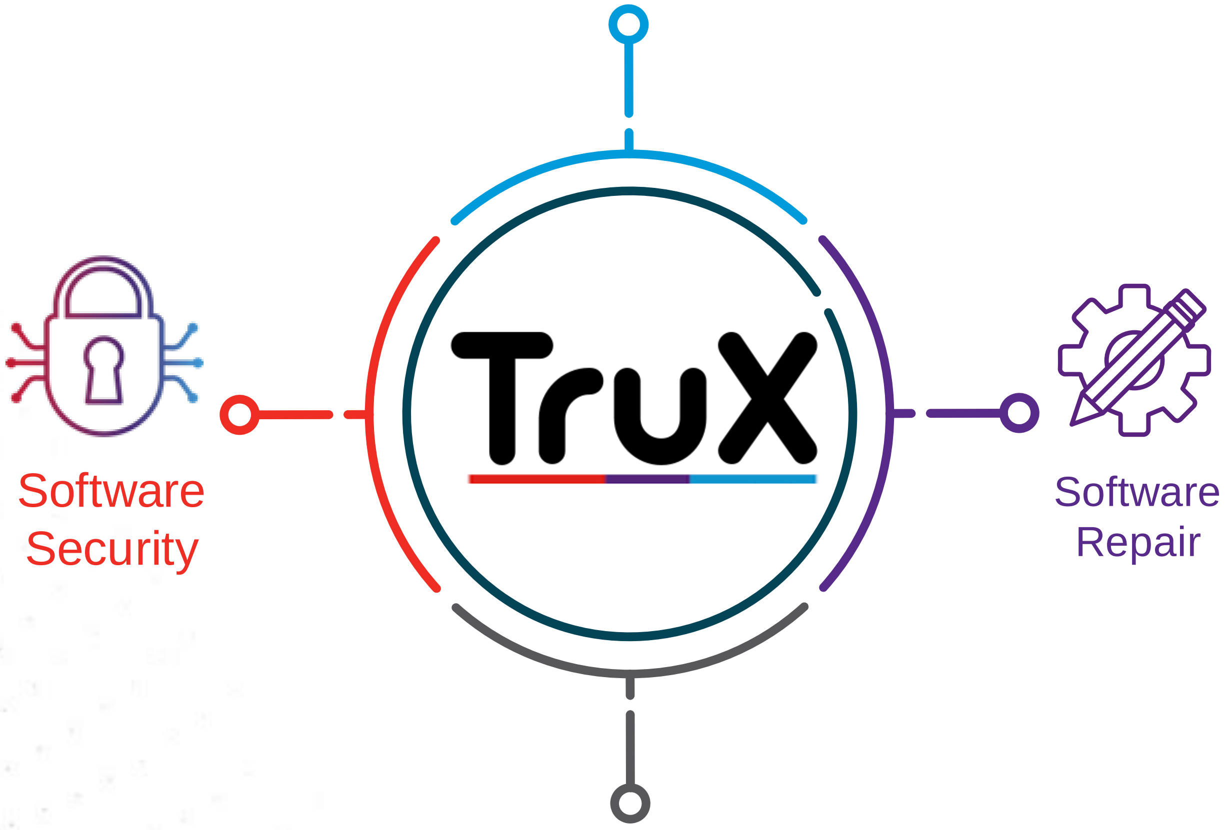
Trustworthy Software Engineering

- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection



Trustworthy Software Engineering

- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection

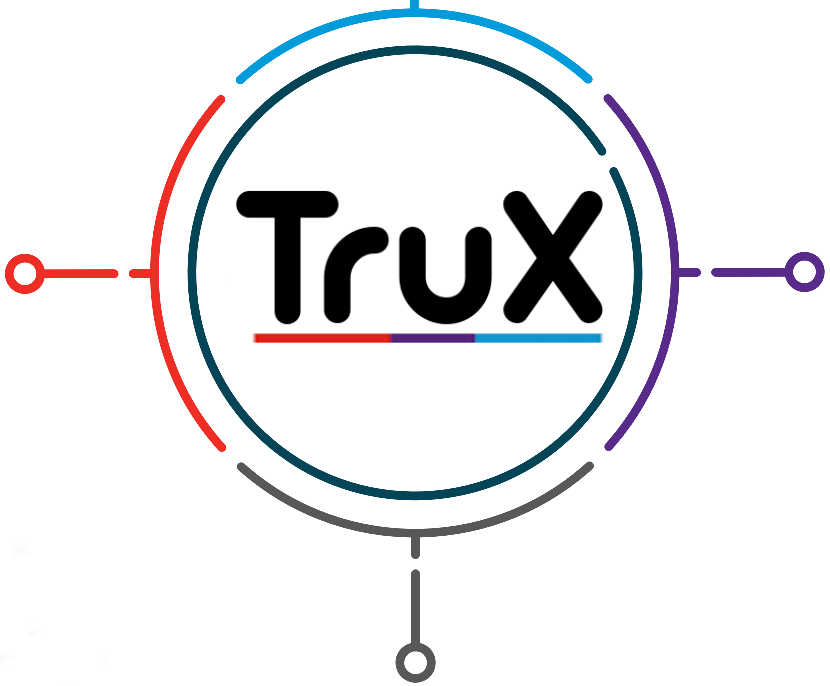


- Patch Recommendation
- Automated Program Repair
- **Bug Detection**
- Vulnerability patching

Trustworthy Software Engineering

Explainable Software

- Information Retrieval
- Natural Language Processing
- Time Series Pattern Recognition
- Machine learning, Explainable ML



Software Security

- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection

Software Repair

- Patch Recommendation
- Automated Program Repair
- **Bug Detection**
- Vulnerability patching

Trustworthy Software Engineering

Explainable Software

- Information Retrieval
- Natural Language Processing
- Time Series Pattern Recognition
- Machine learning, Explainable ML

Software Security

- **Vulnerability detection, Android app Analysis (e.g., Data Leaks)**
- GDPR compliance
- Malware Detection, Piggybacking Detection

TruX

Software Repair

- Patch Recommendation
- Automated Program Repair
- **Bug Detection**
- Vulnerability patching

Application Domains

- Android
- Fintech
- Smart Home
- Business Critical Systems

AI for
Software Vulnerabilities &
Android Malware Detection

UNIVERSITY OF LUXEMBOURG

To save time, let's skip the motivation slides ;)

I assume that we all agree that detecting malware and/or vulnerabilities is essential.

A G E N D A

Malware Detection

The need for a large set of Apps and a ground truth

Performance Assessment Issues

App Code Representation

- An app as a Image
- BERT-Based class representation
- Full App-level representation

Vulnerability Detection

Code is Spatial

WYSiWiM: Representing code as images

CodeGRID: Representing code as grids

Vulnerability Prediction with WYSiWiM and CodeGRID

A
G
E
N
D
A

Malware Detection

The need for a large set of Apps and a ground truth

Performance Assessment Issues

App Code Representation

- An app as a Image
- BERT-Based class representation
- Full App-level representation

Vulnerability Detection

Code is Spatial

WYSiWiM: Representing code as images

CodeGRID: Representing code as grids

Vulnerability Prediction with WYSiWiM and CodeGRID

Part I
AI for Android Malware Detection

Part I-A
Need for a large set of Apps

UNIVERSITY OF LUXEMBOURG

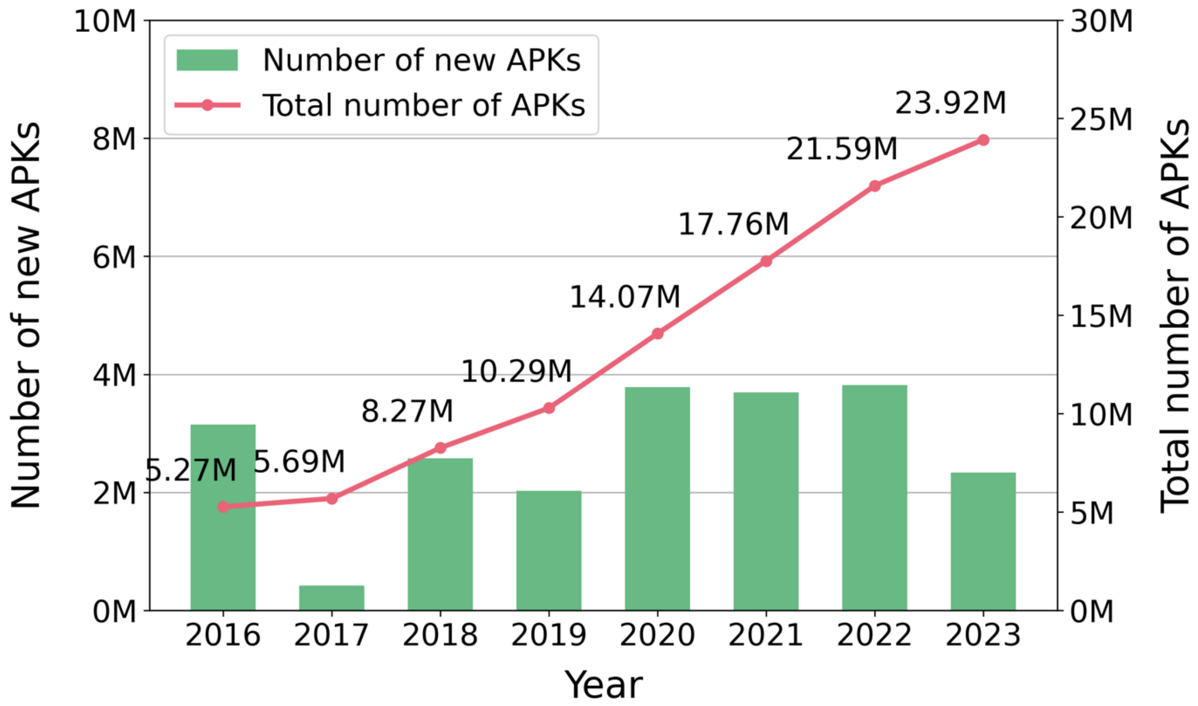
AndroZoo
A repository of Android Apps

[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 $\neq$ Apk

24 million apks, but 8 708 304 apps (average of 2.74 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

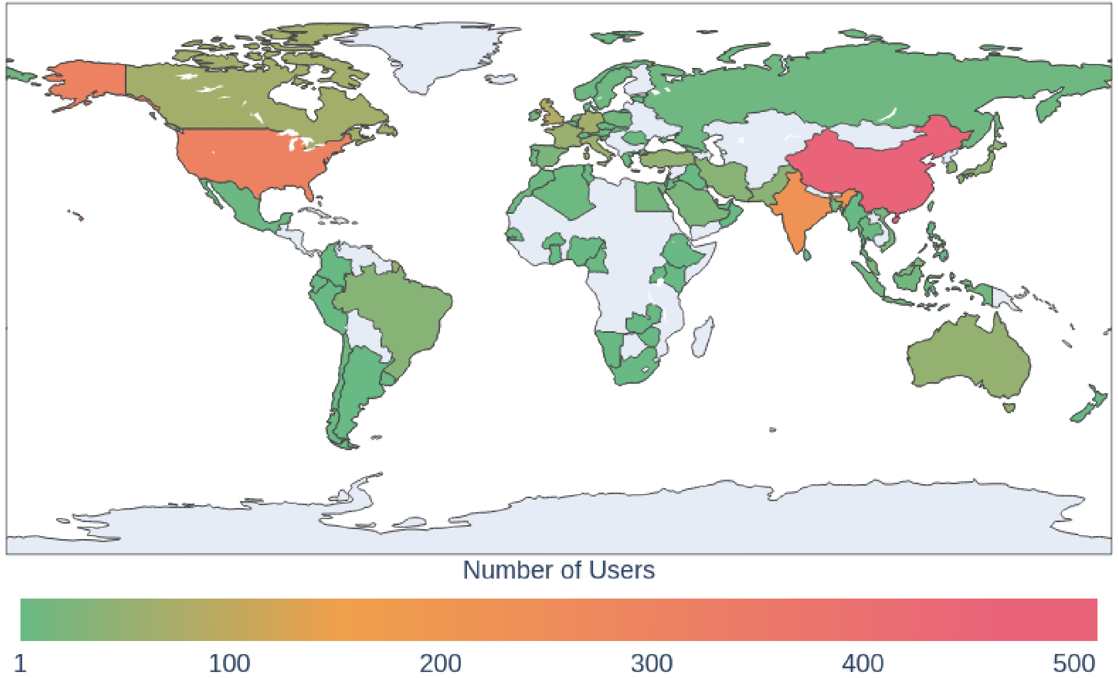
Table 4: Download Statistics from 11-2021 to 11-2023

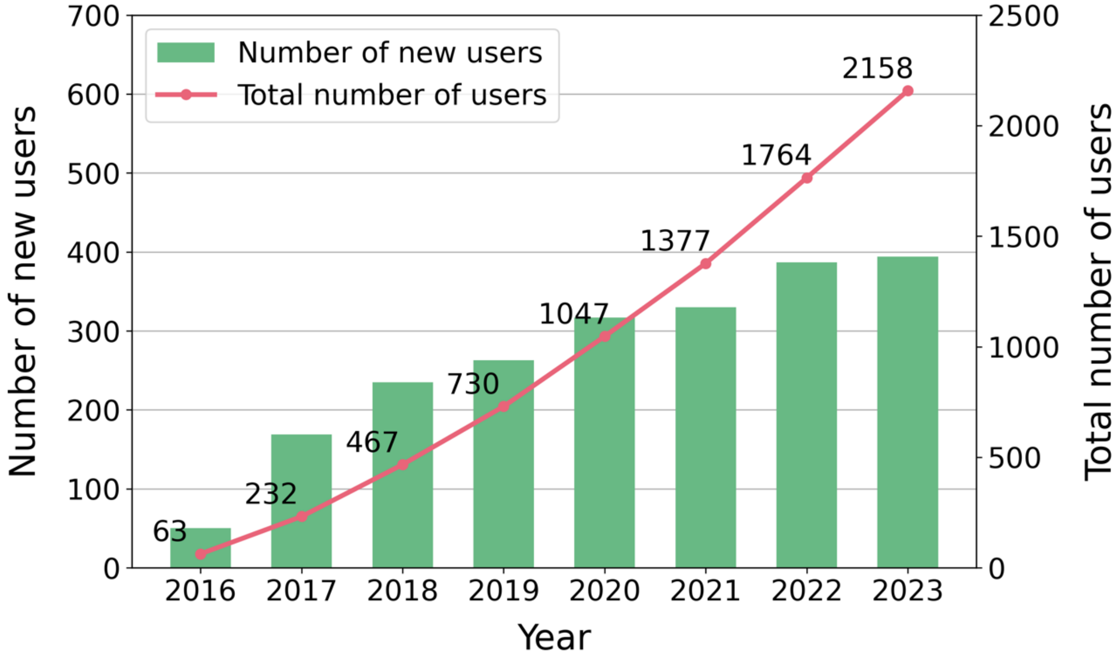
	Day	Month
Average Number of HTTP requests	502 083	15 393 045
Average Download Volume	5.8 TB	170 TB
Highest Number of HTTP requests	7 815 246	40 345 028
Highest Download Volume	31 TB	587 TB
Highest Number of Active Users	43	130

[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

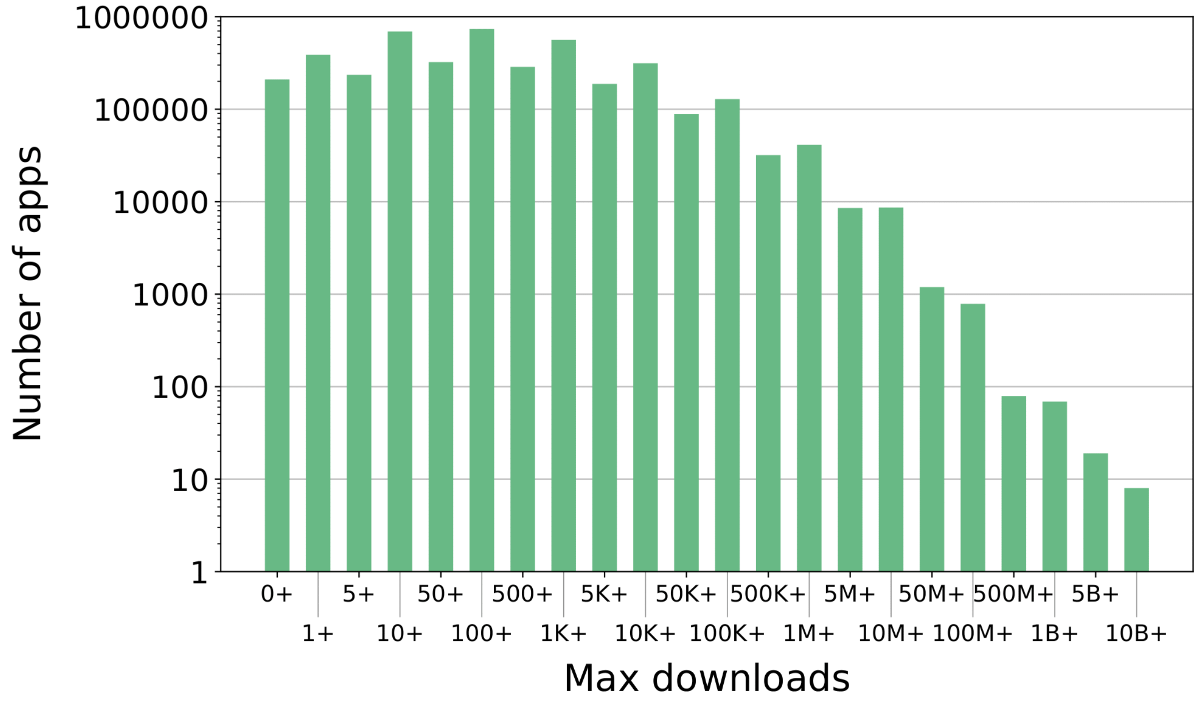
NEW

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 ….



AndroZoo for Malware Detection

=> Each App send to VirusTotal

A bit of Statistics

On 21,570,017 apks (from Google Play) sent to VirusTotal

Flagged by at least	# Apks	%
1 AV	1,787,482	8.29%
5 AVs	251,068	1.16%
10 AVs	85,782	0.4%
20 AVs	11,593	0.05%

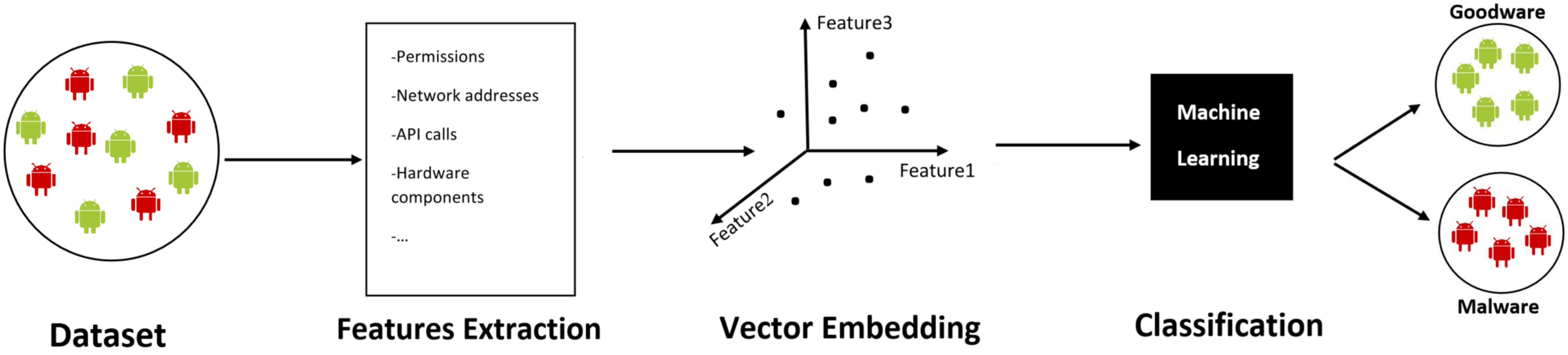
VirusTotal Limitations (among others)

- Disagreements among Antivirus products
  - [DIMVA2016] On the Lack of Consensus in Anti-Virus Decisions: Metrics and Insights on Building Ground Truths of Android Malware
  - [MSR2017] Euphony: Harmonious Unification of Cacophonous Anti-Virus Vendor Labels for Android Malware

- Malware / Adware
  - [SANER2017] Should You Consider Adware as Malware in Your Study?

Part I-B
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

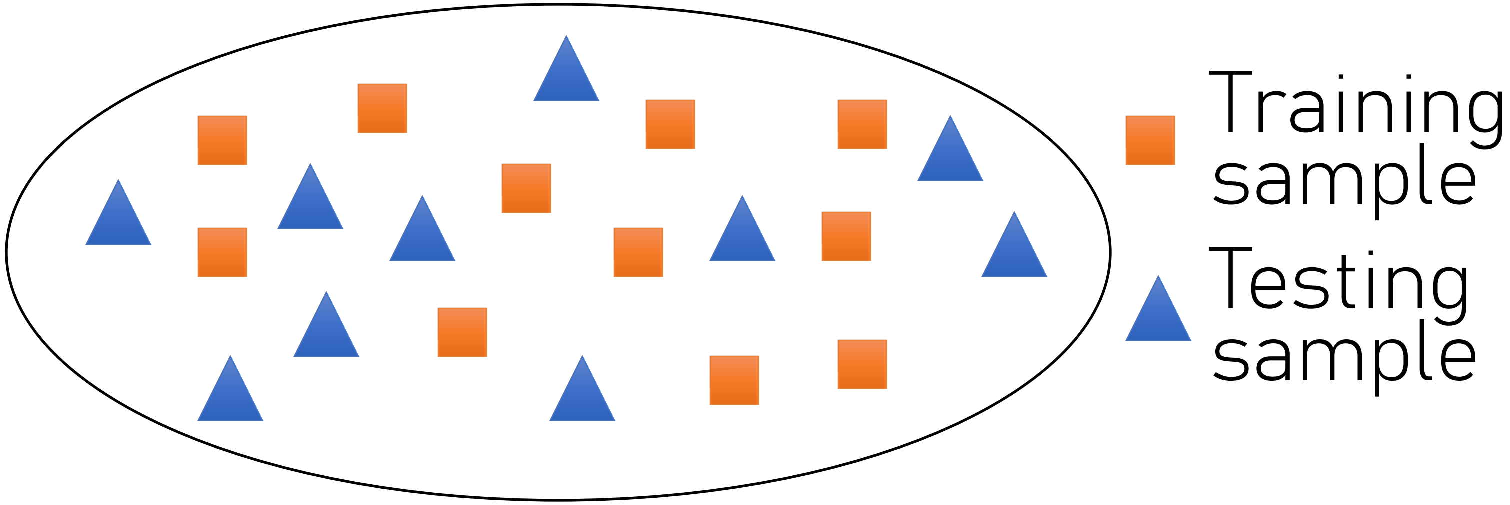
[EMSE2021] “Lessons learnt on reproducibility in machine learning based android malware detection”

Outstanding malware detection score of existing approaches

F1 score = 0.99

Machine Learning to detect Android Malware: main Outcomes

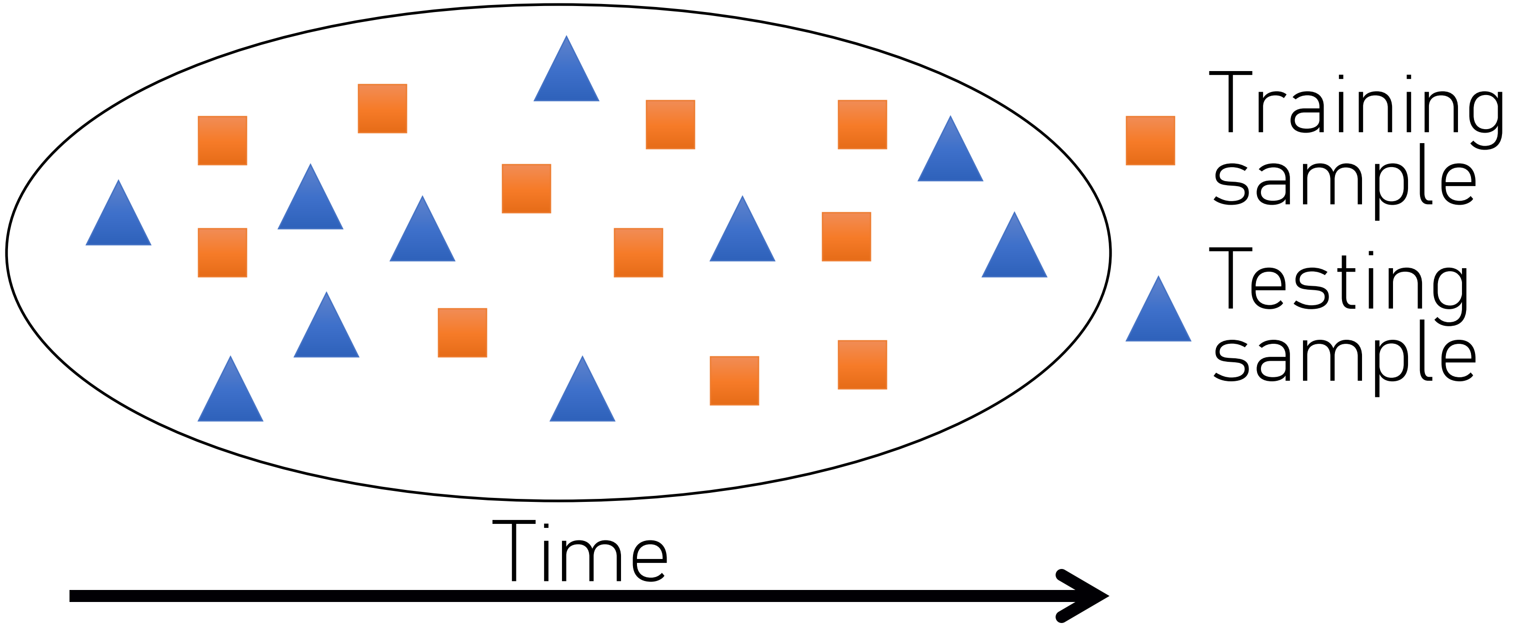
- Be careful about TIME! We don't know the future yet…



[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

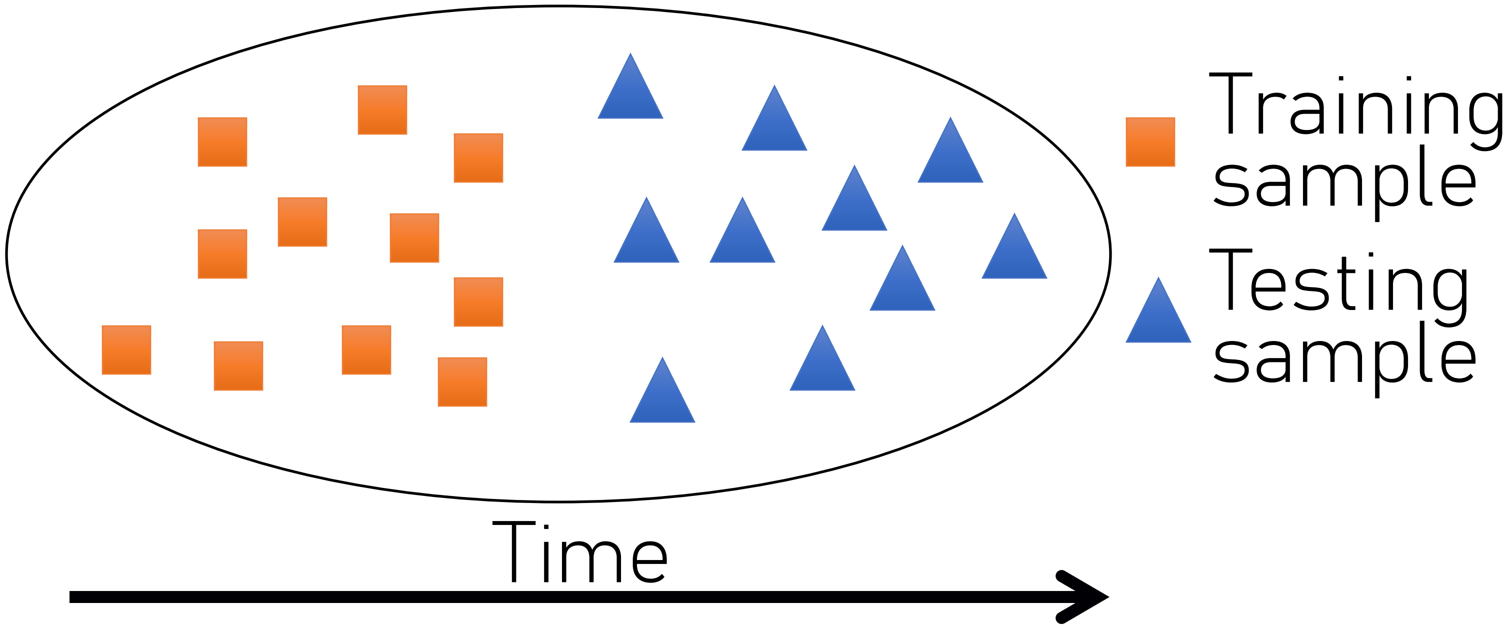
- Be careful about TIME! We don't know the future yet…



[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

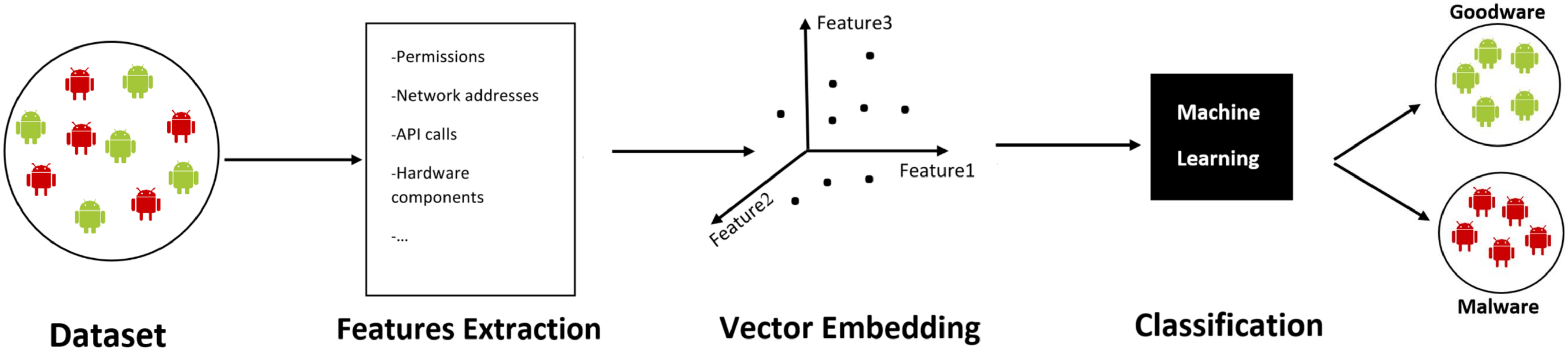
Ten-fold cross validation is not appropriated to assess machine learning-based malware detectors (paper at EMSE [2])

- 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

Part I-C
App Code Representation

Classical ML-based Android malware detection



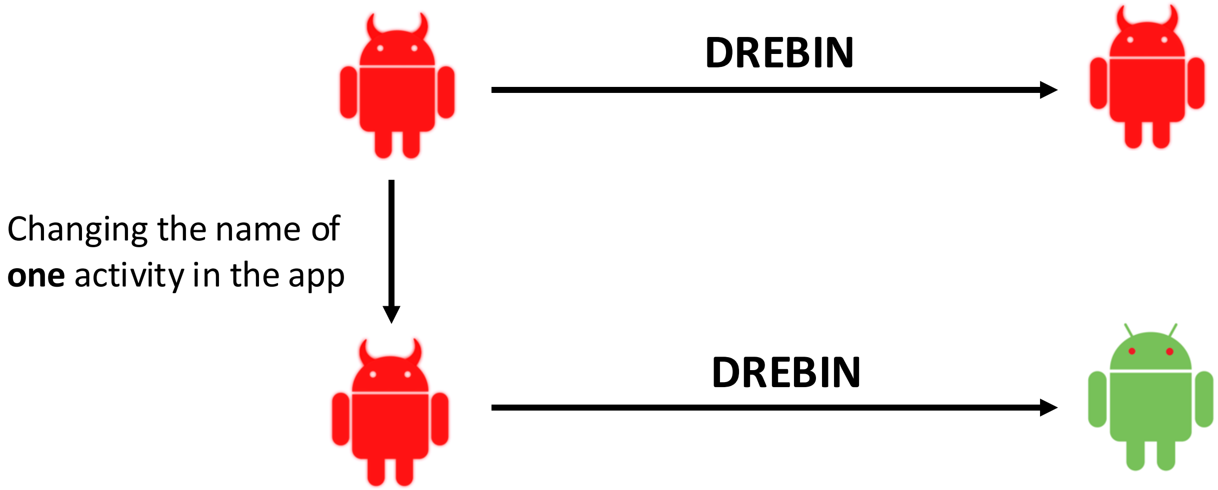
Building Blocks of Machine Learning-based Android malware detection

[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**

Flagged by 8 AV engines

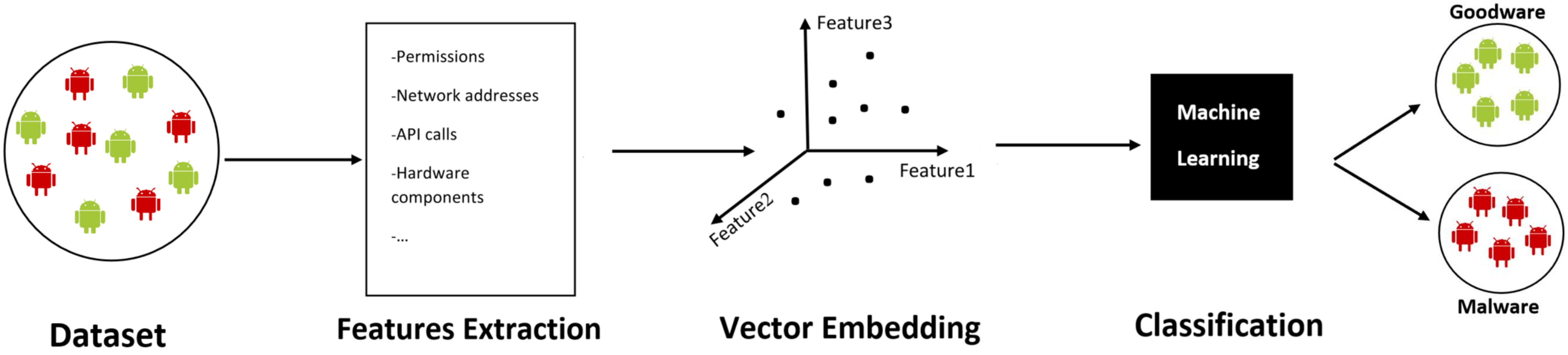


Findings:

- 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"

Classical ML-based Android malware detection



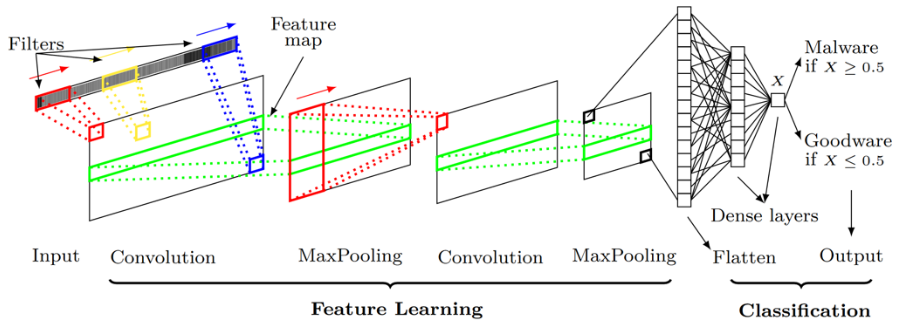
Building Blocks of Machine Learning-based Android malware detection

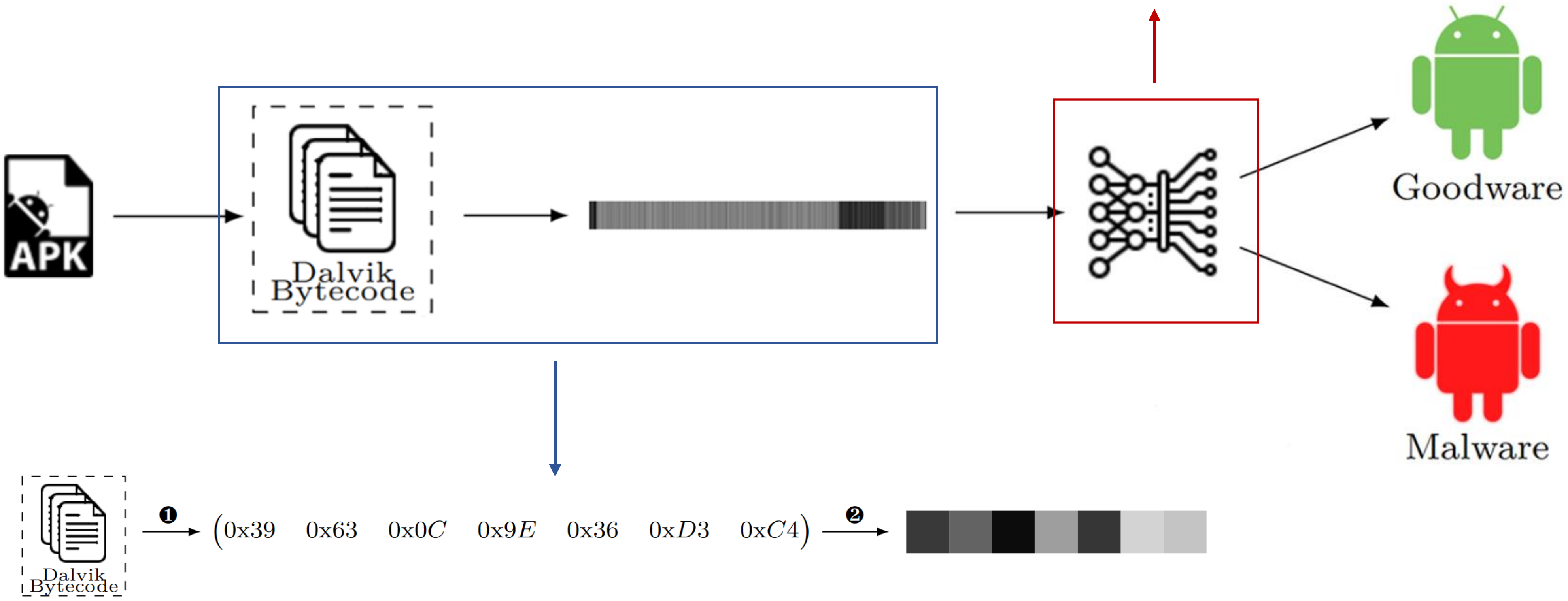
[EMSE2021] “Lessons learnt on reproducibility in machine learning based android malware detection”

Part I-C-1

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

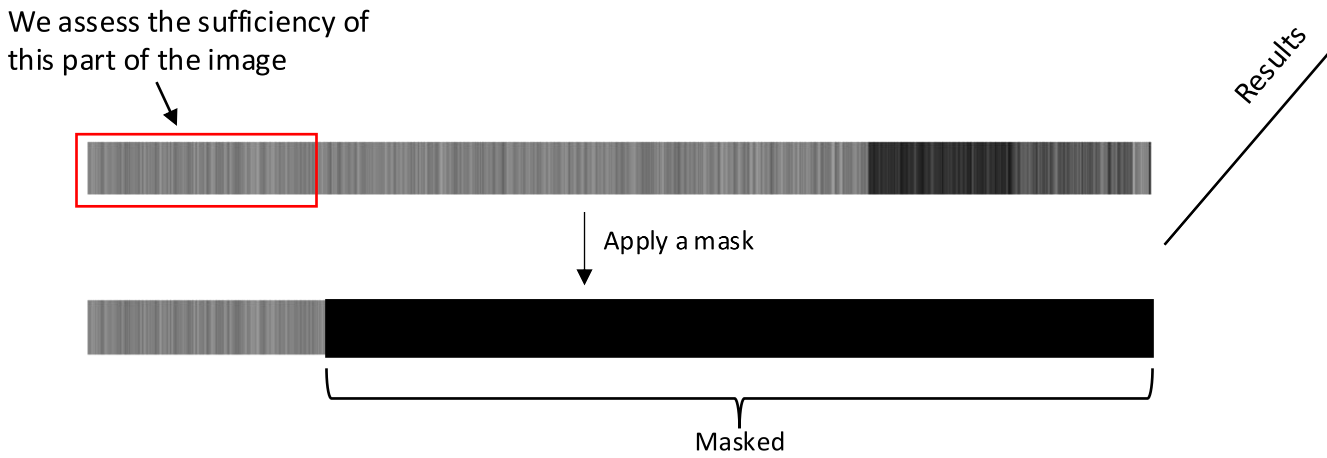
	Accuracy	Precision	Recall	F1-score
DexRay	0.97	0.97	0.95	0.96
Drebin	0.97	0.97	0.94	0.96
R2-D2	0.97	0.96	0.97	0.97
Ding et al.-Model 1	0.94	-	0.93	-
Ding et al.-Model 2	0.95	-	0.94	-
DexRay (Temporally Consistent)	0.97	0.97	0.98	0.98

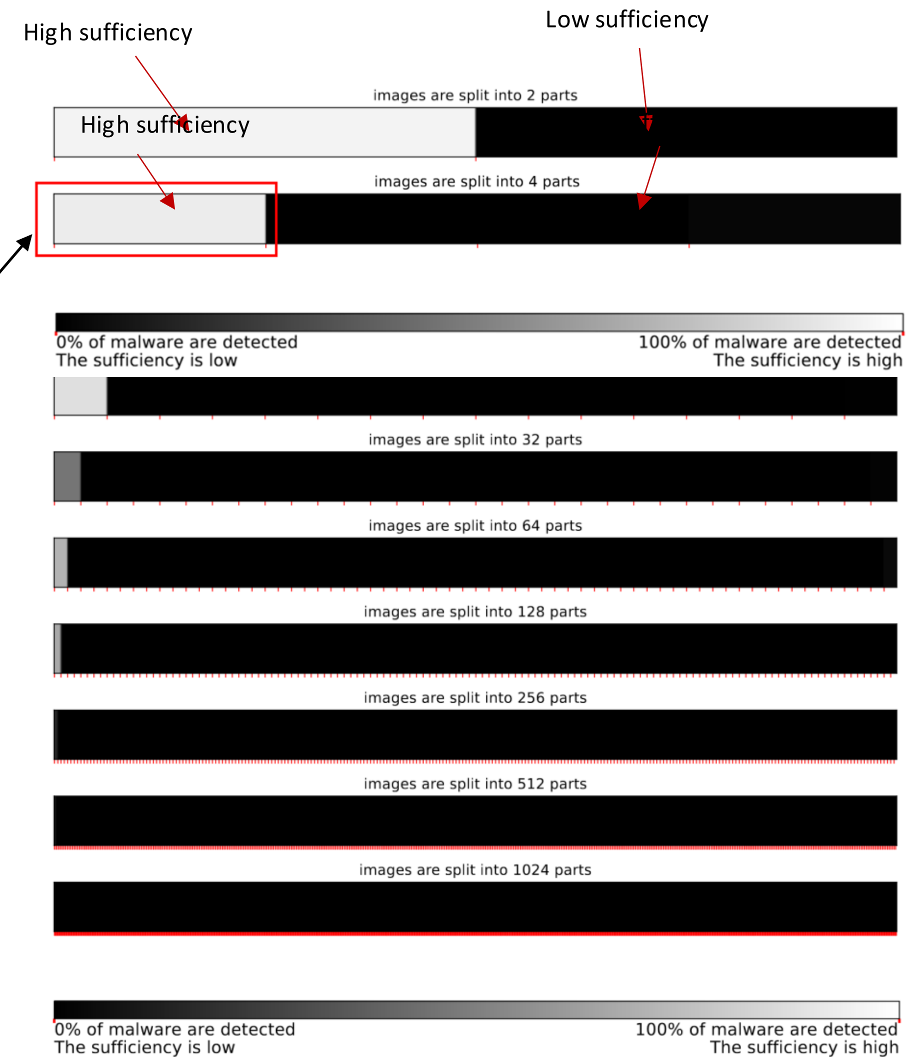
Findings:

- 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.

Possibility to localise malicious code

Sufficiency: A part of the image is sufficient for the detection if DexRay predicts the malware app as malware when only this part of the image is kept, and the rest is masked





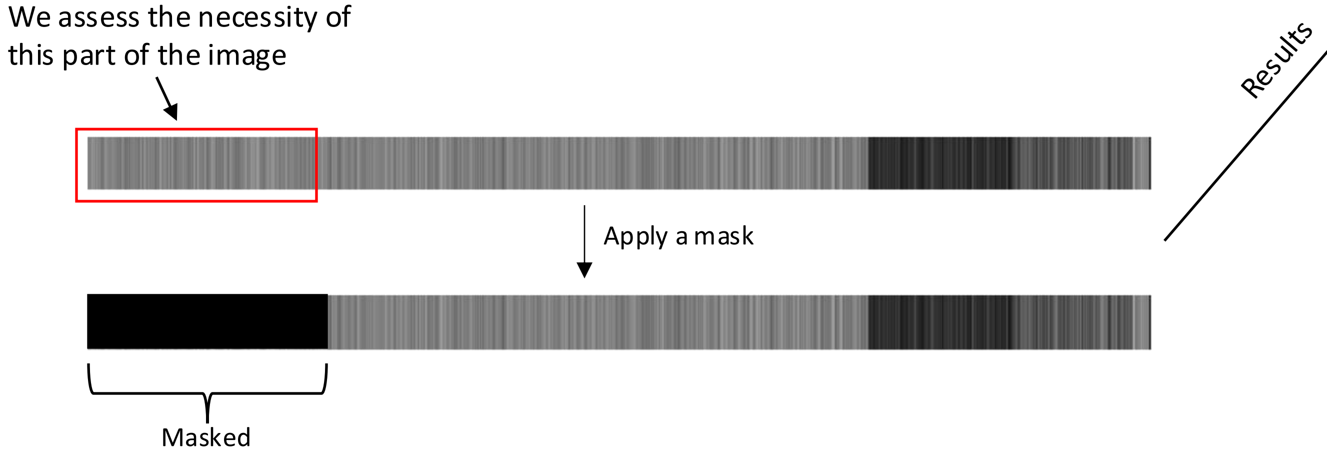
Sufficiency for malware images:
High (resp low) sufficiency is represented by white (resp black) colour

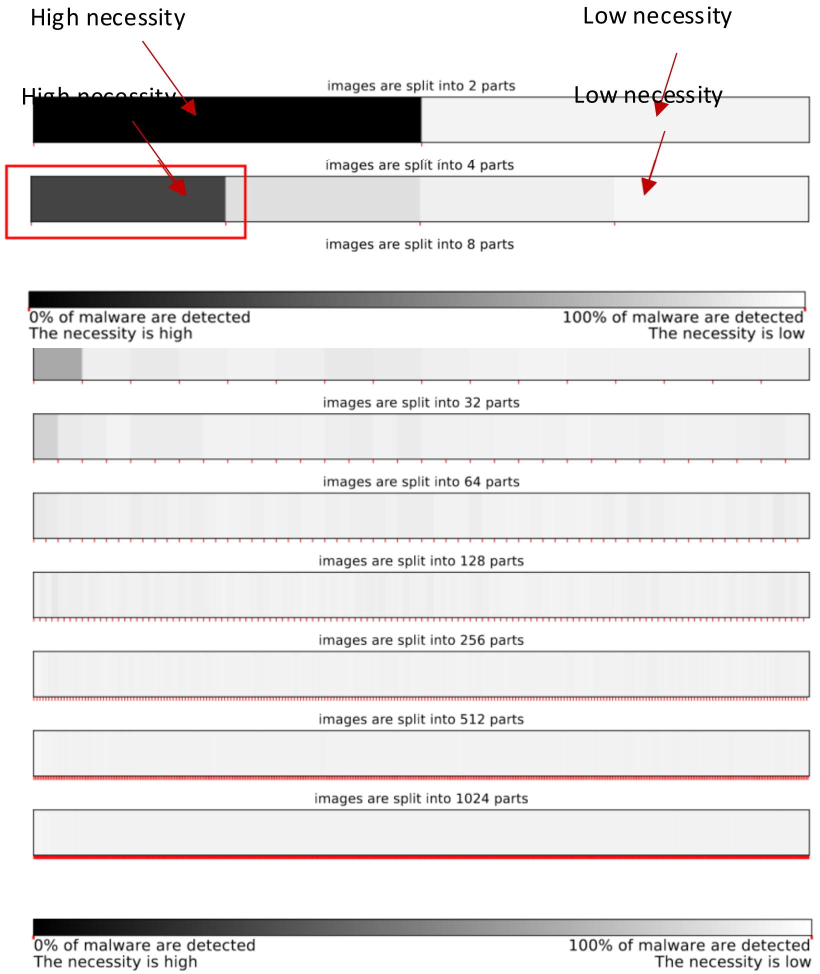
Findings:

- The first half of the vector images is highly sufficient to detect malware, while the second half is almost never sufficient.
- The sufficiency of the first pixels in the images generally decreases when their size decreases.

Possibility to localise malicious code

Necessity : A part of the image is necessary for the detection if DexRay predicts the malware app as benign when this part of the image is masked, and the rest is kept unchanged





Necessity for malware images:

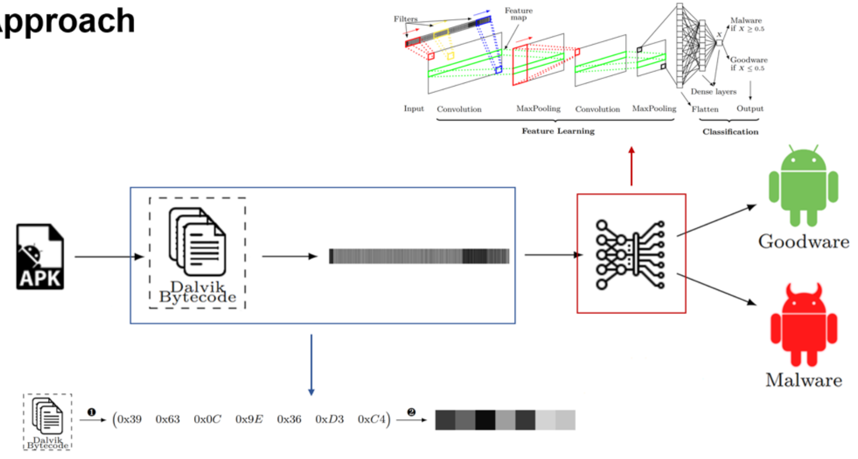
High (resp low) necessity is represented by black (resp white) colour

Findings:

- The first half of the vector images is highly necessary to detect malware.
- The necessity of the first pixels in the images generally decreases when their size decreases.

Summary

Approach



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

uni.lu | SnT

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

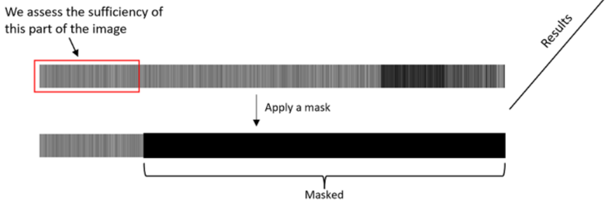
Findings:

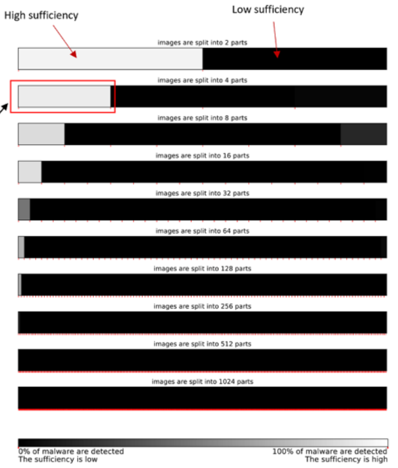
- DexRay yields performance metrics that are comparable to the state of the art.
- Its simplicity has not hindered its performance when compared to similar works presenting sophisticated configurations.

uni.lu | SnT

Possibility to localise malicious code

Sufficiency: A part of the image is sufficient for the detection if DexRay predicts the malware app as malware when only this part of the image is kept, and the rest is masked





Sufficiency for malware images:
High (resp low) sufficiency is represented by white (resp black) colour

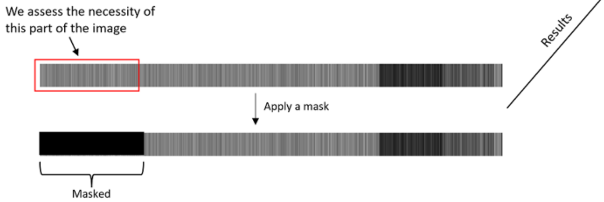
Findings:

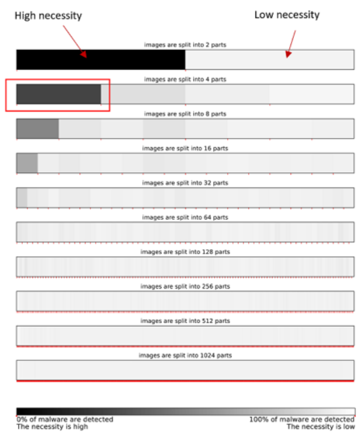
- The first half of the vector images is highly sufficient to detect malware, while the second half is almost never sufficient.
- The sufficiency of the first pixels in the images generally decreases when their size decreases.

uni.lu | SnT

Possibility to localise malicious code

Necessity : A part of the image is necessary for the detection if DexRay predicts the malware app as benign when this part of the image is masked, and the rest is kept unchanged





Necessity for malware images:
High (resp low) necessity is represented by black (resp white) colour

Findings:

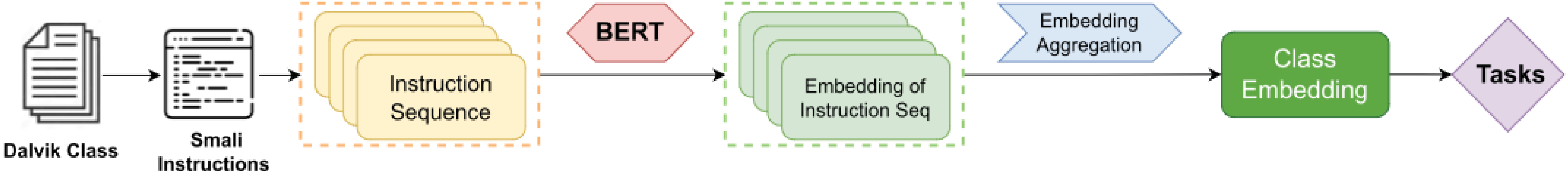
- The first half of the vector images is highly necessary to detect malware.
- The necessity of the first pixels in the images generally decreases when their size decreases.

uni.lu | SnT

Part I-C-2

DexBERT: Class level Representation

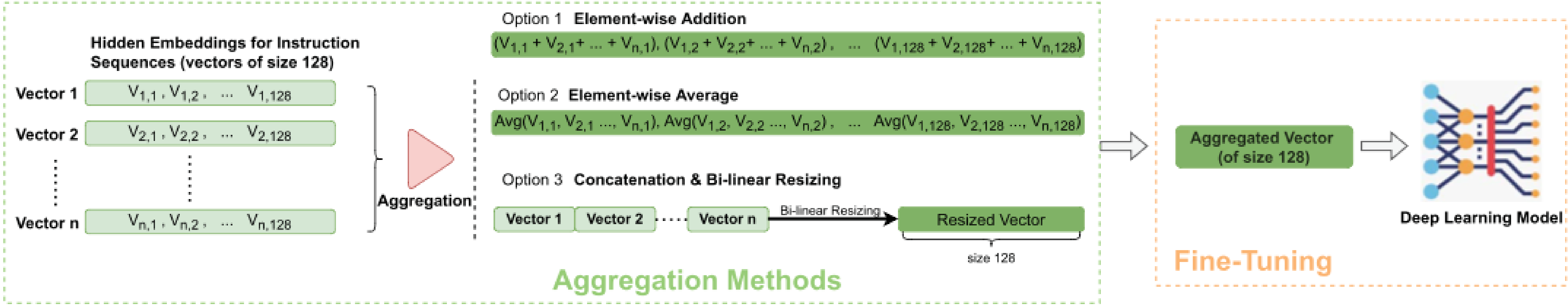
DexBERT: Effective, Task-Agnostic and Fine-Grained Representation Learning of Android Bytecode



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



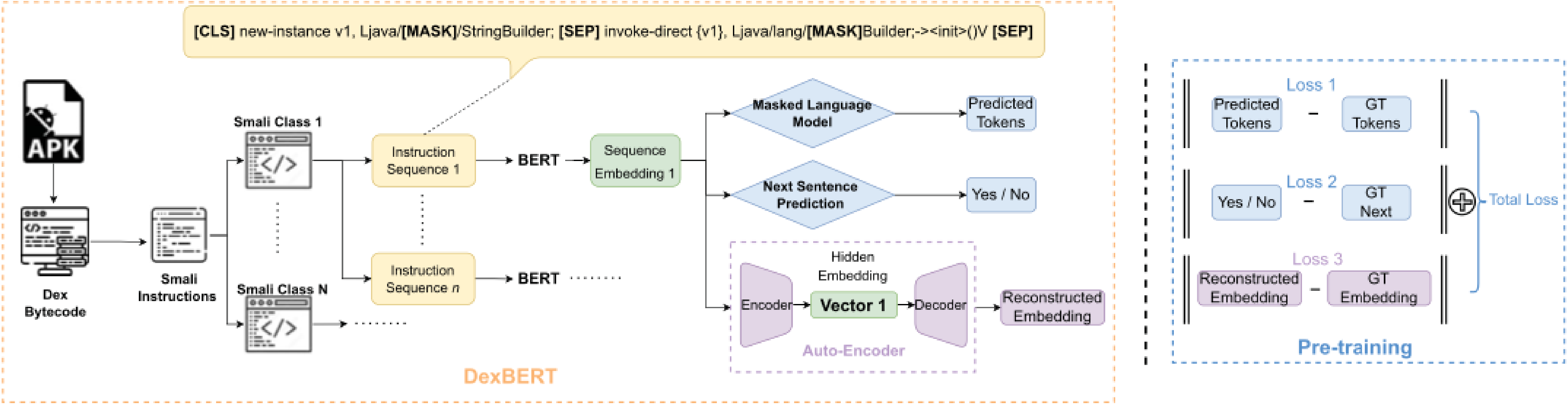
Three embedding aggregation methods and fine-tuning of downstream tasks.

(Addition is working the best)

[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

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

[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.

[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"

SnT

Part I-C-3

Full App-level Representation

UNIVERSITY OF LUXEMBOURG

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

DexBERT for Android class embedding

+

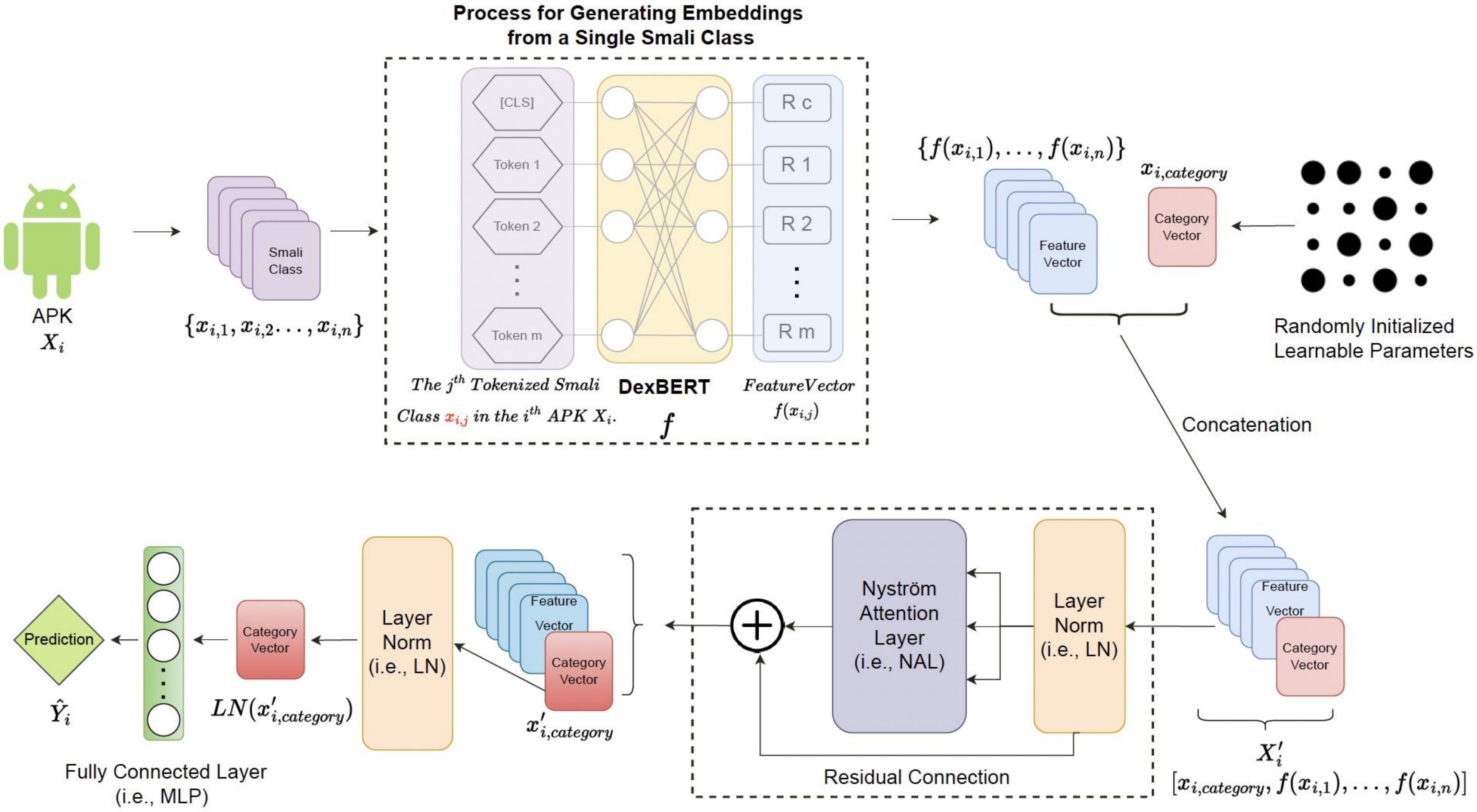
LaFiCMIL

(Correlated Multiple Instance Learning)

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

DetectBERT

DetectBERT: 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

Perspectives

Ground truth quality

Enhanced app representation

Malicious code localisation

Explainability

Artifacts availability and reproducibility

A
G
E
N
D
A

Malware Detection

The need for a large set of Apps and a ground truth

Performance Assessment Issues

App Code Representation

An app as a Image

BERT-Based class representation

Full App-level representation

A
G
E
N
D
A

Malware Detection

The need for a large set of Apps and a ground truth

Performance Assessment Issues

App Code Representation

- An app as a Image
- BERT-Based class representation
- Full App-level representation

Vulnerability Detection

Code is Spatial

WYSiWiM: Representing code as images

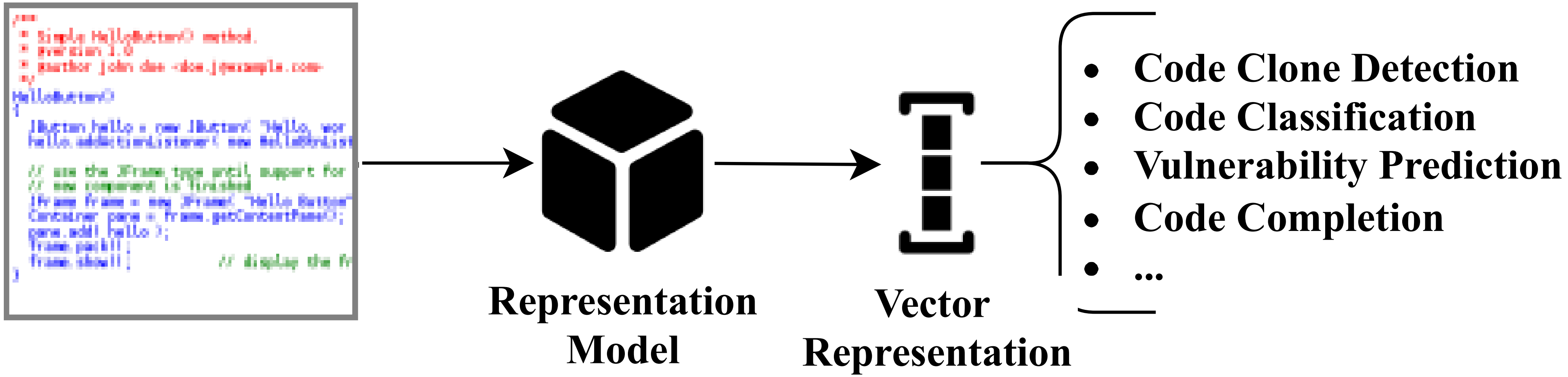
CodeGRID: Representing code as grids

Vulnerability Prediction with WYSiWiM and CodeGRID

Part II
Vulnerability Detection

Part II-A
Code is Spatial

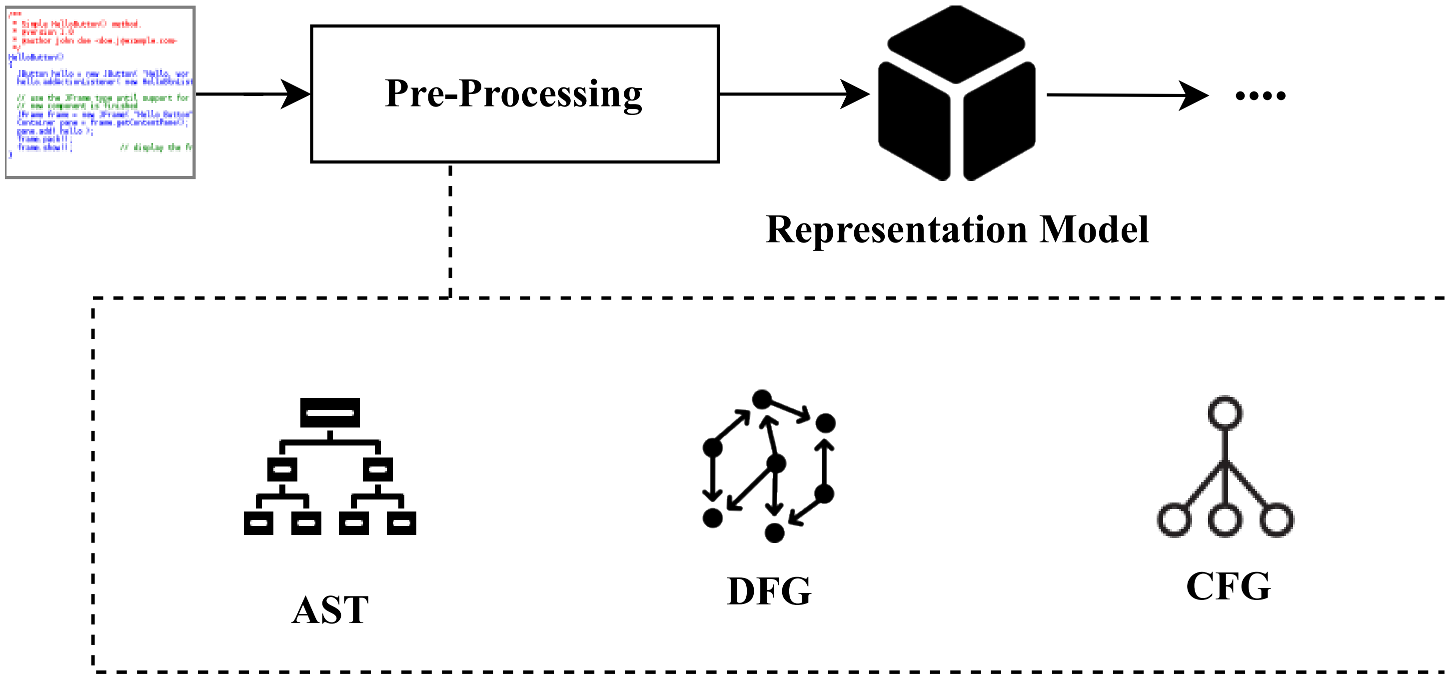
Code representation for ML



- NLP-based representations are effective
- but doesn't exploit the full richness of the code

Code representation for ML

- Code is also about structure



? Other signals may remain unexploited

Code is also spatial

- Every single character can be positioned using x_i and y_i coordinates.

The spatial nature of the code matters

```
int robert_age = 32;
int annalouise_age = 25;
int bob_age = 250;
int dorothy_age = 56;
```

(a) Standard Coding Style.

```
int     robert_age = 32;
int annalouise_age = 25;
int        bob_age = 250;
int    dorothy_age = 56;
```

(b) Grid Alignment.

The shared suffix and the 250 outlier are obscured on the left and jump on the right.

- New code representations using code spatiality as a new signal
- Leverage **computer vision** techniques to perform SE tasks

SnT

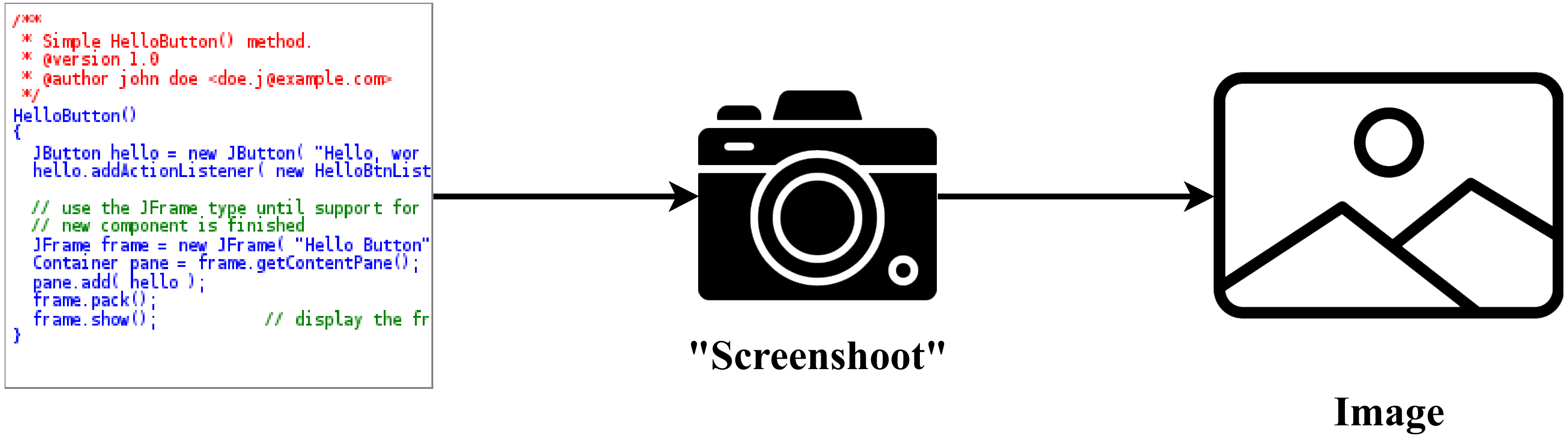
Part II-B

WYSiWiM: Representing code as images

UNIVERSITY OF LUXEMBOURG

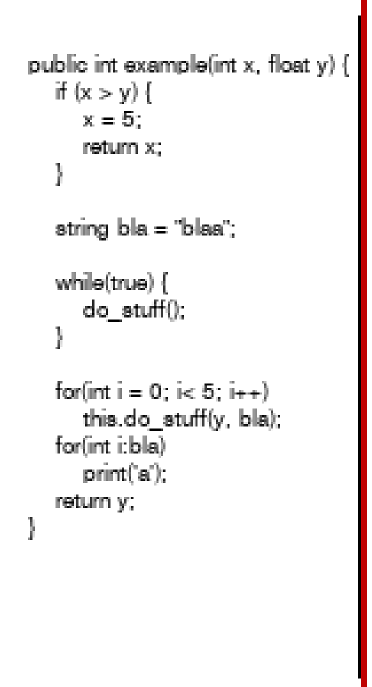
WYSiWiM

- The naive exploitation of code spatiality
- WYSiWiM: What You See is What it Means!

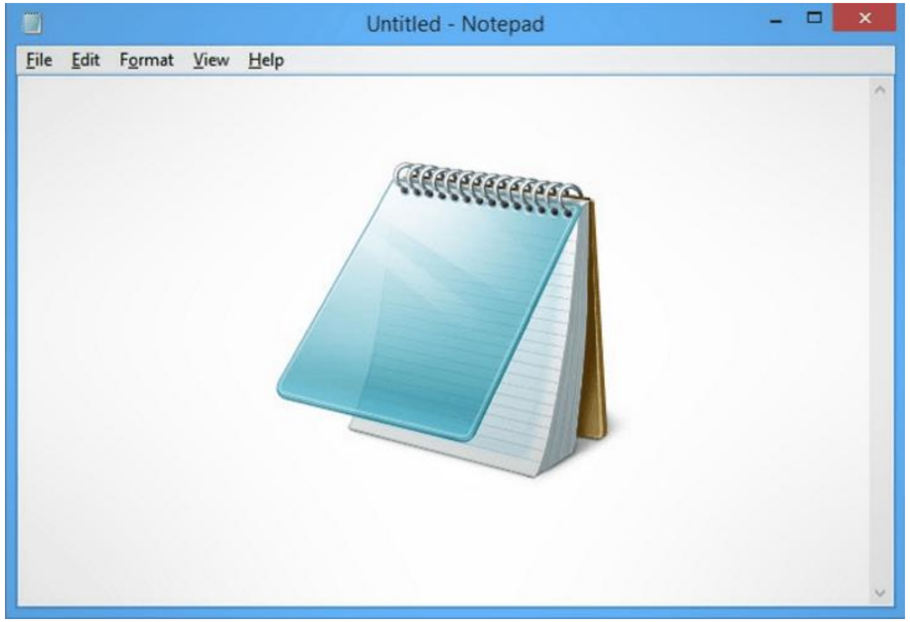


[TOSEM2021] "What You See is What it Means! Semantic Representation Learning of Code based on Visualization"

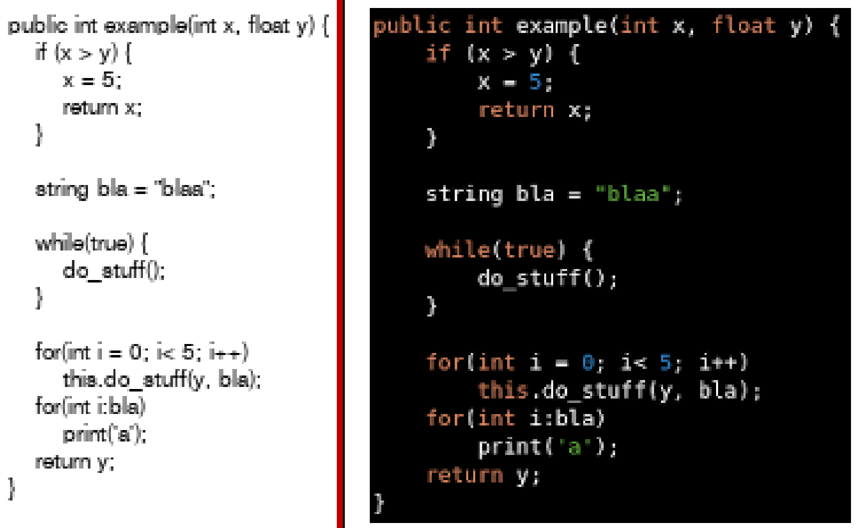
WYSiWiM: four visualization variants



a) Plain Text

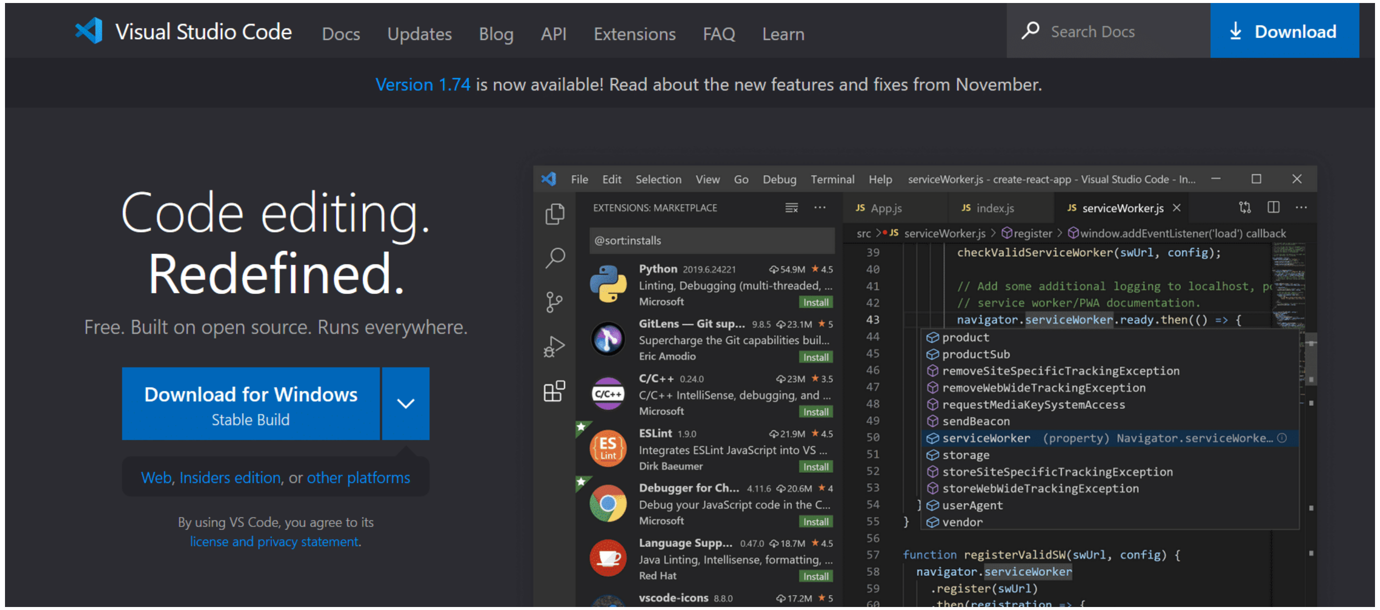


WYSiWiM: four visualization variants

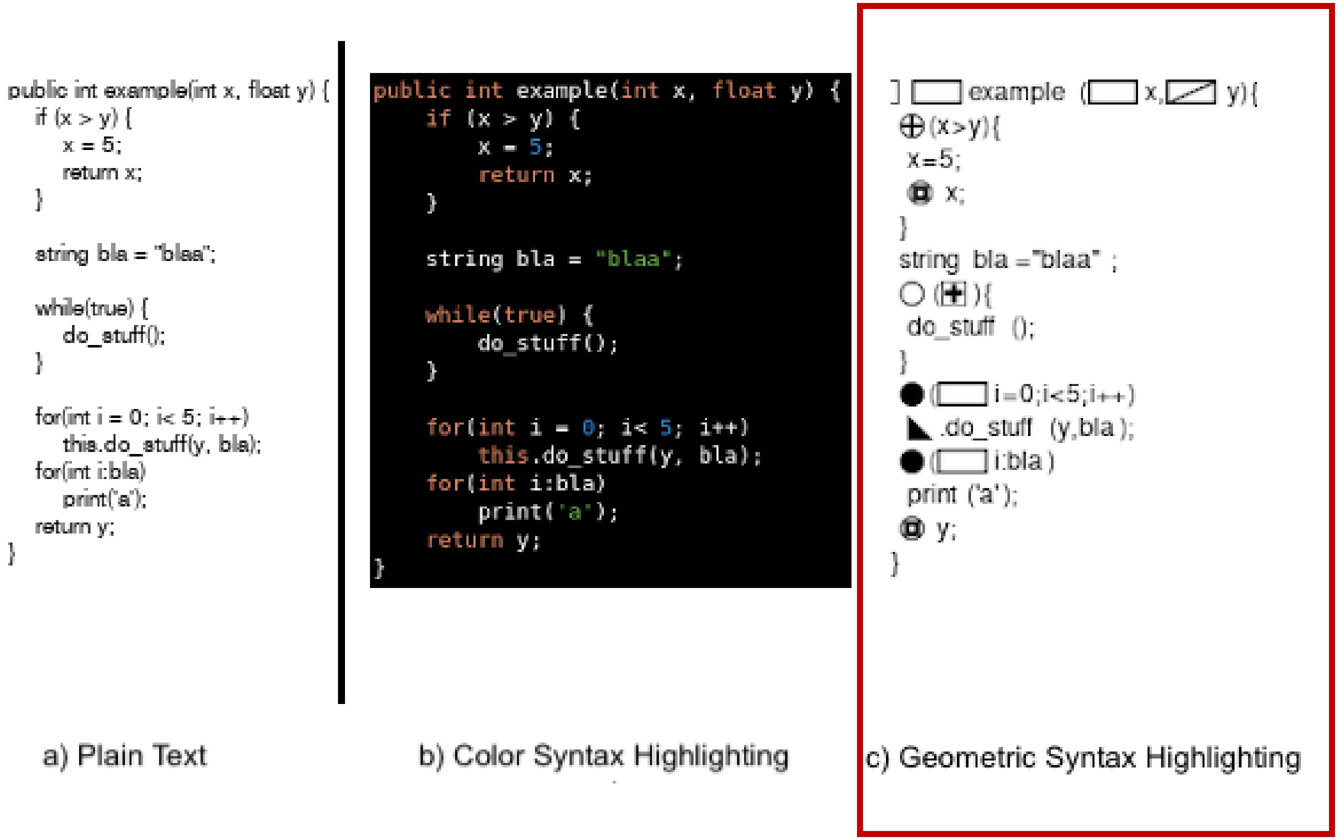


a) Plain Text

b) Color Syntax Highlighting

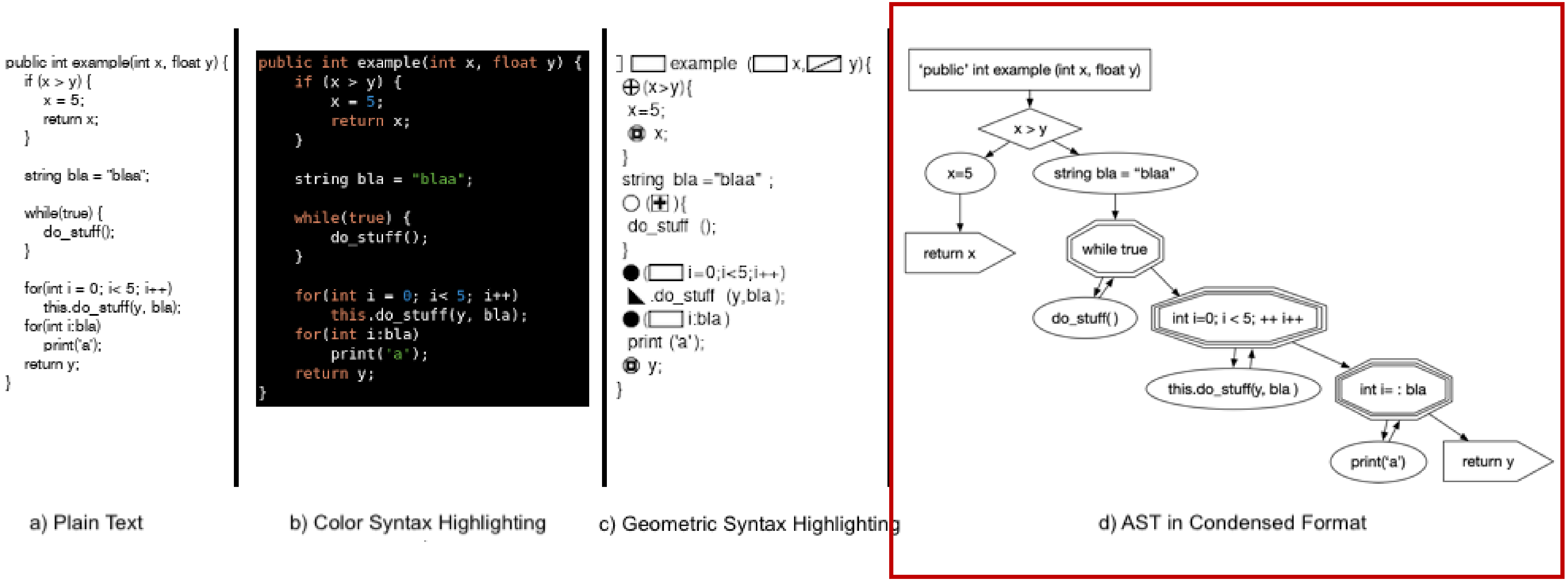


WYSiWiM: four visualization variants

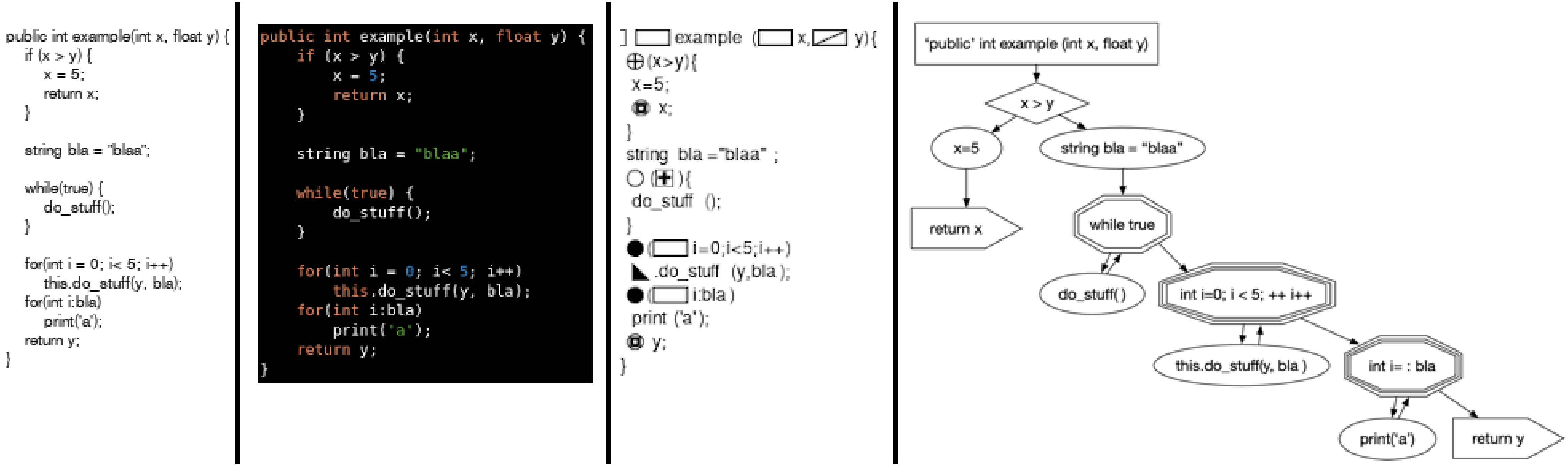


- Mapping and replacing some keywords with geometric form

WYSiWiM: four visualization variants



WYSiWiM: four visualization variants



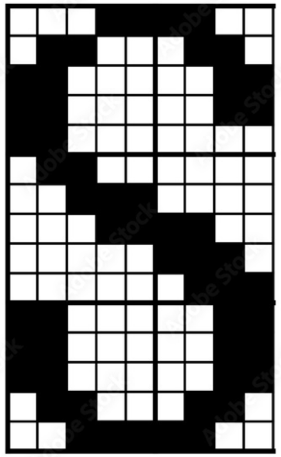
a) Plain Text b) Color Syntax Highlighting c) Geometric Syntax Highlighting d) AST in Condensed Format

WYSiWiM (limitations)

- Code as images: a naive approach:
  - Relying on image pixels: too noisy

  → Impossible to fit a single character in one pixel

  → May be difficult to learn, even with best computer vision techniques



[TOSEM2021] “What You See is What it Means! Semantic Representation Learning of Code based on Visualization”

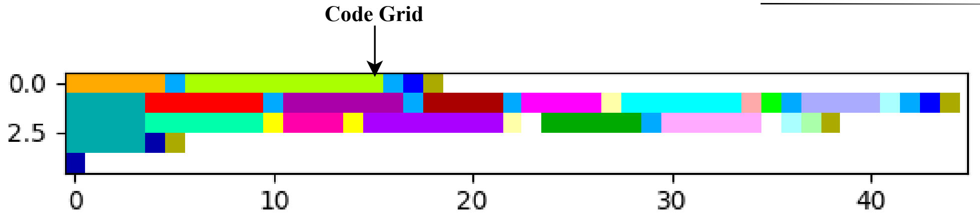
Part II-B

CodeGRID: Representing code as Grids

CODEGRID: REPRESENTING CODE AS GRIDS

Code

```java
class HelloWorld {
    // Hello World Program
    public static void main(String[] args) {
        System.out.println("hello World");
    }
}
```

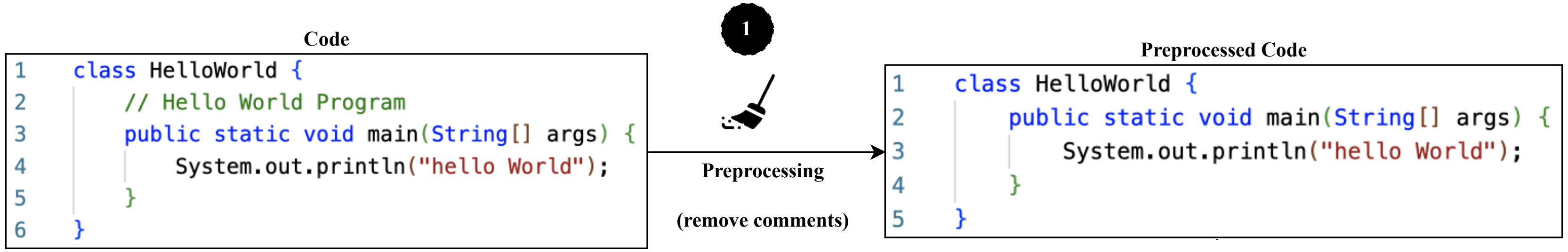


Grid Visualization

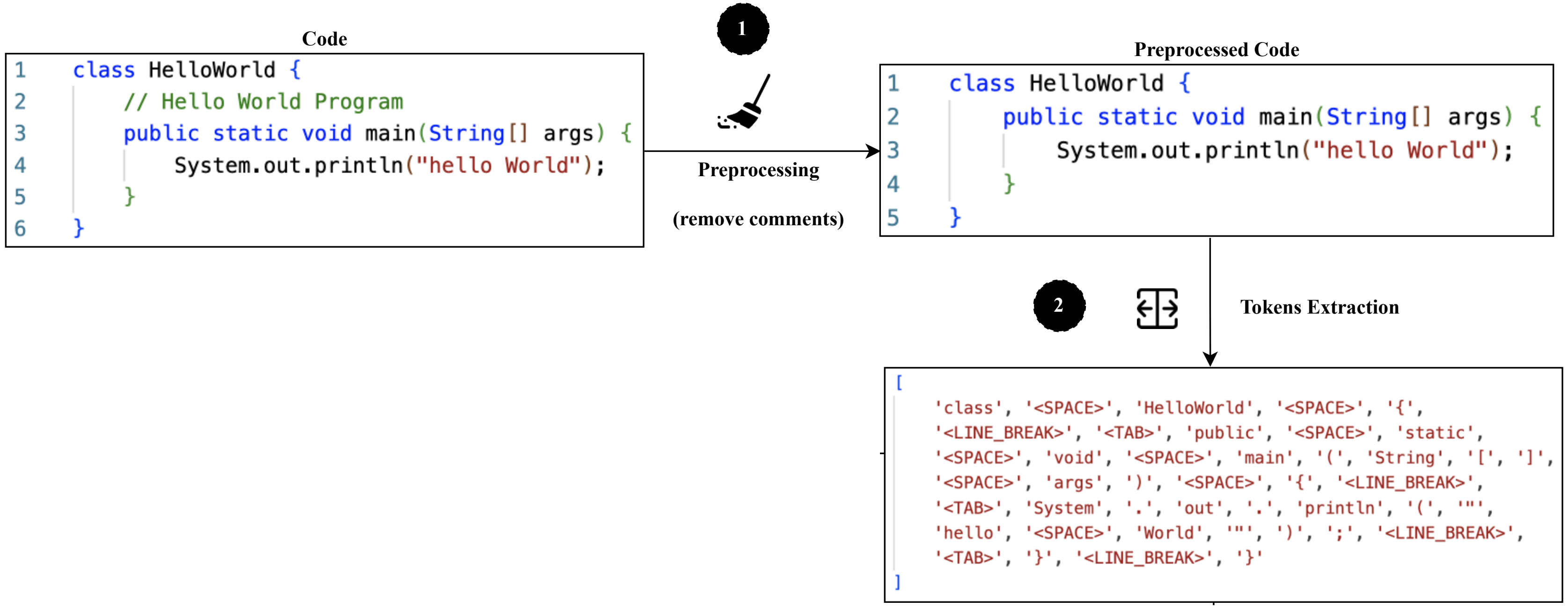
With “Color Vectorizer”

[ISSTA2023] “CodeGrid: A Grid Representation of Code”

CODEGRID: REPRESENTING CODE AS GRIDS



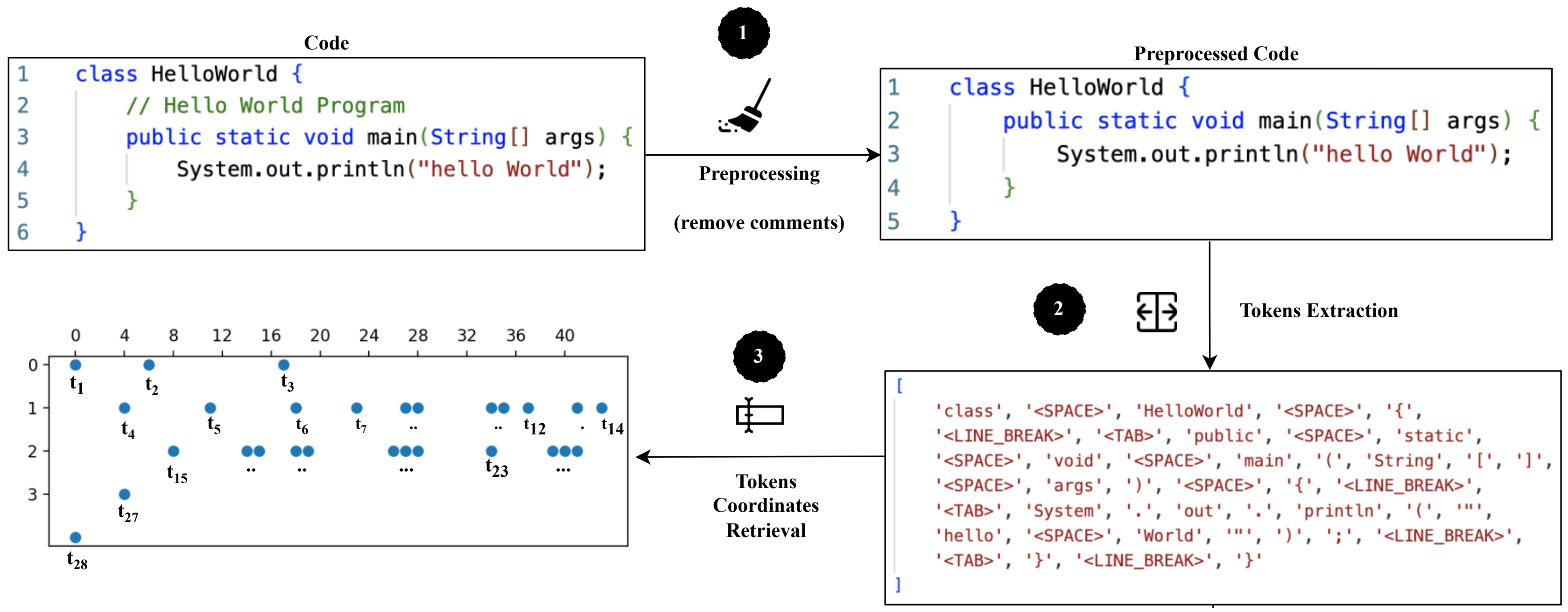
[ISSTA2023] “CodeGrid: A Grid Representation of Code”



Tokens extraction

- All code elements, including whitespaces, tabulations and line breaks

→ Preserving code spatiality

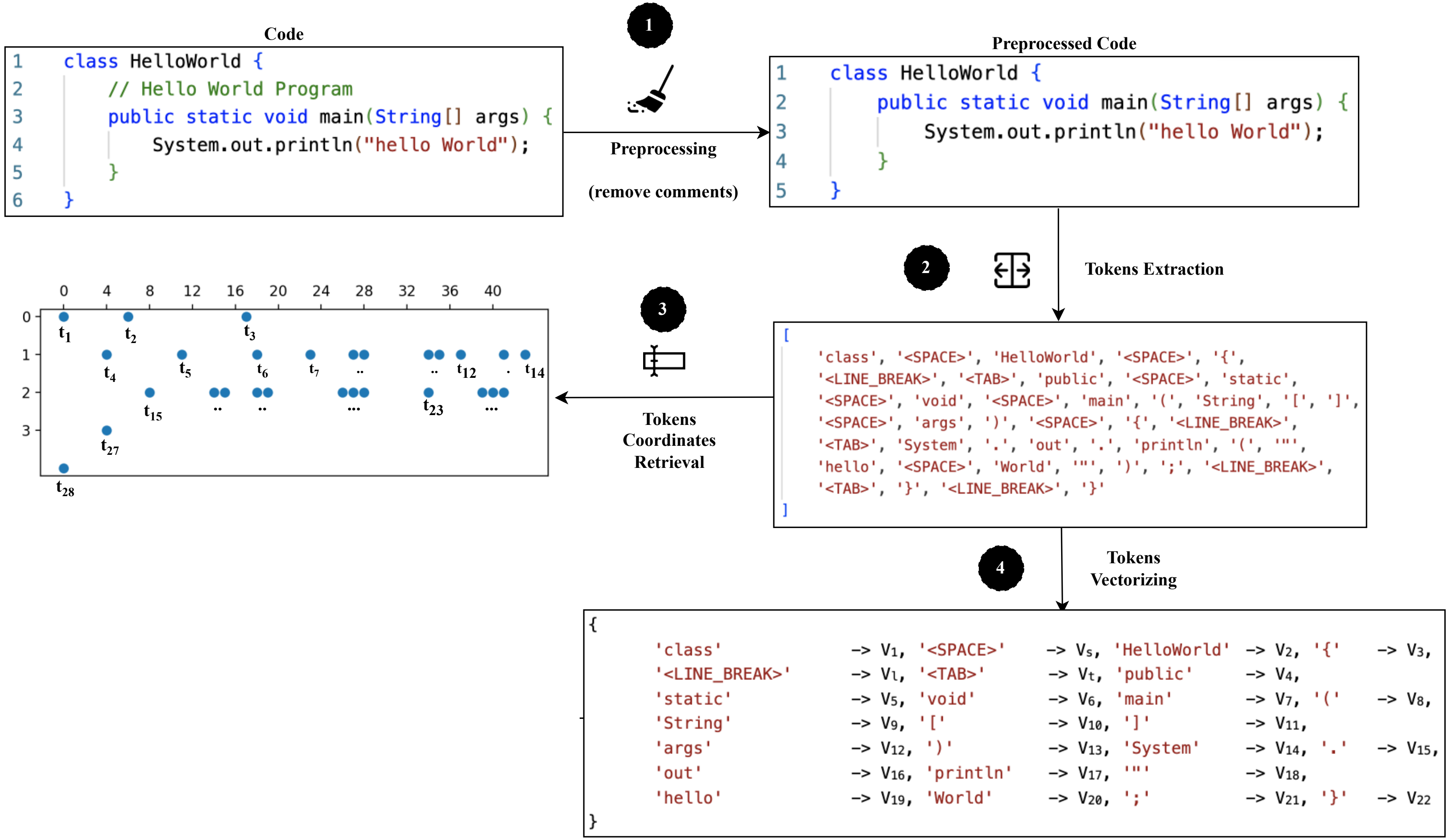


Tokens coordinates retrieval

- Place in a 2D reference the location of each token
  - Y: Line number
  - X: Location of the token's first caracter in the line

$\rightarrow$ if $x_{t1} = 0$, $x_{t2} = x_{t1} + len(t1)$

[ISSTA2023] "CodeGrid: A Grid Representation of Code"



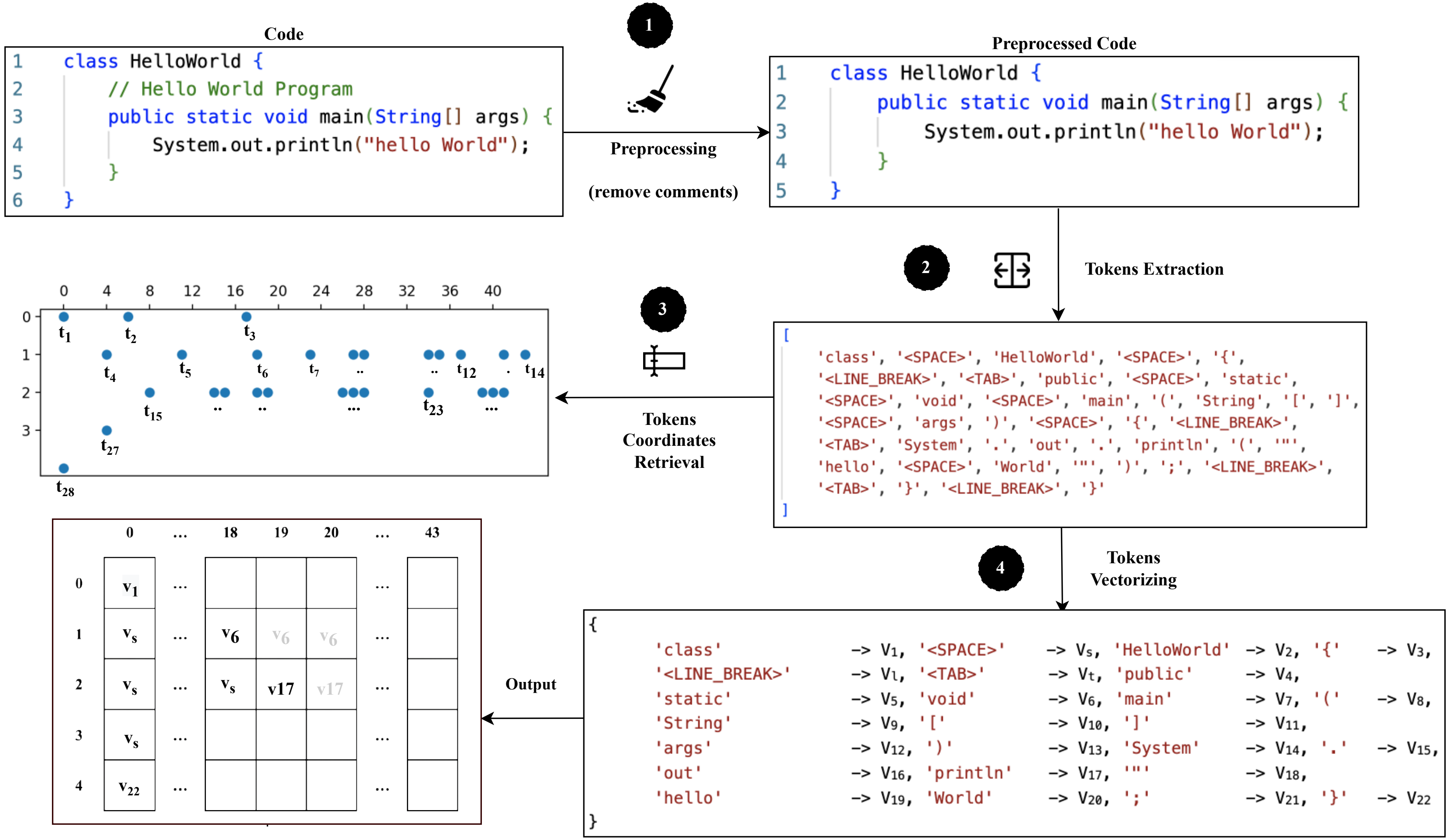
[ISSTA2023] "CodeGrid: A Grid Representation of Code"

CODEGRID: Three Tokens Vectorizing Methods

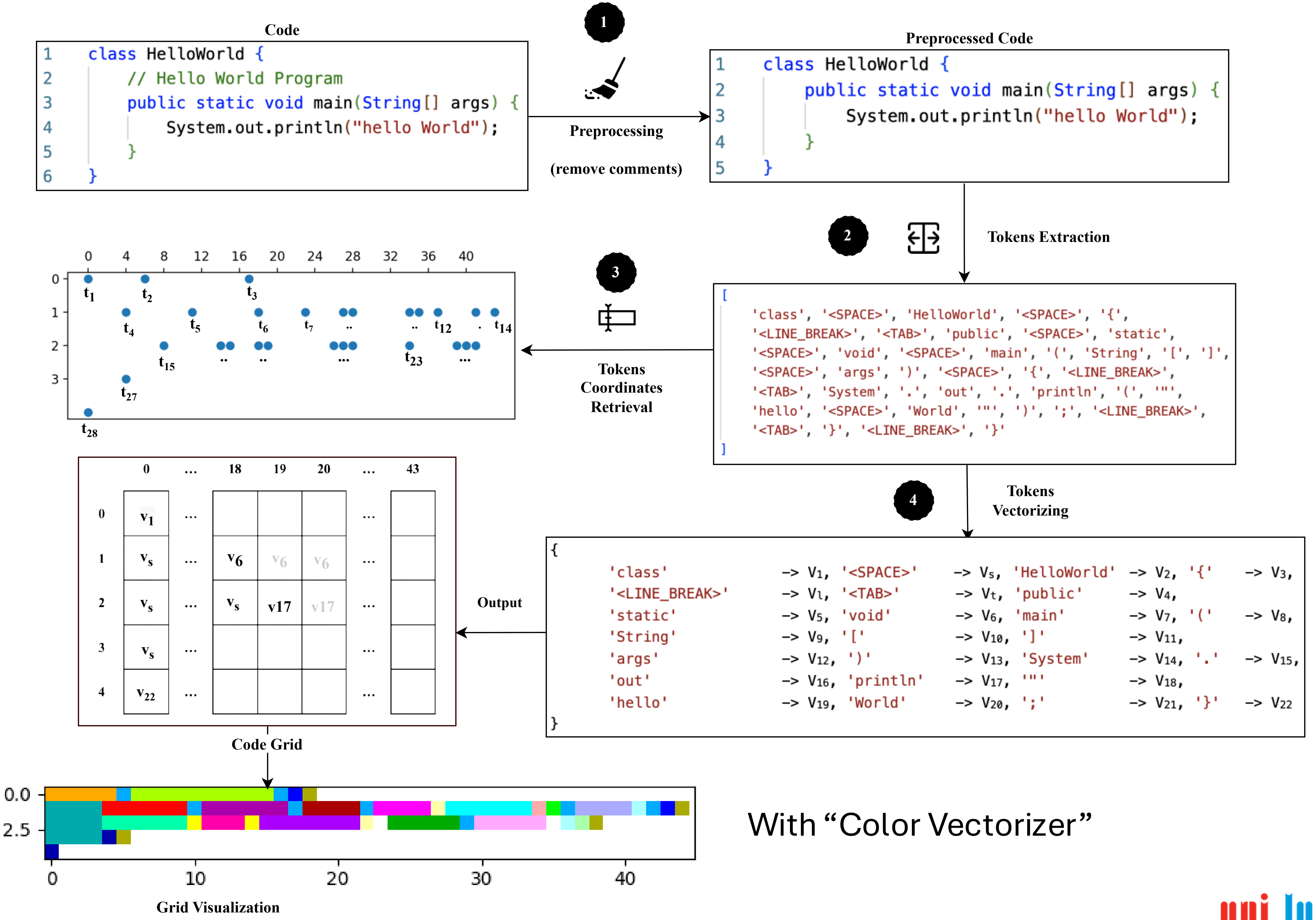
- Color Vectorizer
  - Rely on TF-IDF[1] to map each token with a color
- Word2Vec Vectorizer
- Code2Vec Vectorizer
  - Reuse of a Code2Vec[2] pretrained model

[1] Term Frequency–Inverse Document frequency; measures the relevance of a token
[2] Code2Vec is a NN model that capture the semantic meanings of code tokens

CODEGRID: REPRESENTING CODE AS GRIDS



CODEGRID: REPRESENTING CODE AS GRIDS



With "Color Vectorizer"

[ISSTA2023] "CodeGrid: A Grid Representation of Code"

SnT

Part II-C
Vulnerability Prediction with WYSiWiM and CodeGRID

UNIVERSITY OF LUXEMBOURG

Experimental Setup

- Dataset
  - Labelled samples (vulnerable or non-vulnerable)

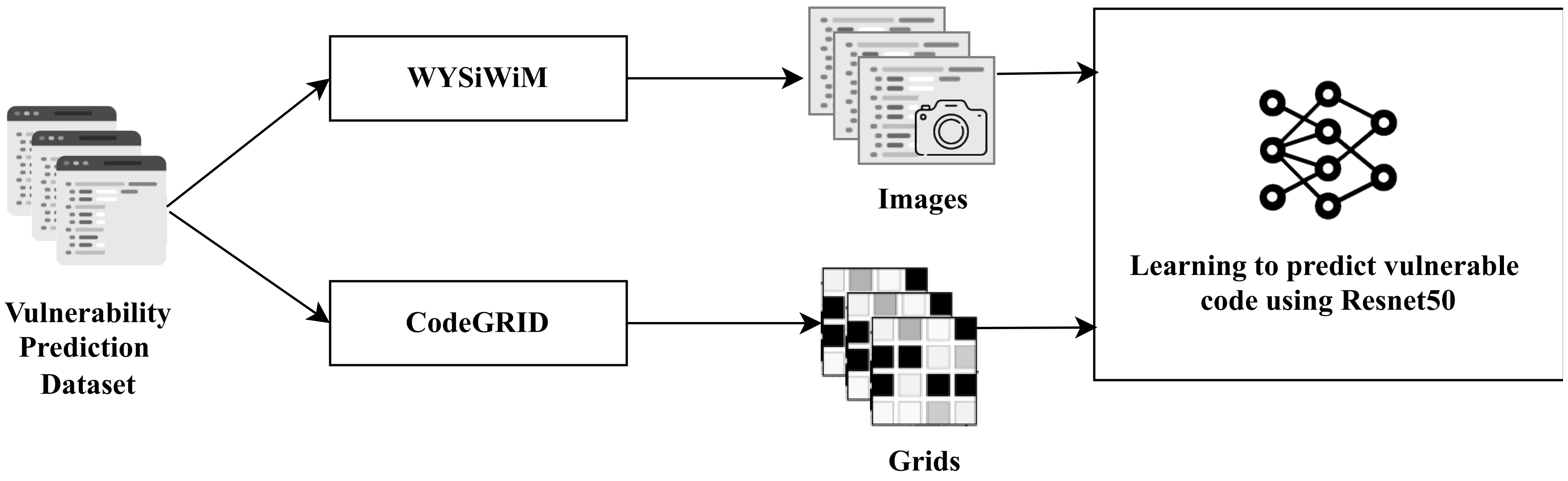
Dataset	# of samples	
	Total	Used in Testing
The KB Project[1]	1,240	248
SySeVR[2] dataset (based on NVD and SARD data)	420,627	84,126

[1] Collaborative knowledge database of vulnerabilities affecting open-source software

[2] Dataset by Zhen et al (2018)

Experimental Setup

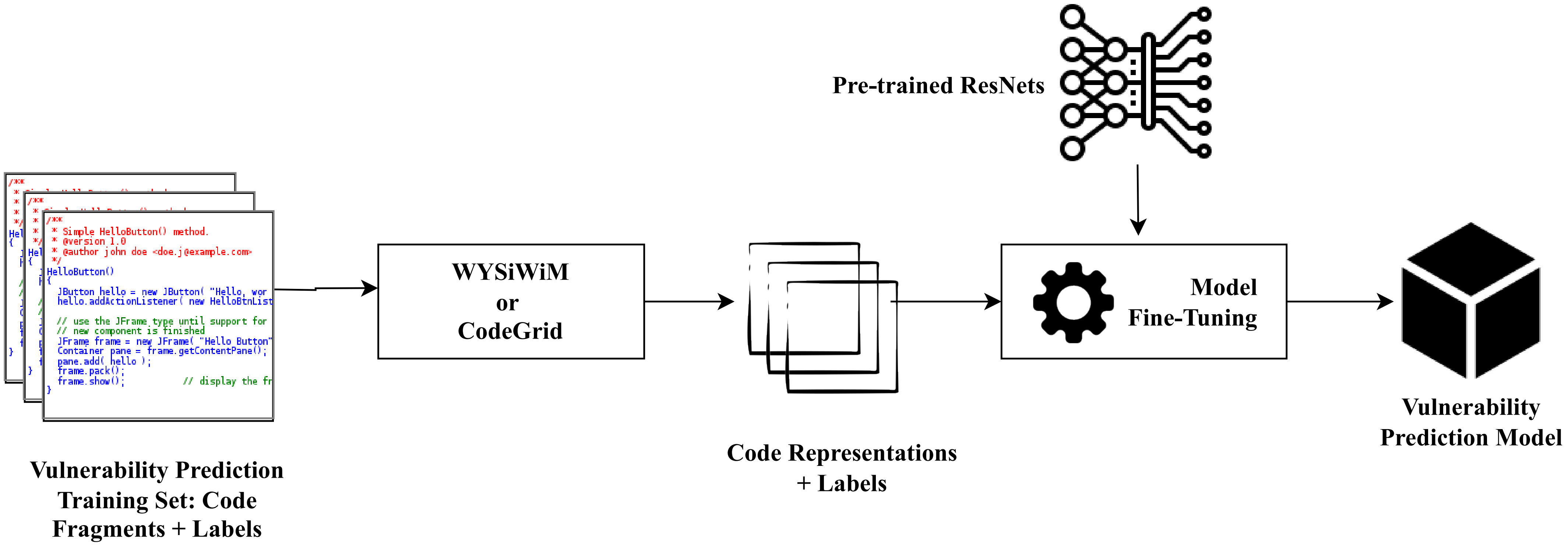
- Learning to predict vulnerable code snippets



Resnet is a CNN architecture characterized by residual connections that allow training much deeper neural networks by addressing the vanishing gradient problem. (Kaiming He et al. 2015)

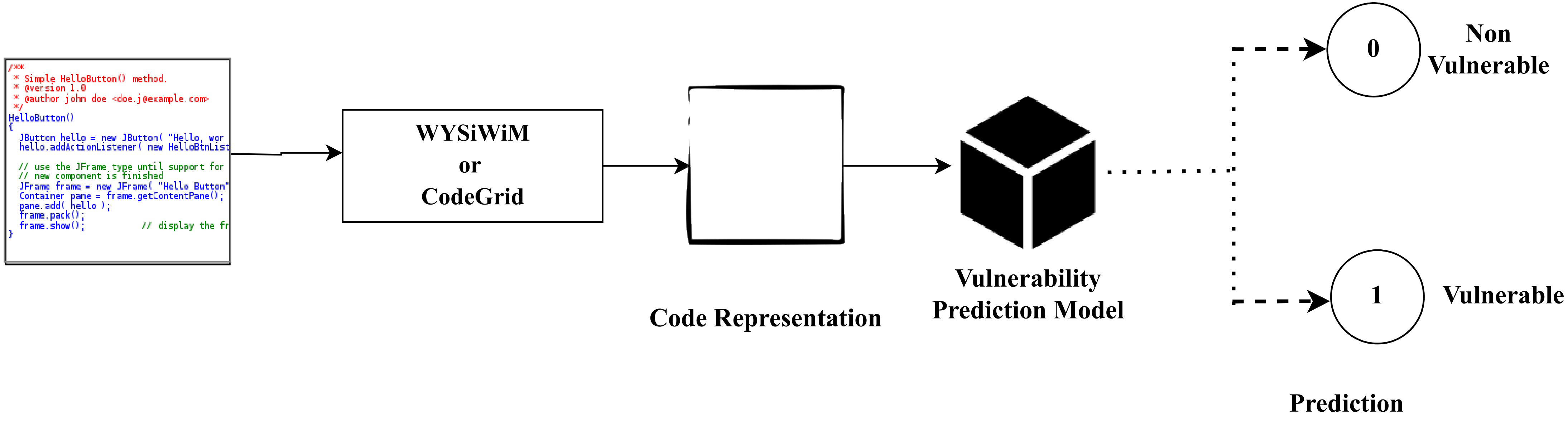
Experimental Setup

- Learning to predict vulnerable code snippets: Model training



Experimental Setup

- Learning to predict vulnerable code snippets: Inference/Testing



Experimental Results

- Performance

Approach	Variants	Accuracy	Precision	F1 score
WYSiWiM	PLAIN TEXT	88.8	88.8	88.8
	COLOR Syntax Highlighting	90.9	90.9	90.9
	GEOMETRIC syntax highlighting	62.1	62.2	62.0

WYSiWiM + "Color Syntax Highlighting" outperforms the other visualization methods.

Experimental Results

- Performance

Approach	Variants	Accuracy	Precision	F1 score
CODEGRID	Word2Vec	96.2	93.8	90.7
	Code2Vec	98.4	94.9	92.9
	Color	93.8	90.7	92.2

CODEGRID + “Code2Vec” outperforms the other variants.

Experimental Results

- Performance

Approach	Variants	Accuracy	Precision	F1 score
WYSiWiM	PLAIN TEXT	88.8	88.8	88.8
	COLOR Syntax Highlighting	90.9	90.9	90.9
	GEOMETRIC syntax highlighting	62.1	62.2	62.0
CODEGRID	Word2Vec	96.2	93.8	90.7
	Code2Vec	98.4	94.9	92.9
	Color	93.8	90.7	92.2
SySeVR[1]	-	98.0	90.8	92.6
Checkmarx[2]		72.9	30.9	36.1

CODEGRID + "Code2Vec" outperforms the SySeVR and Checkmarx

[1] A Framework for Using Deep Learning to Detect Software Vulnerabilities (Zhen et al.)
[2] Checkmarx: a commercial tool

Summary

- Code's layout is a strong signal.

Summary

- Code's layout is a strong signal.
- WYSiWiM
    - Rely on simple "screenshot"
    - Achieve near SOTA performances in vulnerability prediction with Resnet50
    - *Accepted at ACM Transactions on Software Engineering and Methodology (TOSEM), 2021*

Summary

- Code's layout is a strong signal.
- WYSiWiM
- CODEGRID
  - More rational exploitation of code spatiality
  - Complements existing code representations (CodeGRID + Code2Vec)
  - Outperforms SySeVR and Checkmarx in vulnerability prediction
  - *Accepted at the 32nd ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), 2023*

Ongoing Works

Just-in-Time Detection of Silent Security Patches

- This paper is about patch representation.
- **Key idea:** leverage large language models (LLMs) to augment patch information with generated code change explanations

T

H

A

N

K

S

Malware Detection

The need for a large set of Apps and a ground truth

Performance Assessment Issues

App Code Representation

An app as a Image

BERT-Based class representation

Full App-level representation

Vulnerability Detection

Code is Spatial

WYSiWiM: Representing code as images

CodeGRID: Representing code as grids

Vulnerability Prediction with WYSiWiM and CodeGRID