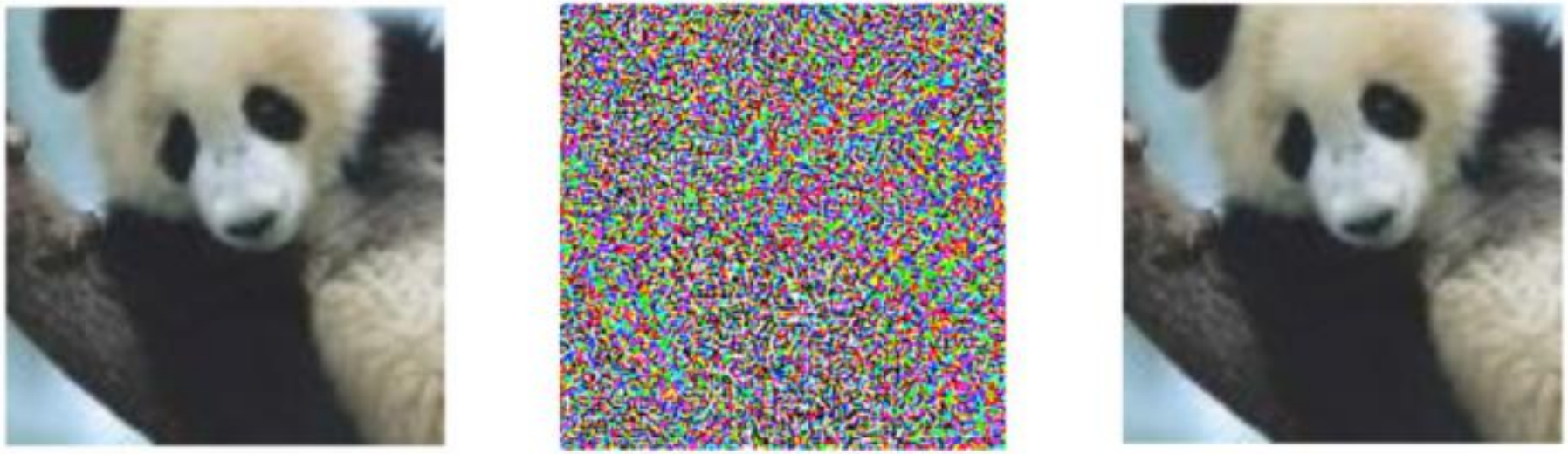
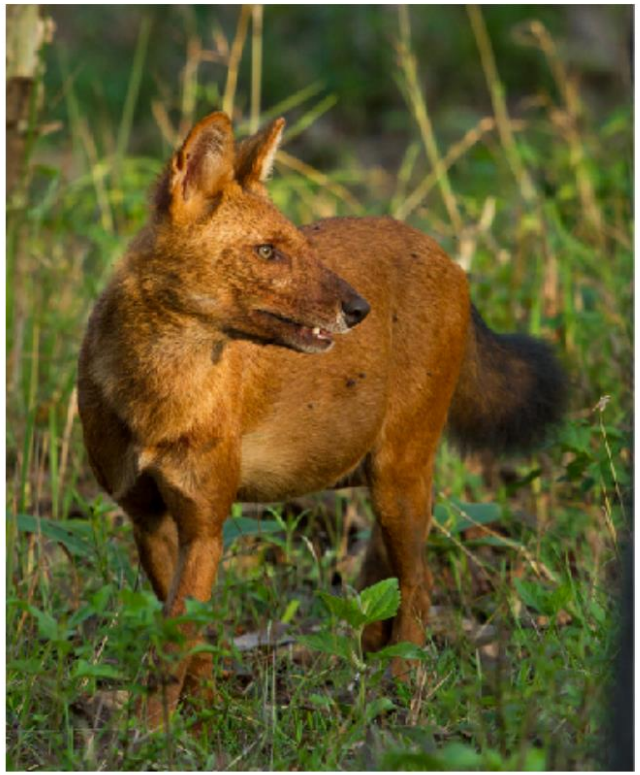
Adversarial Machine Learning

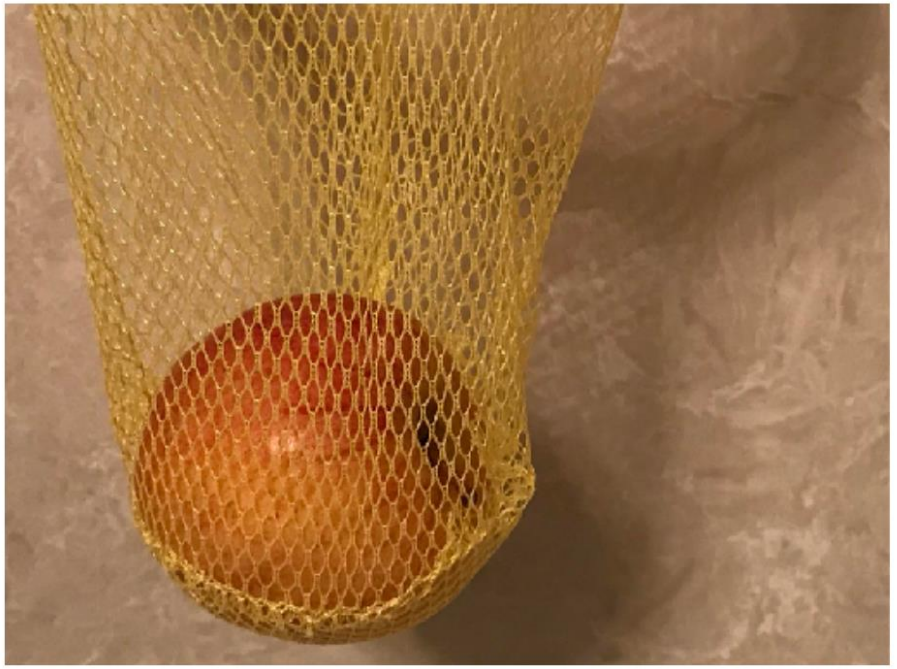


Example from: Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. ICLR 2015.

Adversarial Machine Learning



Humans are not very good at some parts of the benchmark



The test data is not very diverse. ML models are fooled by natural but unusual data.

(Goodfellow 2018)

Adversarial Machine Learning

(Eykholt et al, 2017)

Issue?

Research problem but not realistic problems.

Understanding Machine Learning in Security Settings

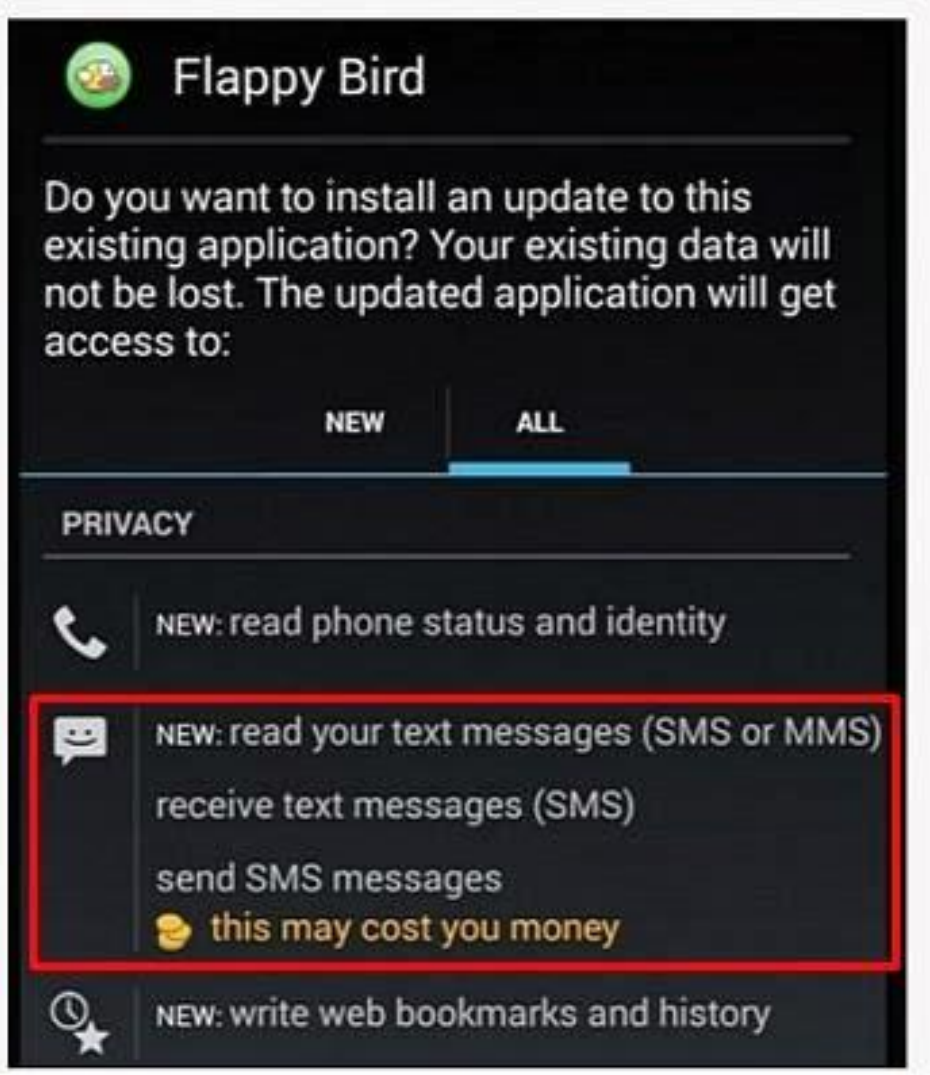
- AppContext
  - Why security needs machine learning?
- MRV
  - What type of vulnerabilities may exist in ML-based security systems?
- Permutation Invariance
  - How to use ML to attack ML?

AppContext: Differentiating Malicious and Benign Mobile App Behaviors Using Context

Wei Yang, Xusheng Xiao, Benjamin Andow, Sihan Li, Tao Xie and William Enck

ICSE 2015

Example - A malicious app



I'm being charged for unwanted premium rate text messages

Are you paying for texts you don't want or didn't ask for?

It pays to read the small print before you sign up to a text service so you know exactly what it will cost.

The regulator for premium rate phone services - PhonepayPlus - handled almost 16,000 complaints in the year to 2014.

Of these, 80% related to SMS messages, with over 8,000 leading to enforcement action.

http://www.which.co.uk/consumer-rights/problem/im-being-charged-for-unwanted-premium-rate-text-messages

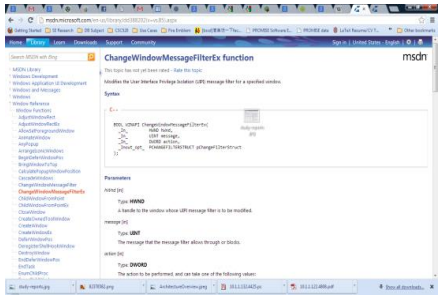
Checking security-sensitive behaviors

Permission List

Do you want to install this application?

Allow this application to:

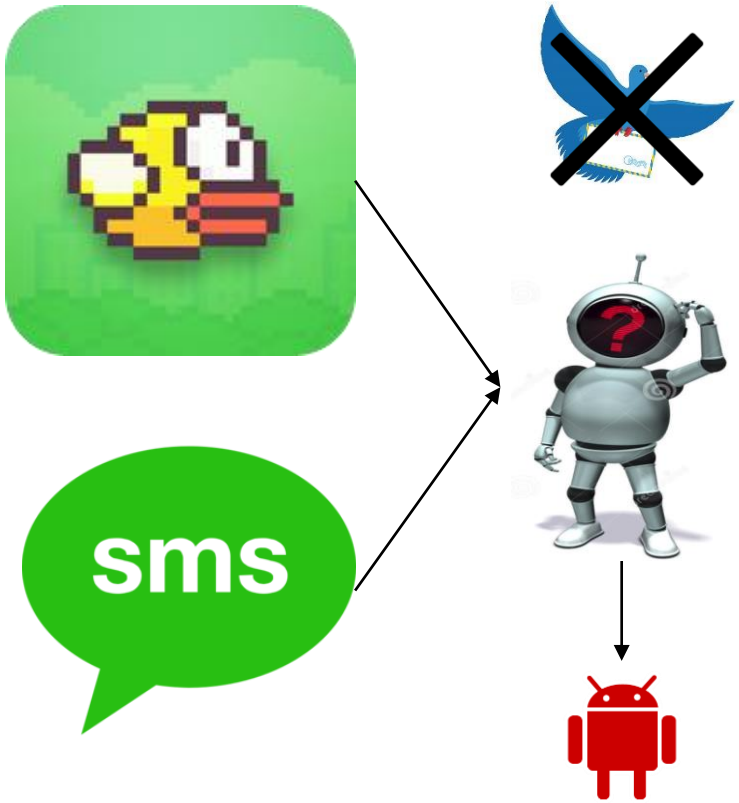
- **Your location**
  coarse (network-based) location, fine (GPS) location
- **Network communication**
  full Internet access
- **Storage**
  modify/delete SD card contents
- **Services that cost you money**
  directly call phone numbers
- **Phone calls**
  read phone state and identity



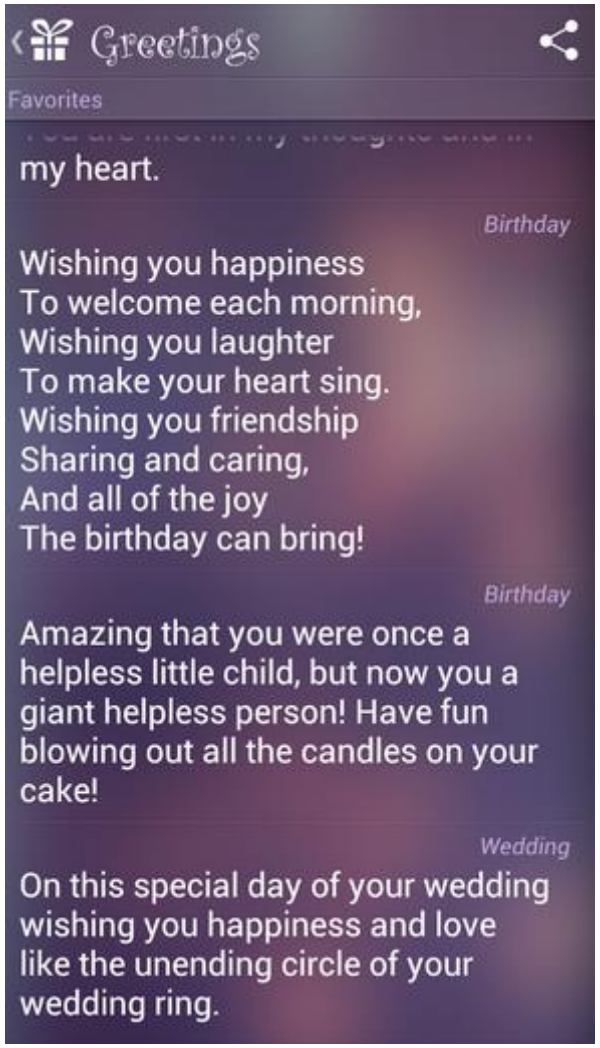
API Documents

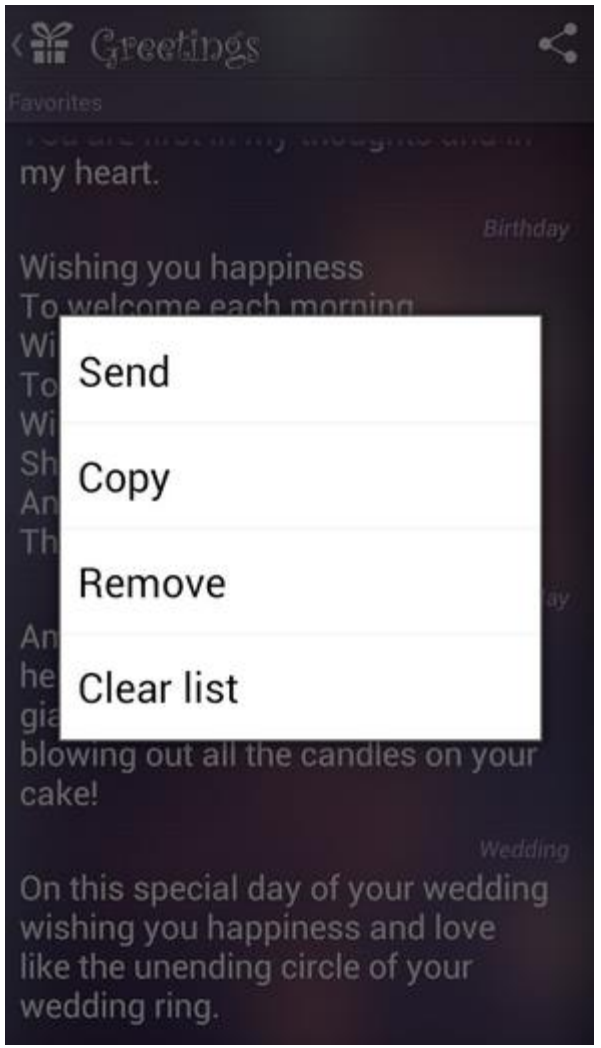
SmsManager.sendTextMessage()

Sensitive APIs

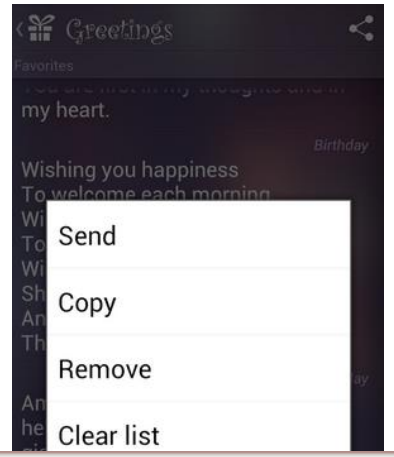


A benign app —— Greetings





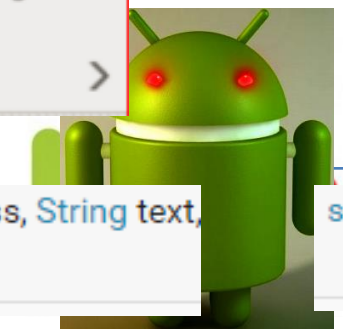
An adaptive adversary



Permissions

Your messages
Edit your text messages (SMS or MMS), read your text messages (SMS or MMS), receive text messages (SMS), send SMS messages

Your messages
Edit your text messages (SMS or MMS), read your text messages (SMS or MMS), receive text messages (SMS), send SMS messages



Google play

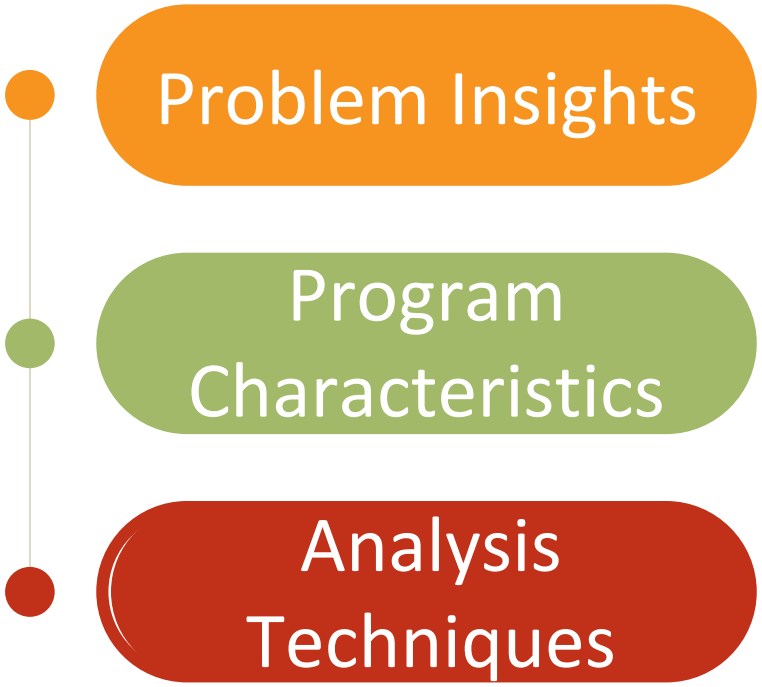
sendTextMessage (String destinationAddress, String scAddress, String text,
Send a text based SMS.

sendTextMessa ess, String scAddress, String text,
Send a text based SMS.

Sensitive API methods

My research methodology

- What intrinsic property makes malware and benign apps different?



- What intrinsic property makes malware and benign apps different?
- What is the representation of such difference in the mobile programs?
- How to automatically extract such representation from mobile programs?

Our Insight

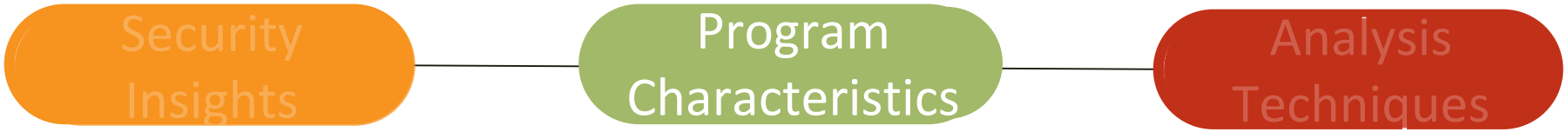
Different inherent goals of benign apps vs. malware as differentiating factors

- Benign apps
  - Meet requirements from users (as delivering utility)
- Malware
  - Trigger malicious behaviors frequently (as maximizing profits)
  - Evade detection (as prolonging lifetime)

Differentiating characteristics

Mobile malware (vs. benign apps)

- **Frequently enough** to meet the need: **frequent** occurrences of **imperceptible** system events;
  - E.g., many malware families trigger malicious behaviors via background events.



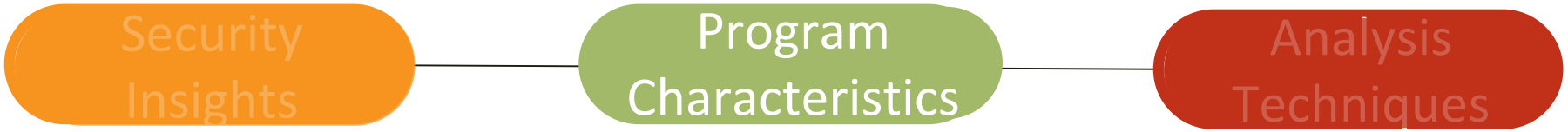
Differentiating characteristics

Mobile malware (vs. benign apps)

- **Frequently enough** to meet the need: **frequent** occurrences of **imperceptible** system events;
  - E.g., many malware families trigger malicious behaviors via background events.

Balance!!!

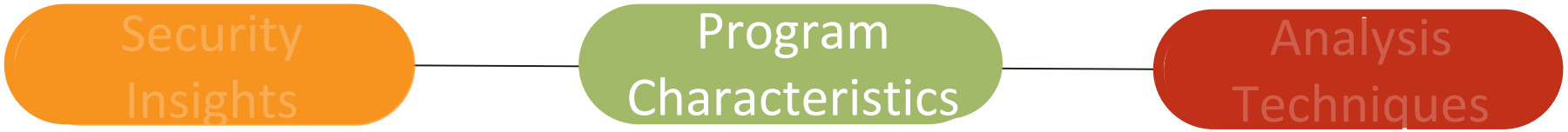
- **Not too frequently** for users to notice anomaly: **indicative** states of external environments
  - E.g., Send premium SMS every 12 hours



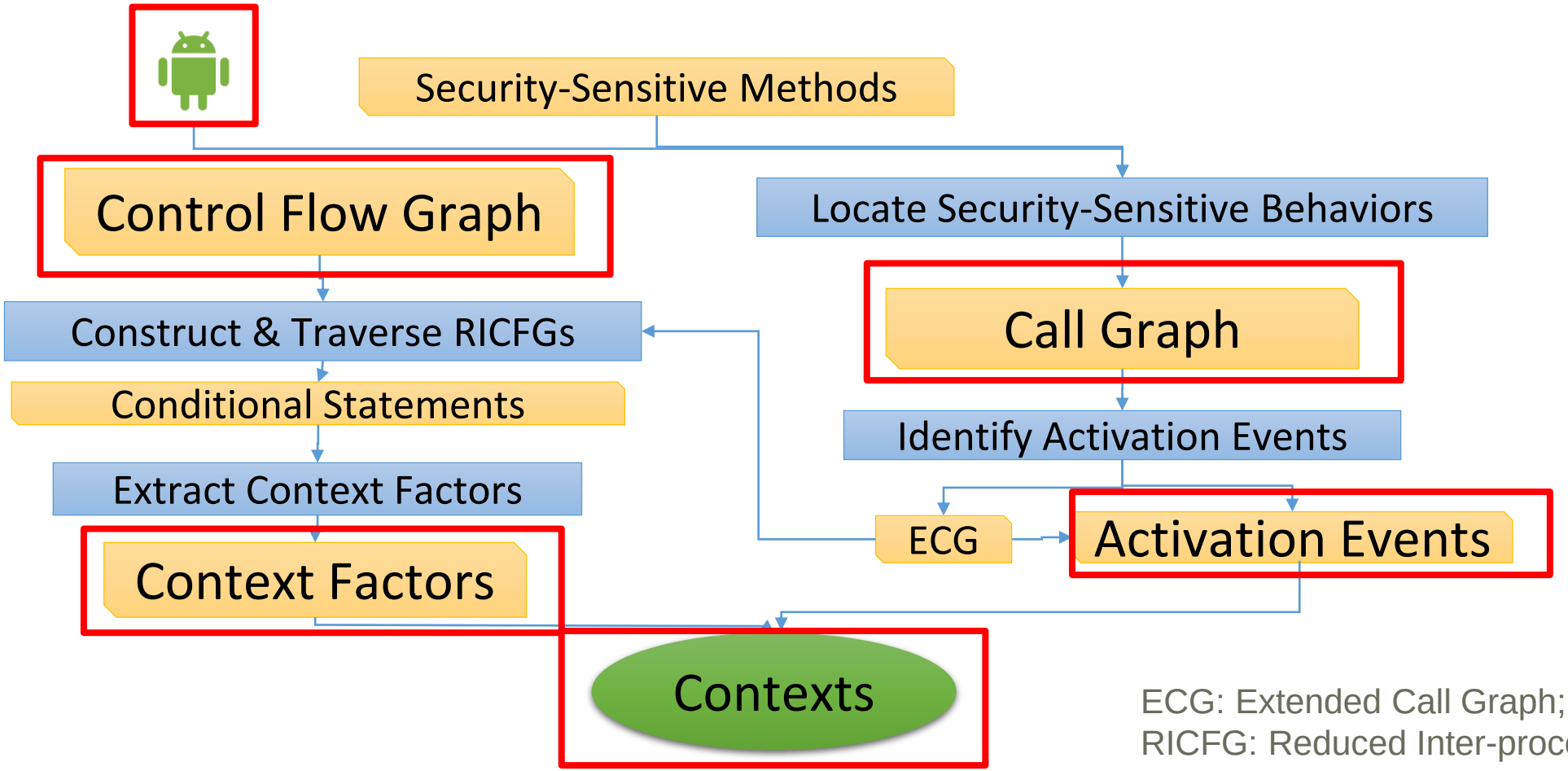
Differentiating characteristics

Mobile malware (vs. benign apps)

- **Frequently enough** to meet the need: **frequent** occurrences of **imperceptible** system events;
  - Activation events, e.g., signal change
  - E.g., many malware families trigger malicious behaviors via background events.
- **Not too frequently** for users to notice anomaly: **indicative** states of external environments
  - Context factors, e.g., current system time
  - E.g., Send premium SMS every 12 hours



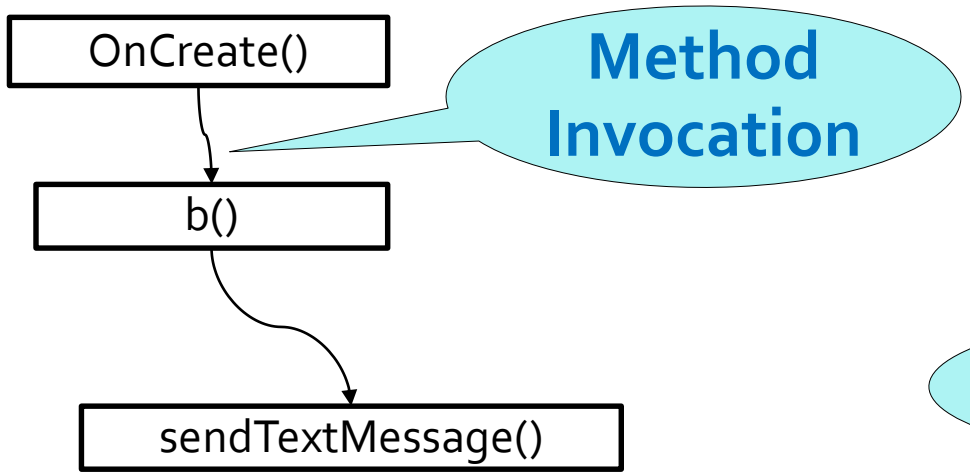
How to extract contexts automatically?



ECG: Extended Call Graph;
RICFG: Reduced Inter-procedure Control Flow Graph

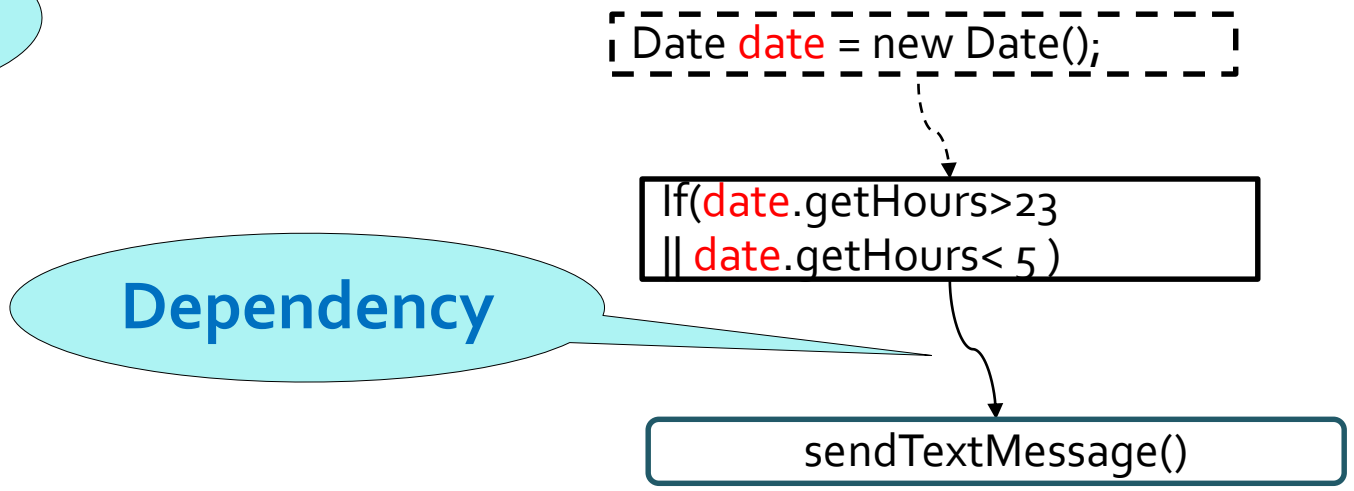
Security Insights — Program Characteristics — Analysis Techniques

How to extract contexts automatically?



Call Graph

Representing triggering relationship

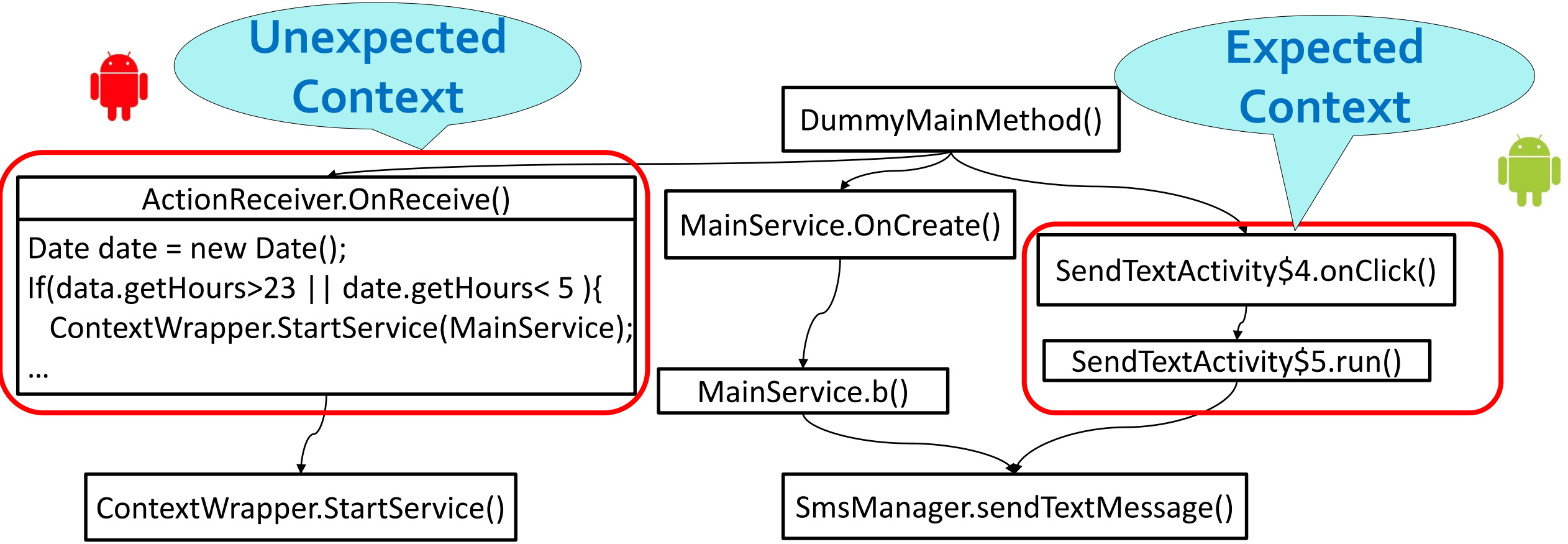


Control-flow Graph

Representing controlling relationship

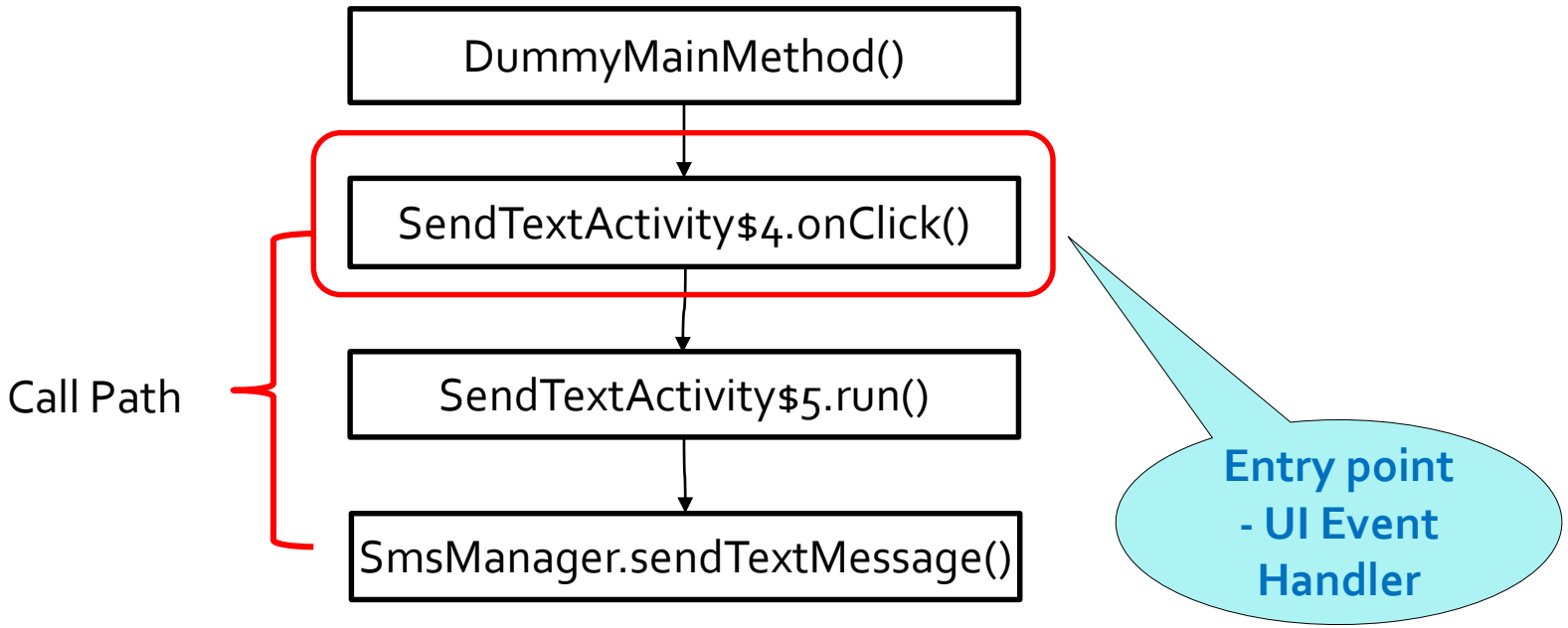
Security Insights — Program Characteristics — Analysis Techniques

Example



Security Insights — Program Characteristics — Analysis Techniques

Expected Context

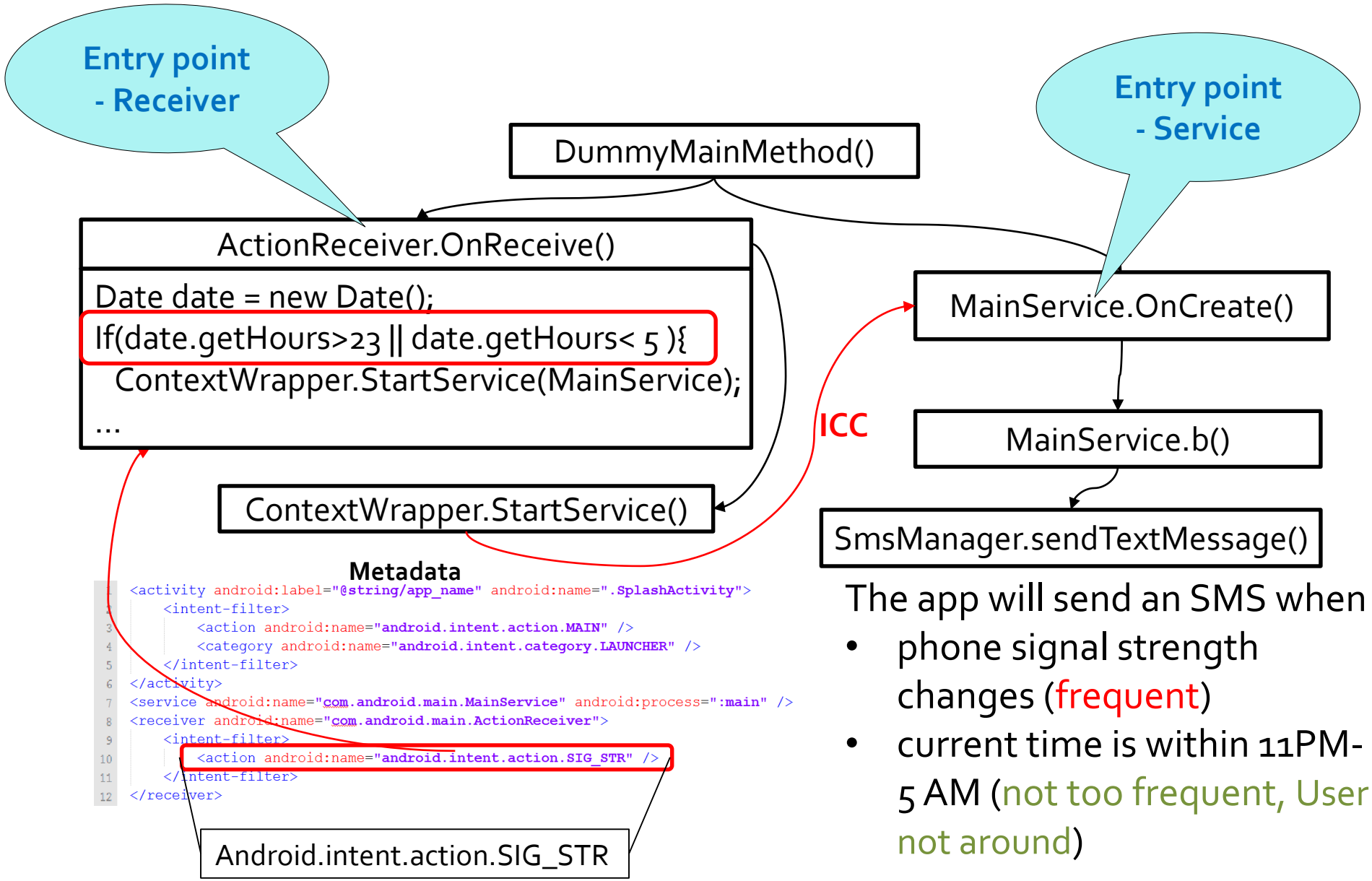


The app will send an SMS when

- user clicks a button in the app

Security Insights — Program Characteristics — Analysis Techniques

Unexpected Context



The app will send an SMS when

- phone signal strength changes (frequent)
- current time is within 11PM-5 AM (not too frequent, User not around)

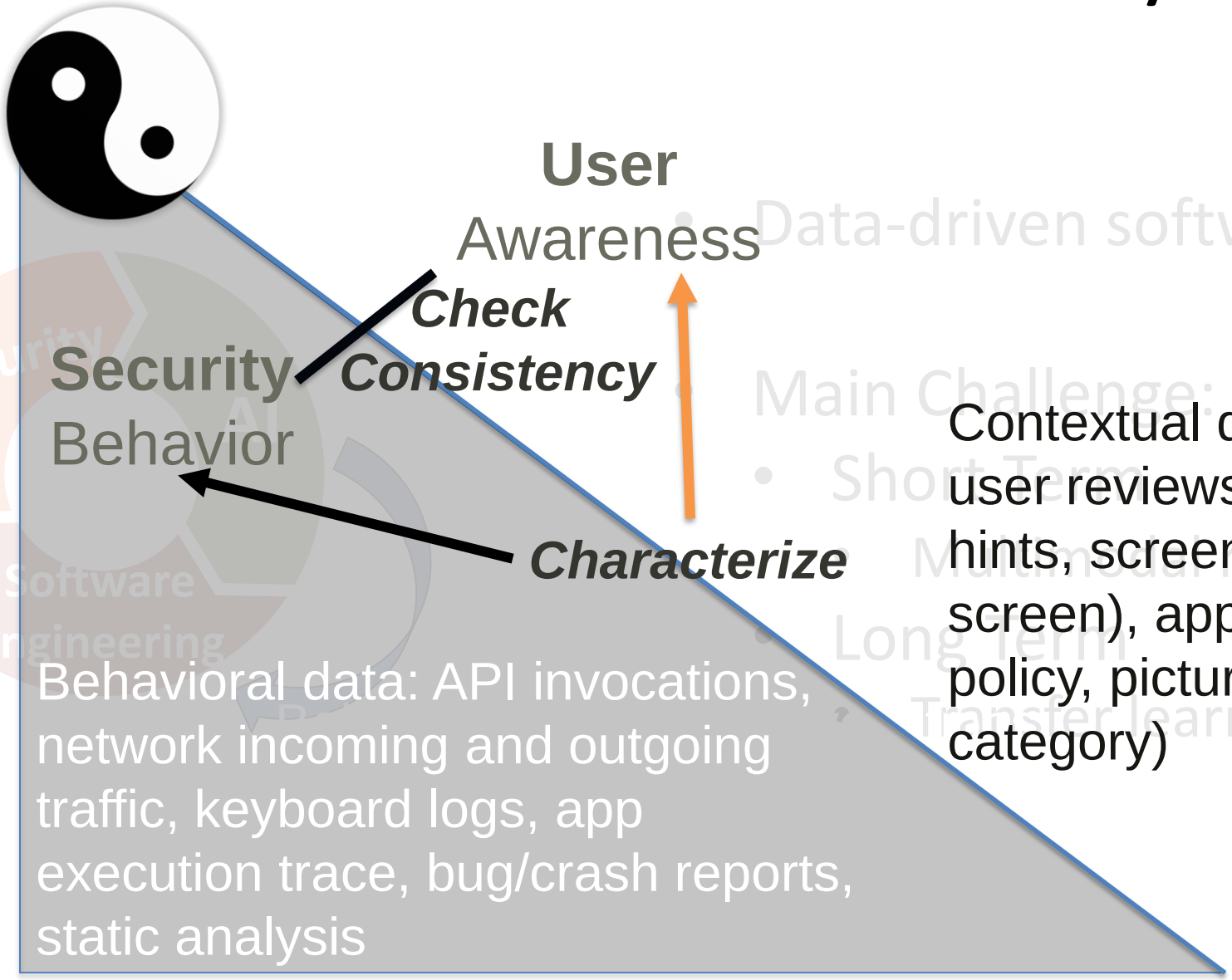
Why security needs machine learning?

The app will send an SMS when
- user clicks a button in the app

The app will send an SMS when
- phone signal strength changes (frequent)
- current time is within 11PM-5 AM (not too frequent, User not around)

How to determine which context is malicious?

User Awareness vs. Security Behavior



Contextual data: user usage data, user reviews, UI screen (labels, hints, screen, buttons, sequence of screen), app descriptions, privacy policy, pictures/videos, tags (app category)

Context-based Security-Behavior Classification

Context1:
(Event: Signal strength changes),
(Factor: Calendar)

Context2:
(Event: Clicking a button)

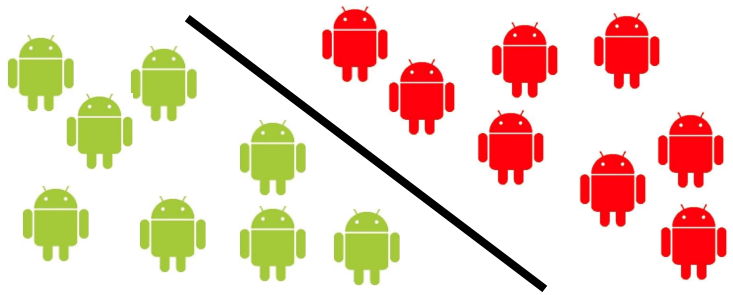
Existing Features

Permission	Method	...	Hardware	System	UI	F_1	F_2	F_3	F_4	F_5	...
SEND_SMS	SendTextMessage	...	N/A	SIG_STR	N/A	0	0	1	0	0	...
SEND_SMS	SendTextMessage	...	N/A	N/A	Click	0	0	0	0	0	...

F_3 = Calendar

Support Vector Machine (SVM)

- Resilient to over-fitting
- Effective for high dimension data

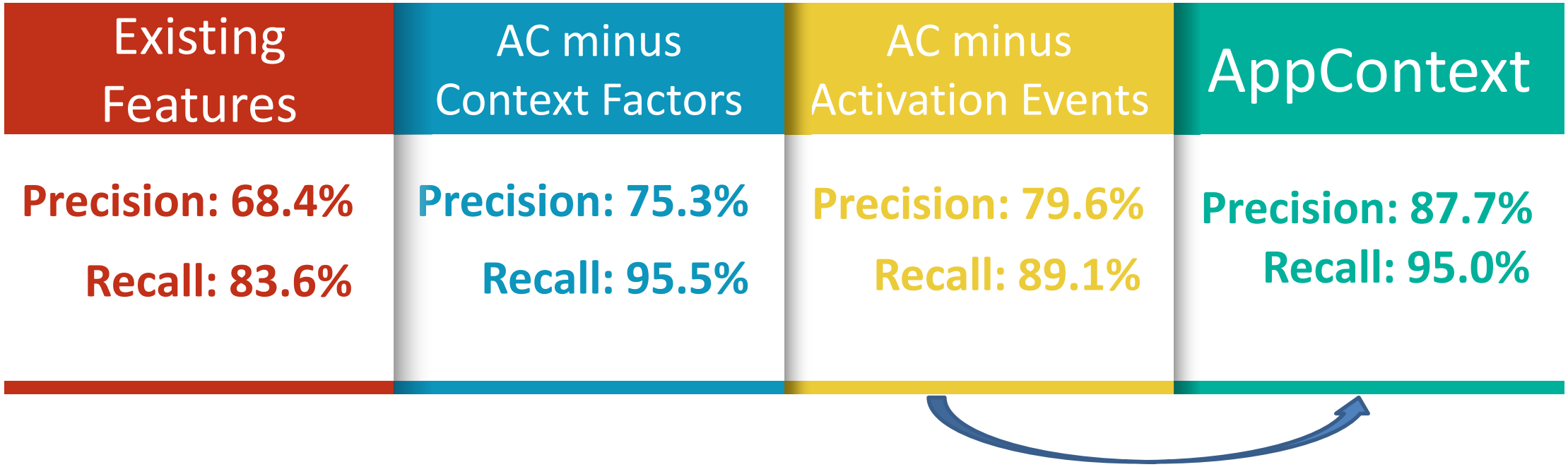


Security Insights — Program Characteristics — Analysis Techniques

Evaluation

Existing Features	AC minus Context Factors	AC minus Activation Events	AppContext
Precision: 68.4%	**Precision: 75.3%**	**Precision: 79.6%**	**Precision: 87.7%**
Recall: 83.6%	**Recall: 95.5%**	**Recall: 89.1%**	**Recall: 95.0%**

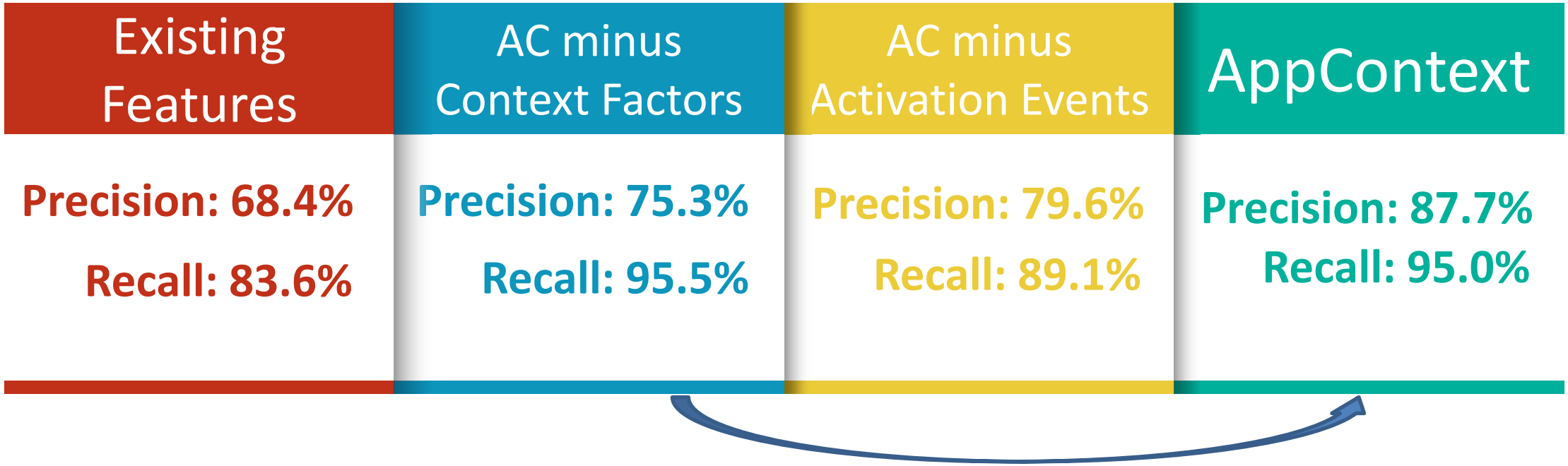
Evaluation



Activation events effectively help identify malicious method calls **without context factors**

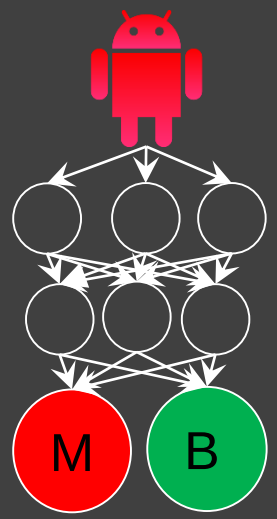
Malicious behavior (activation event) <-> Benign behavior (update the UI) **E.g.,** UMS_DISCONNECTED, SIG_STR

Evaluation



Context factors effectively help identify malicious behaviors triggered by **UI events** or malicious behaviors with **no activation events**

Why not using everything in malware bytecode as features?



Magic happens?

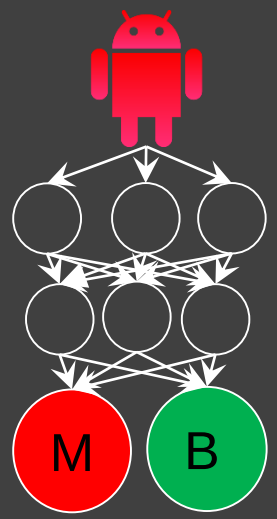
- Discriminative features are not resilient in adversarial settings.
  - Slower and need more labelled data.

Insight: Malware detectors often include non-essential features in code clones as discriminative features.

```
for (int i =0; i < n/2; i++){
        char temp = a[i];
        a[i] = a[n-1-i];
        a[n-1-i] = temp;
} //reverse the SMS message
……
for (int i =0; i < n/2; i++){
        char temp = a[i];
        a[i] = a[n-1-i];
        a[n-1-i] = temp;
} //reverse the SMS message again
sendTextMessage(a);
```

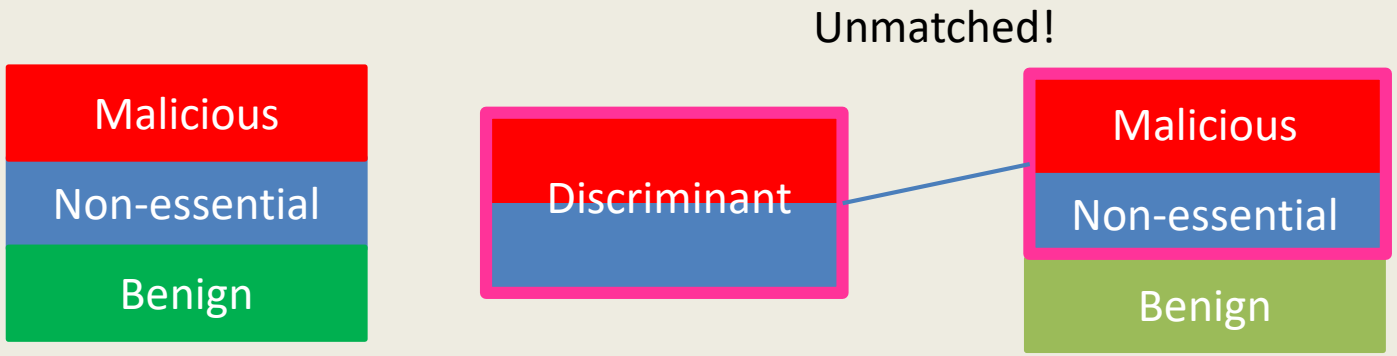
Code clones

Why not using everything in malware bytecode as features?



Magic happens?

- Discriminative features are not resilient in adversarial settings



Malware Detection in Adversarial Settings: Exploiting Feature Evolutions and Confusions in Android Apps

Wei Yang, Deguang Kong, Tao Xie and Carl A. Gunter
ACSAC 2017

Generating adversarial example helps build better classifiers

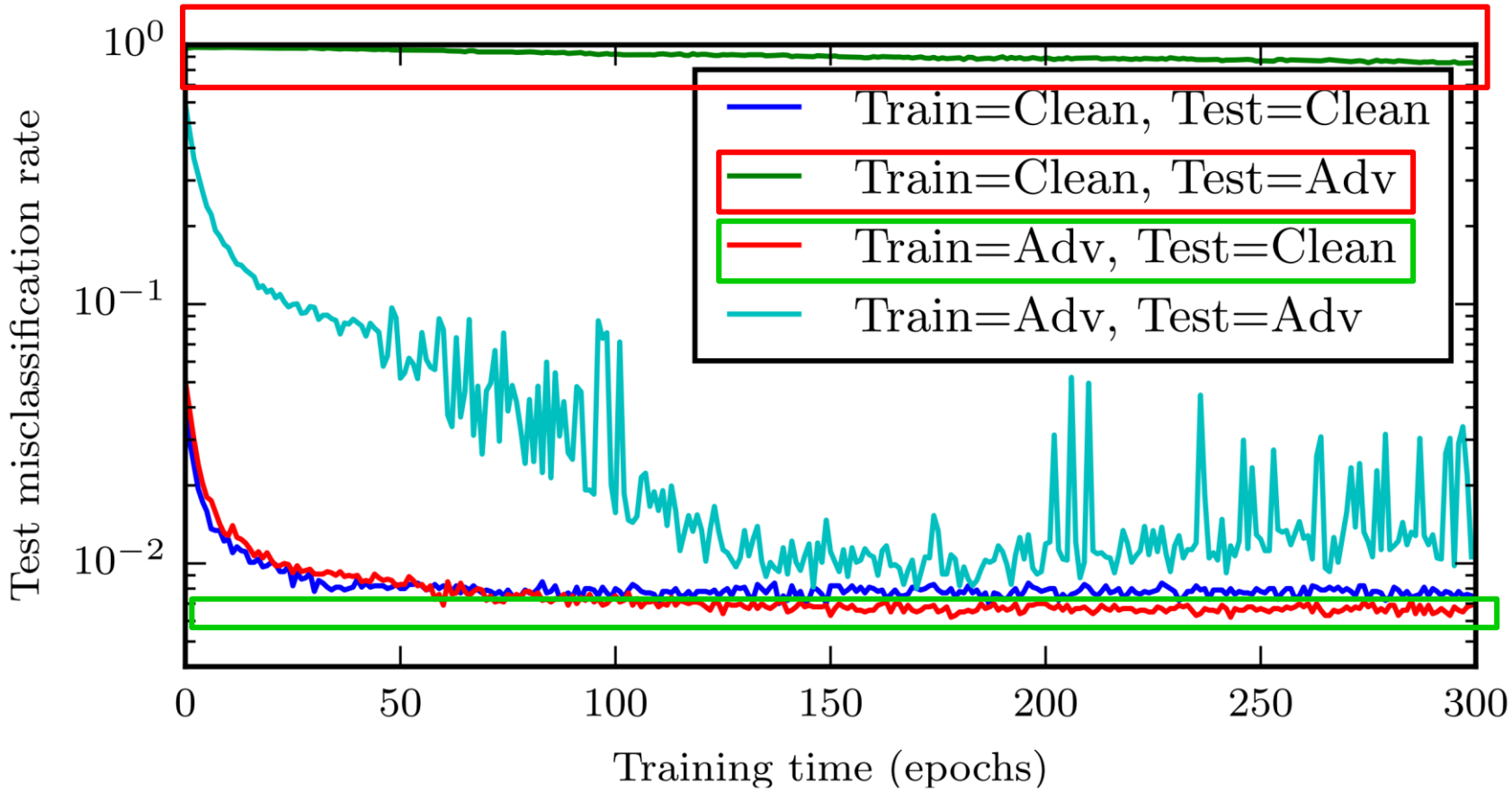


Figure Credit: GoodFellow 2016

Not all evasive samples are good adversarial testing inputs

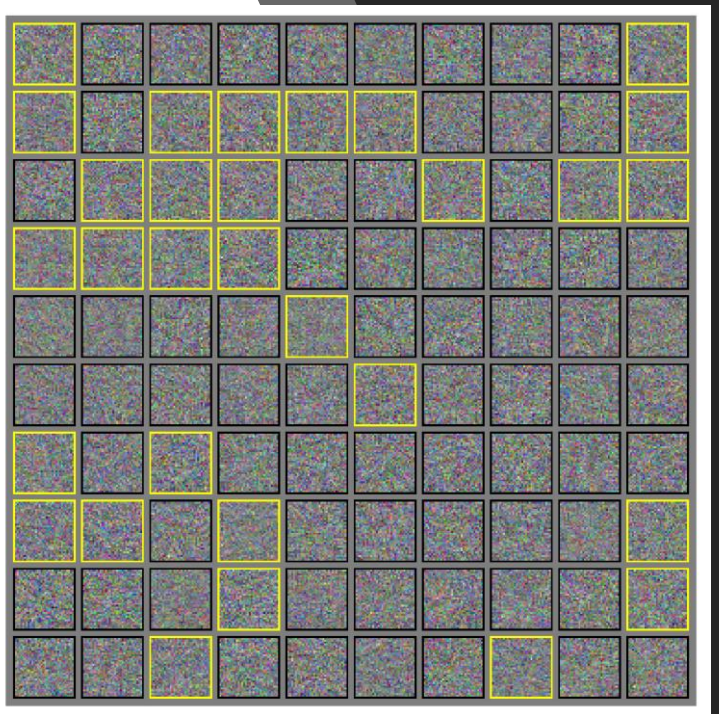


Figure Credit: GoodFellow 2016

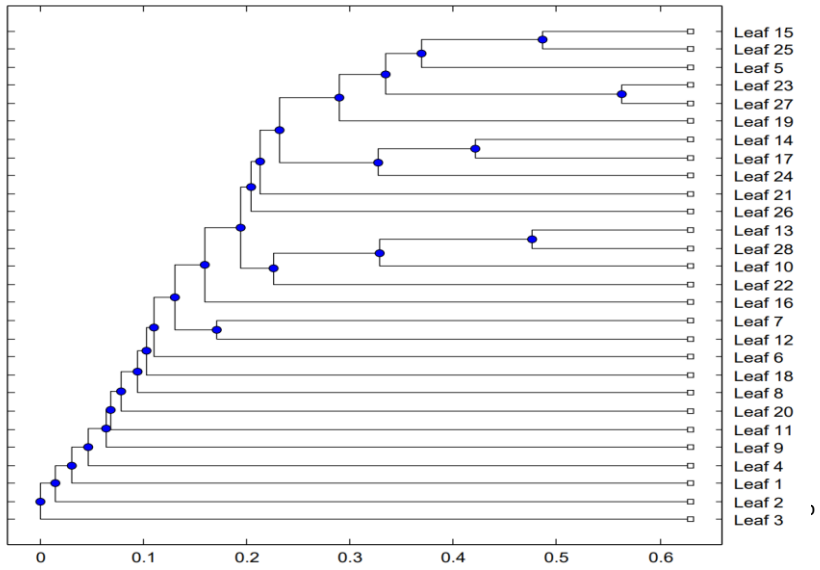
- Potential side effect
  - –crash the app
  - –cause undesirable behaviors
  - –disable malicious functionalities.
  - –the code cannot even be compiled.
- Automatically generating meaningful adversarial malware is challenging!

Three practical constraints to craft a realistic adversarial-test input

Malicious	Preserving Malicious Behaviors.
Robust	Maintaining the Robustness of Apps.
Evasive	Evading Malware Detectors.

Malware Recomposition Variation (MRV)

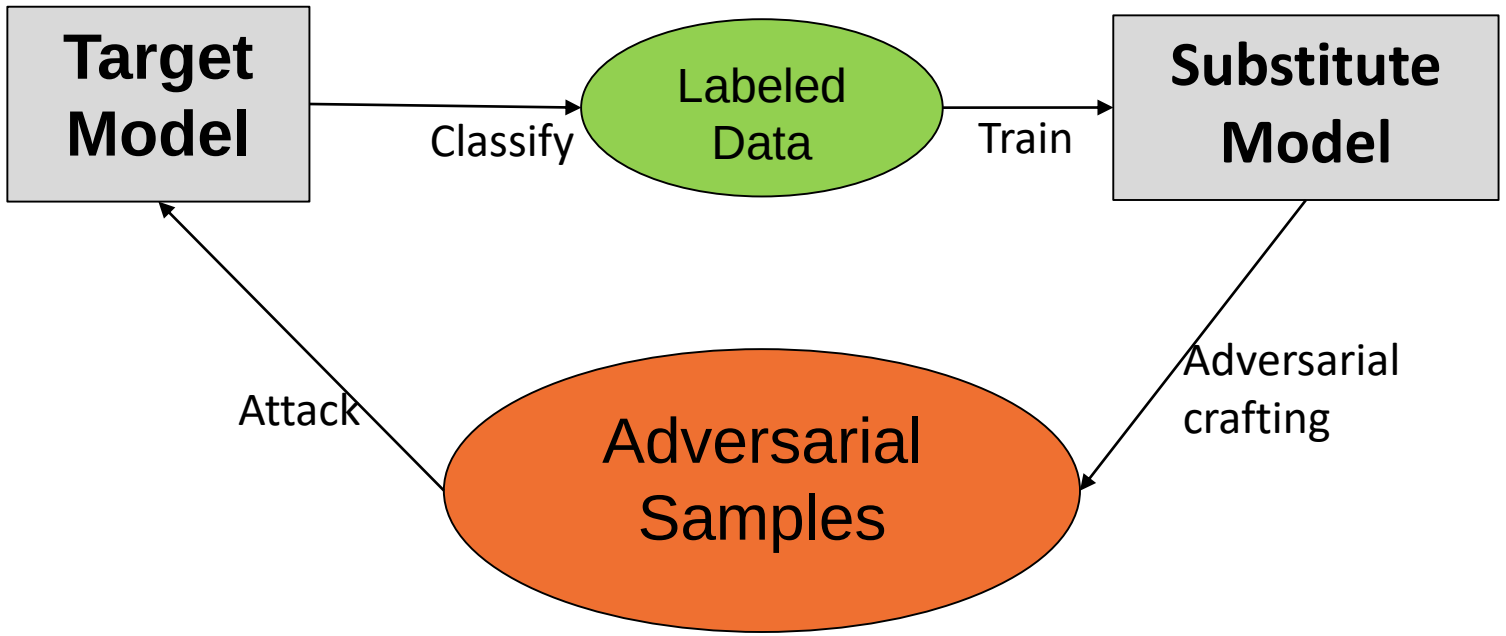
- Malware Evolution Strategy
  - Phylogenetic analysis
- Malware Confusion Strategy
  - Similarity metric
- Insight
  - Follow existing patterns!



1: 5e2c1f35ea3196a7d81b42d932729c4f253fbc8a -> DroidKungFu.AB.Gen
2: b77e28f4018dfb1e73e83a617db9f36a708ccfae -> DroidKungFu.AB.Gen
3: ba92a5bbb79dace47a76455754865d8fab9ab2cc -> DroidKungFu.AB.Gen
4: c3d37de639c0909ad78cec8c52f63f04742fbe6b -> DroidKungFu.AB.Gen
5: a997762fd1edc5971071ec574e6e8c4e -> DroidKungFu.AW
6: 1e036ab0c29dd1c8d8b95bdd2eb3400d -> DroidKungFu.AW
7: 1e7203279f153c3282eeecadbf2a9b232bc4ffda -> DroidKungFu.AW
8: f1e4a265c516b104d4e2340483fca24d60be4c08 -> DroidKungFu.AW
9: 267043ab471cf7a7d82dda6f16fa4505f719f187 -> DroidKungFu.AW
10: 683e1f3e67b43631690d0fec49cec284 -> DroidKungFu.BB
11: 517dc7b67c852392491ce20baff70ada92496e6d -> DroidKungFu.BL
12: 044f87b435c583ca4aa116aef4b54e09bdb42c7b -> DroidKungFu.BL
13: b9eae89df96d9d62ed28504ee01e868b -> DroidKungFu.C
14: 4d56a6705afa9b162f652303f6c16741 -> DroidKungFu.C
15: f30c4058f1e6cb46f81d21b263cf35454638275a -> DroidKungFu.C
16: 6d3b9cdf559443db567113585b40eef459307c94 -> DroidKungFu.G
17: 537db43883f55b112a4206fb99093bbb31c9c3f2 -> DroidKungFu.G
18: 4c1775c28def41a221e5bf872c5e593de22bb9a6 -> DroidKungFu.G
19: 92888228c556f94b8be3d8bf747a2427481f32d1 -> DroidKungFu.G
20: 0657a70a655dc439b4222c8161b1f5a9667e84e3 -> DroidKungFu.G
21: 8c841b25102569be4a1a5f407108482473fad43e -> DroidKungFu.G
22: e3f0de8ad898f0b5a7c9d327ffc266635b8af32b -> DroidKungFu.G
23: 03ddb783e7ab88c838b1888b38eba992 -> DroidKungFu.M.Gen
24: 00b89bcb196a138f9f20a6853c41673a18a2575f -> DroidKungFu.M.Gen
25: 584f8a2801a8e7590dc466a6ebc58ea01a2d1758 -> DroidKungFu.M.Gen
26: 8edb188204c6ad79006e555b50e63705b68ea65d -> DroidKungFu.M.Gen
27: bd249c0843df2f8acfe7615feba505ead30e5bbc -> DroidKungFu.M.Gen
28: 2cfa26bb22bbdc4e310728736328bde16a69d6b4 -> DroidKungFu.M.Gen

Adversarial testing in black-box scenarios

- Transferability property



RTLD Model:

Resource Temporal Locale Dependency

Example – RTLD features

```
public class User extends Application{
        public String androidid;
        public String tel;
}

public class MainActivity extends Activity{
    public void onCreate(android.os.Bundle b){
        ……
        User u = (User) getApplication();
        u.androidid =  Settings.Secure.getString(getContentResolver(),
"android_id");
        u.tel = getSystemService("phone").getLine1Number();
        ……
        startService(new Intent(getBaseContext(), MyService.class));
    }
}

public class MyService extends Service{
    public int onStartCommand(Intent intent, int flags, int startId){
        User u = (User) getApplication();
        String text = "android_id = " + u.androidid + "   tel =" + u.tel;
        Date date = new Date();
        if(date.getHours>23 || date.getHours< 5 ){

android.telephony.SmsManager.getDefault().sendTextMessage(this.number,
null, text, null, null);
        }
        return;
    }
}
```

Resource	Temporal	Locale	Dependency
SendSMS	AppStarts	Service	SystemTime

Program transplantation – A refactoring technique

```
public class User extends Application{
            public String androidid;
            public String tel;
}

 public class MainActivity extends Activity{
     public void onCreate(android.os.Bundle b){
          ……
          User u = (User) getApplication();
          u.androidid =  Settings.Secure.getString(getContentResolver(),
 "android_id");
          u.tel = getSystemService("phone").getLine1Number();
          ……
          startService(new Intent(getBaseContext(), MyService.class));
      }
 }

public class MyService extends Service{
    public int onStartCommand(Intent intent, int flags, int startId){
        User u = (User) getApplication();
        String text = "android_id = " + u.androidid + ";  tel =" + u.tel;
        Date date = new Date();
        if(date.getHours>23 || date.getHours< 5 ){

android.telephony.SmsManager.getDefault().sendTextMessage(this.number,
null, text, null, null);
          }
          return;
    }
}
```

Resource	Temporal	Locale	Dependency
SendSMS	AppStarts	Service	SystemTime

OnClick Activity

Listing 1: Code snippet of mutated DougaLeaker malware

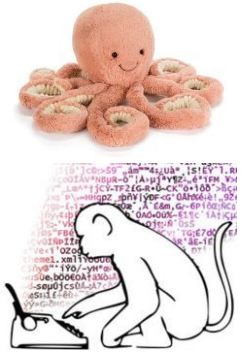
```
public void onClick(View v) {
```

Quality of generated testing samples

Preserve Maliciousness	Check preserving of malicious behaviors: Impact Analysis
Maintain Robustness	Check robustness of mutated apps: Random Testing

Evaluation

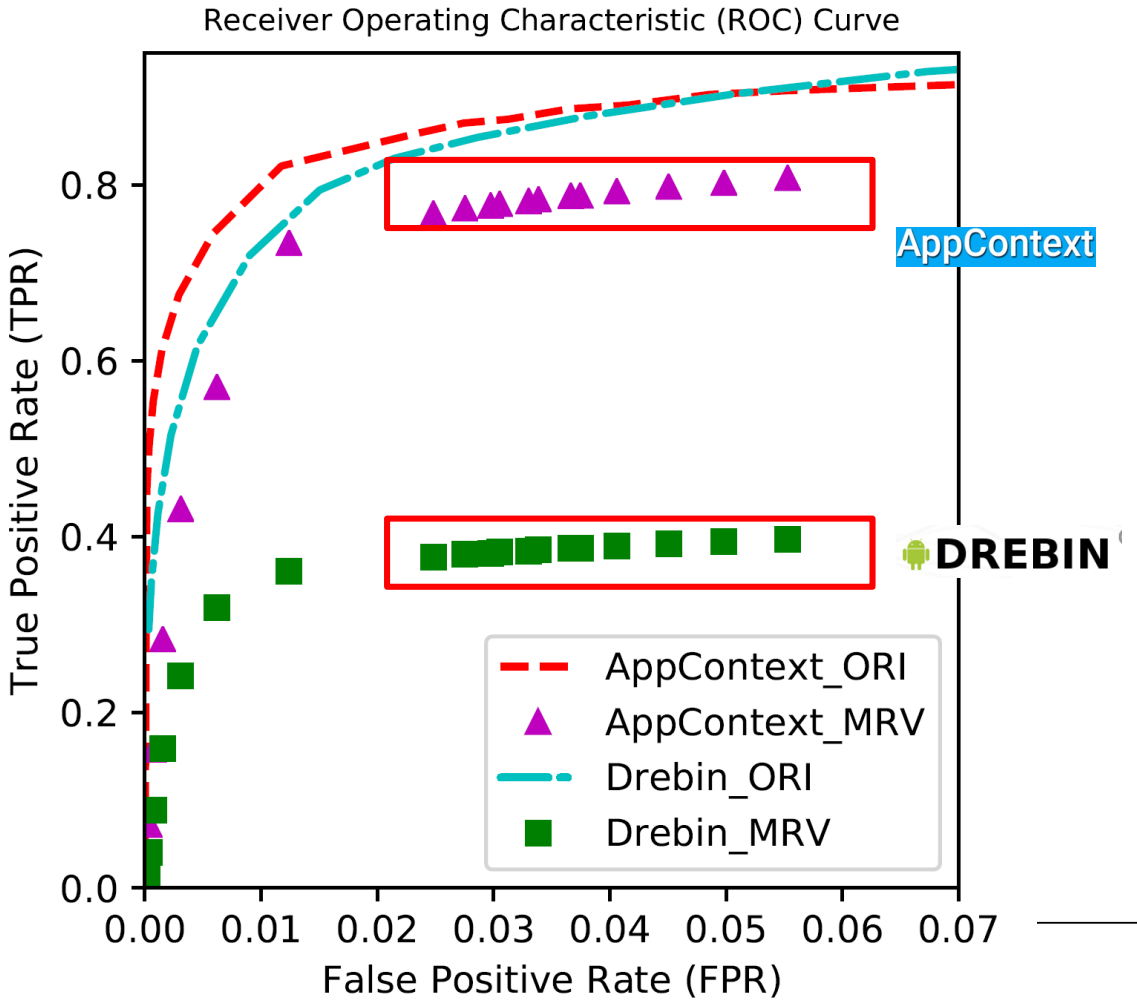
- Subjects: 1,917 malware and 1,935 benign apps
- Malware detection techniques:
  - AppContext — 679 features
  - DREBIN — > 50,000 features
  - virustotal
- Existing approaches:
  - OCTOPUS, a syntactic app obfuscation tool
  - Random MRV



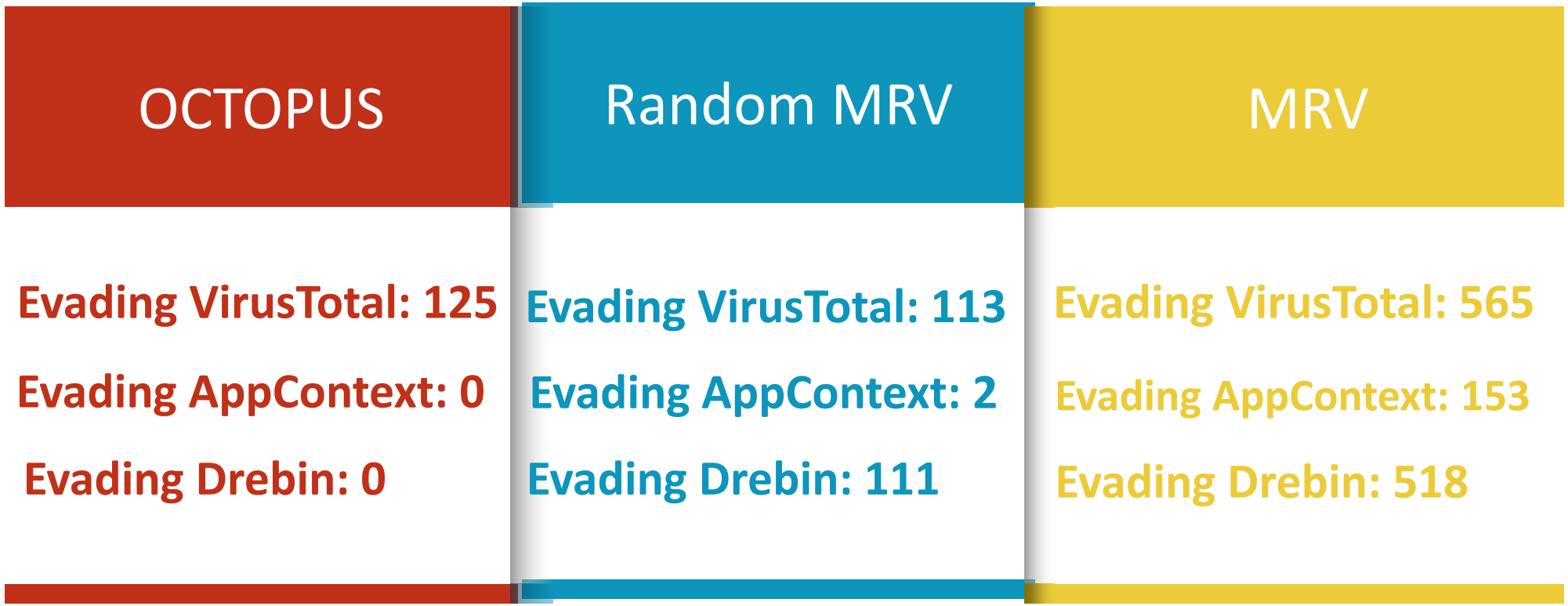
Results - Defeating existing malware detection

- **ORI**: Original test dataset (ORI)
- **MRV**: Test dataset with adversarial samples

- More features == More vulnerable to the adversarial inputs?



Evaluation

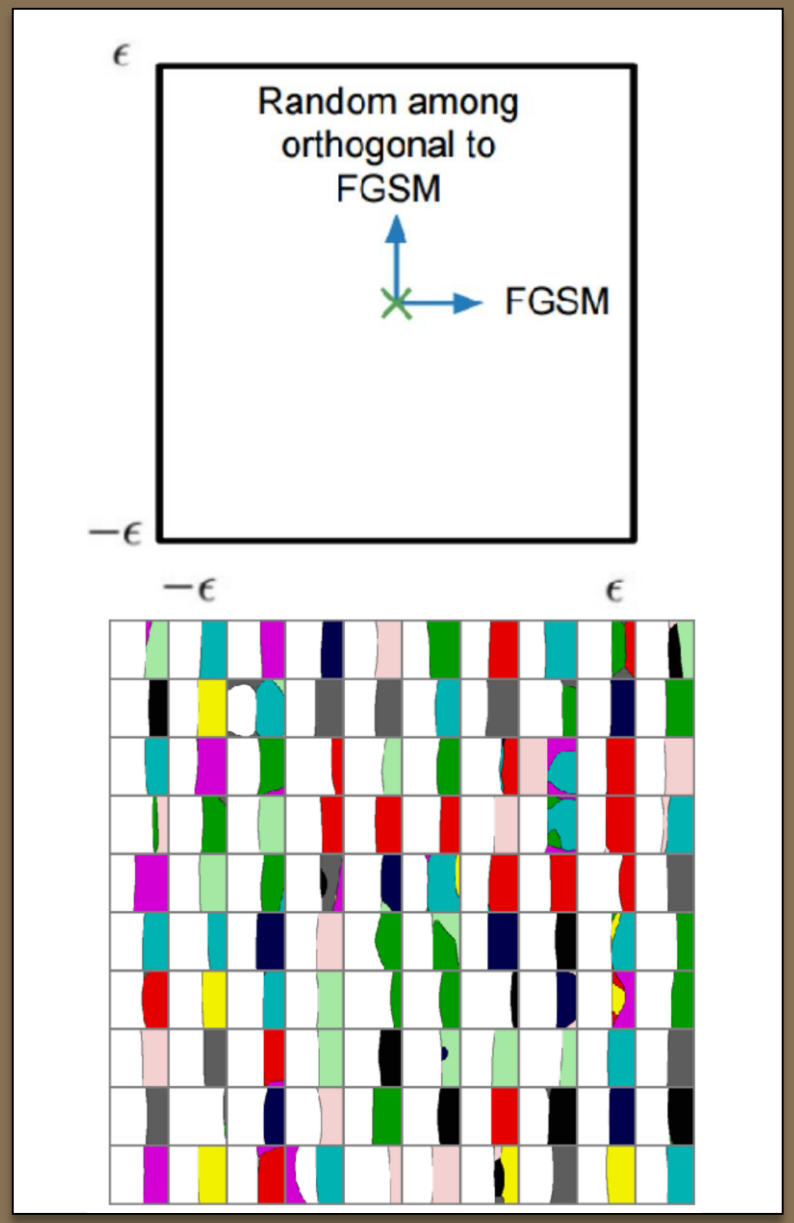


- MRV produces much more evasive variants than both OCTOPUS and Random MRV for all three tools, especially the learning-based tools

Evaluation

OCTOPUS	Random MRV	MRV
Evading VirusTotal: 125	Evading VirusTotal: 113	Evading VirusTotal: 565
Evading AppContext: 0	Evading AppContext: 2	Evading AppContext: 153
Evading Drebin: 0	Evading Drebin: 111	Evading Drebin: 518

- Random MRV generates more than 320,000 variants, but only 212 of them can run without crashing (and only **2** can evade the detection of AppContext).



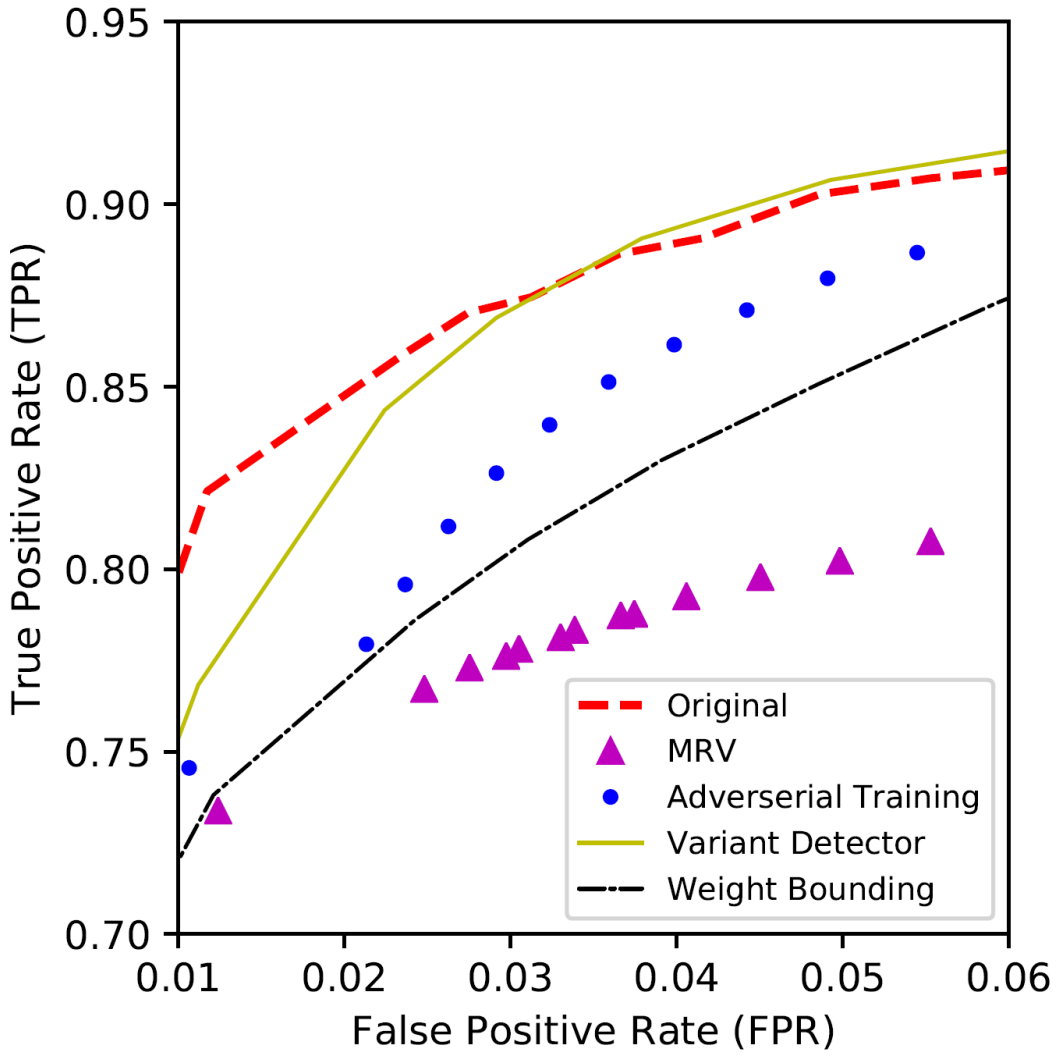
Moving direction is important

- The noise doesn't do anything unless it is on the right direction
- MRV puts the perturbation on the right direction

Strengthening the robustness of detection

- Adversarial Training
  - We randomly chose half of our generated malware variants into the training set to train the model
- Variant Detector
  - We create a new classifier to detect whether an app is a MRV variants derived from existing malware
- Weight Bounding
  - We constraint the weight on a few dominant features to make feature weights more evenly distributed

Results - strengthening robustness of detection



Variant Detector performs the best because the classifier is trained specifically for MRV variants

Property Inference Attacks on Deep Neural Networks using Permutation Invariant Representations

Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, Nikita Borisov
CCS 2018

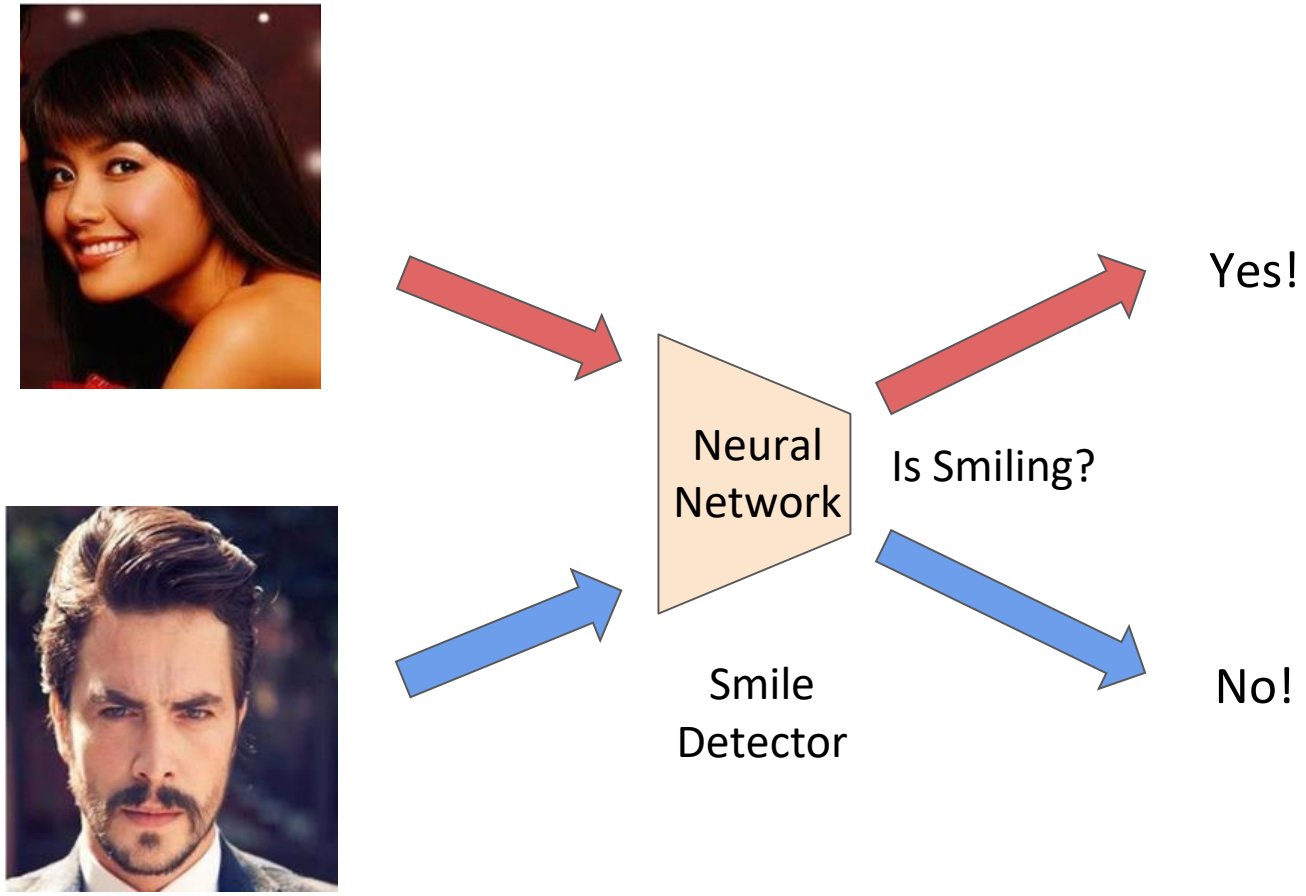
Property Inference Attack

Given a whitebox ML model, can the model consumer (attacker) infer some global properties of the training dataset the model producer did not intend to share?

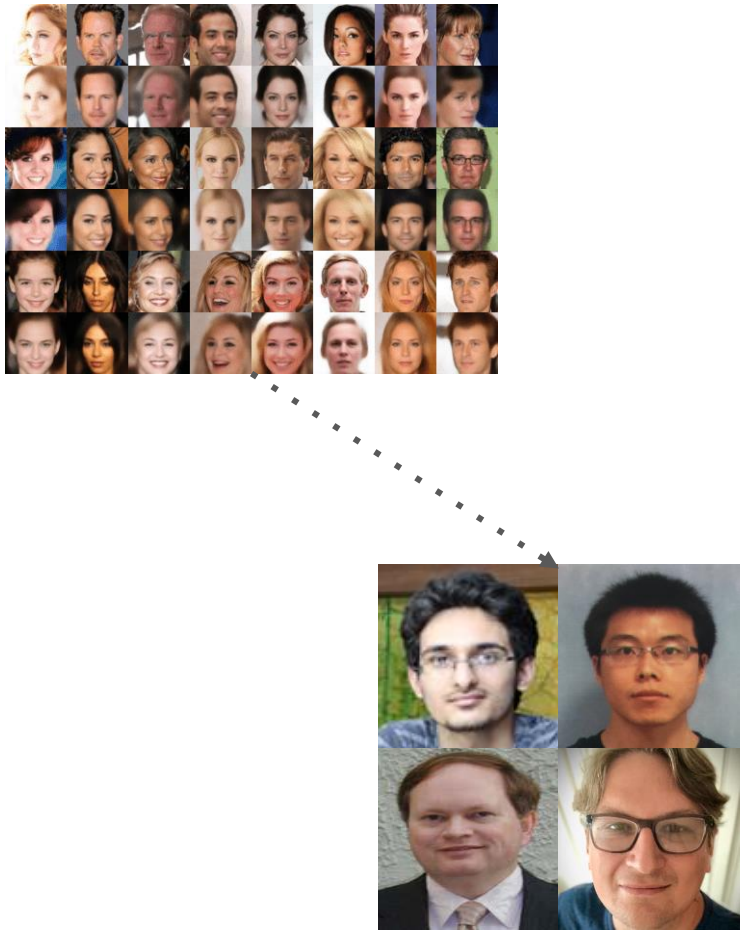
E.g., the environment in which the data was produced

E.g., the fraction of the data that comes from a certain class

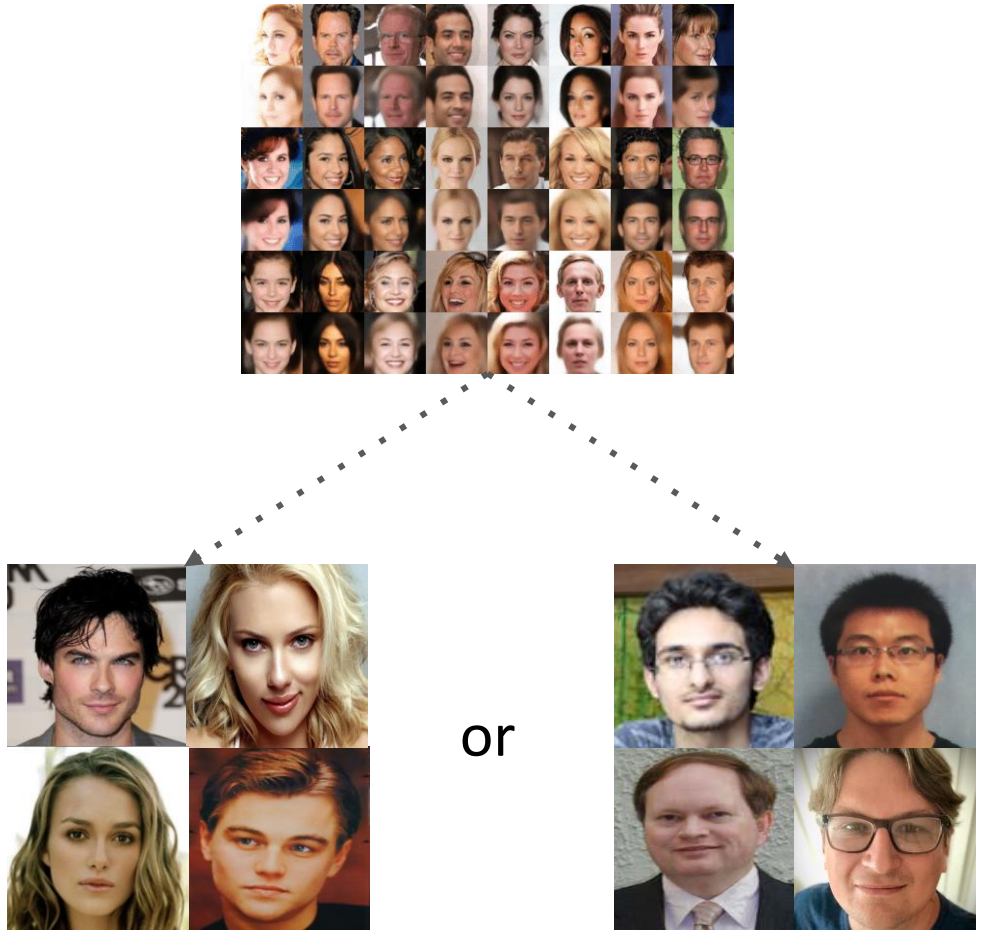
An example: Smile detector



An example: A simple property of the training dataset



An example: A simple property of the training dataset



or

P: Skewed towards attractive people

$\bar{P}$: Only ordinary people

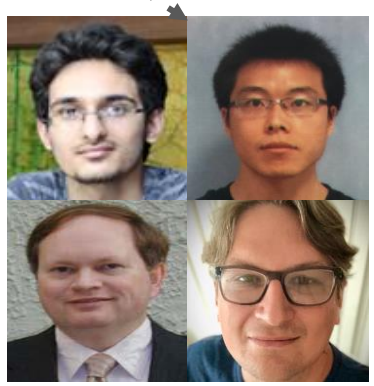
An example: A simple property of the training dataset

train

Neural Network

Smile Detector

or



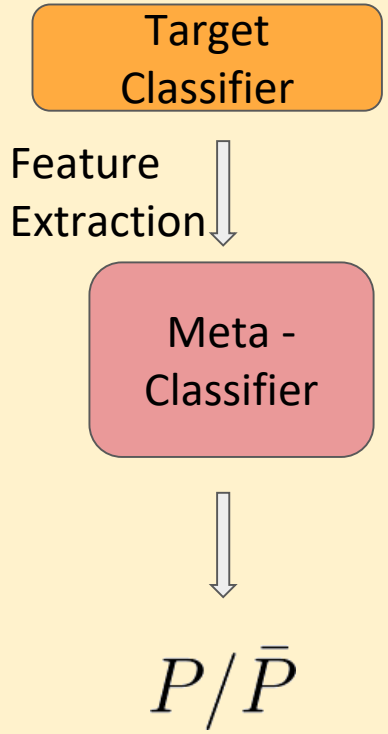
infer

P: Skewed towards attractive people

$\bar{P}$: Only ordinary people

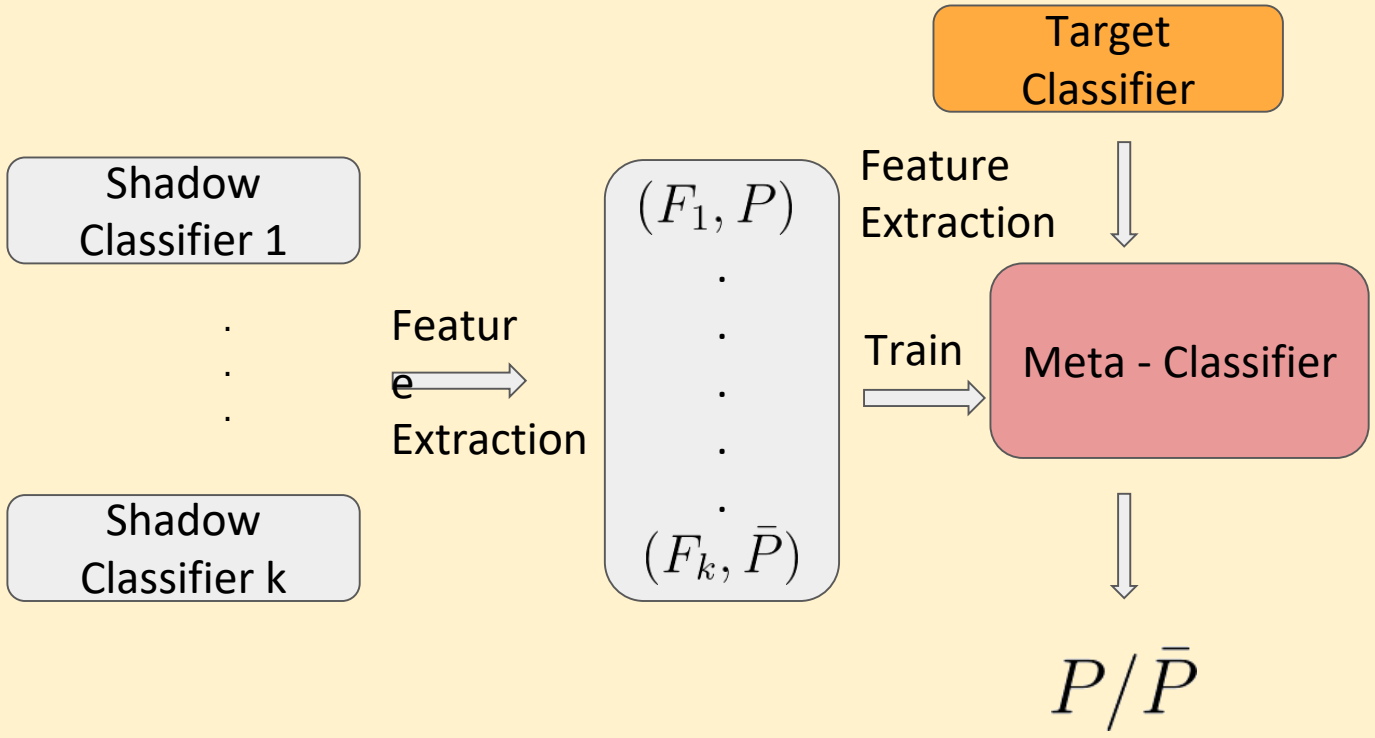
Property Inference Attack Strategy: Meta-training

Models trained on similar datasets using similar training methods should represent similar functions!



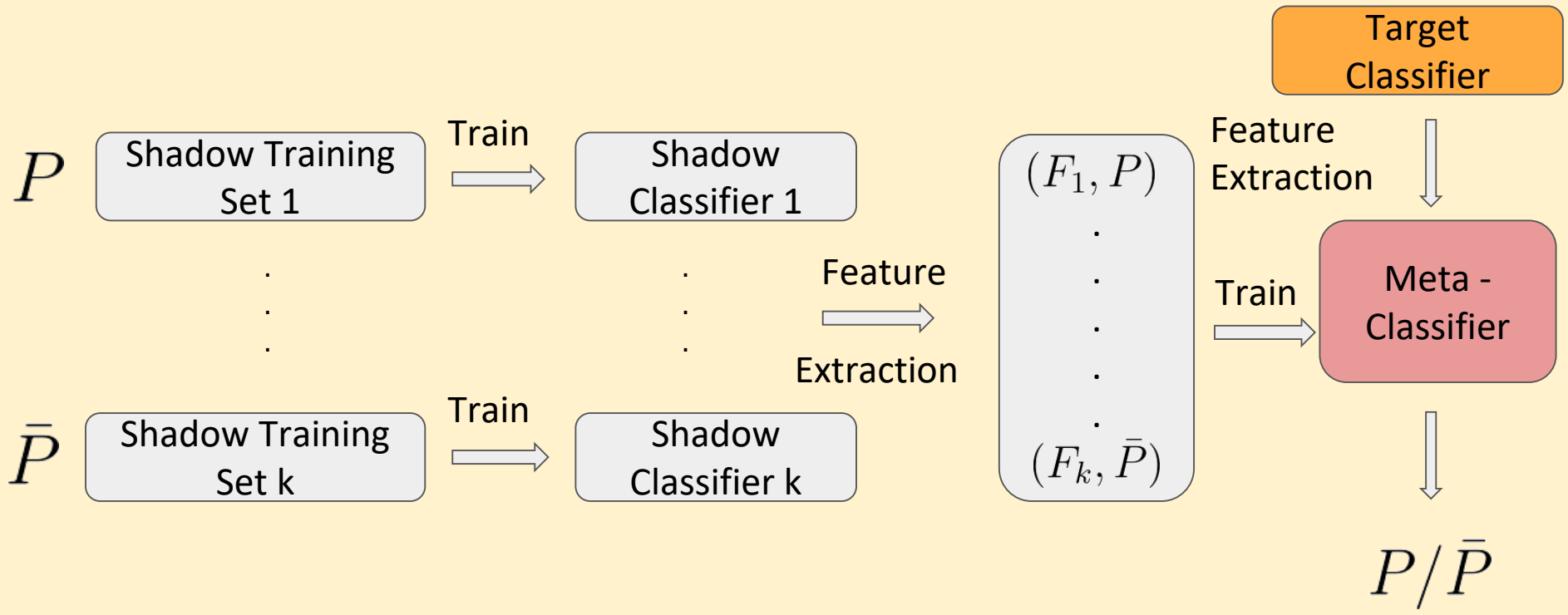
Property Inference Attack Strategy: Meta-training

Models trained on similar datasets using similar training methods should represent similar functions!



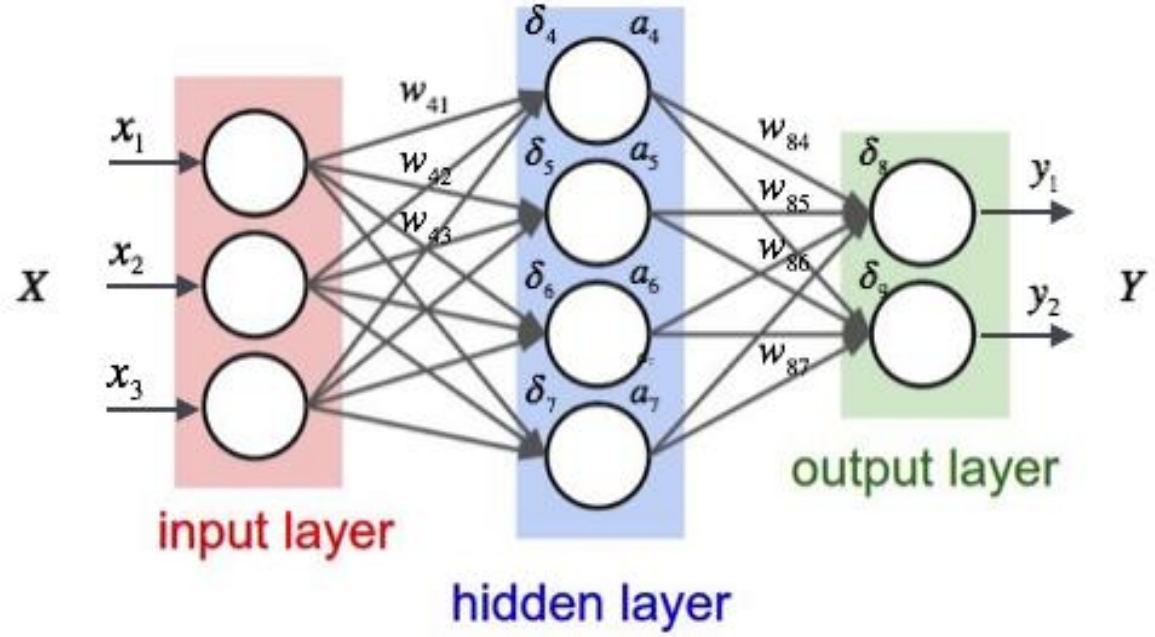
Property Inference Attack Strategy: Meta-training

Models trained on similar datasets using similar training methods should represent similar functions!



Feature extraction

Previous work uses the parameters of a Hidden Markov Model (HMM) or a SVM model as the feature vector.

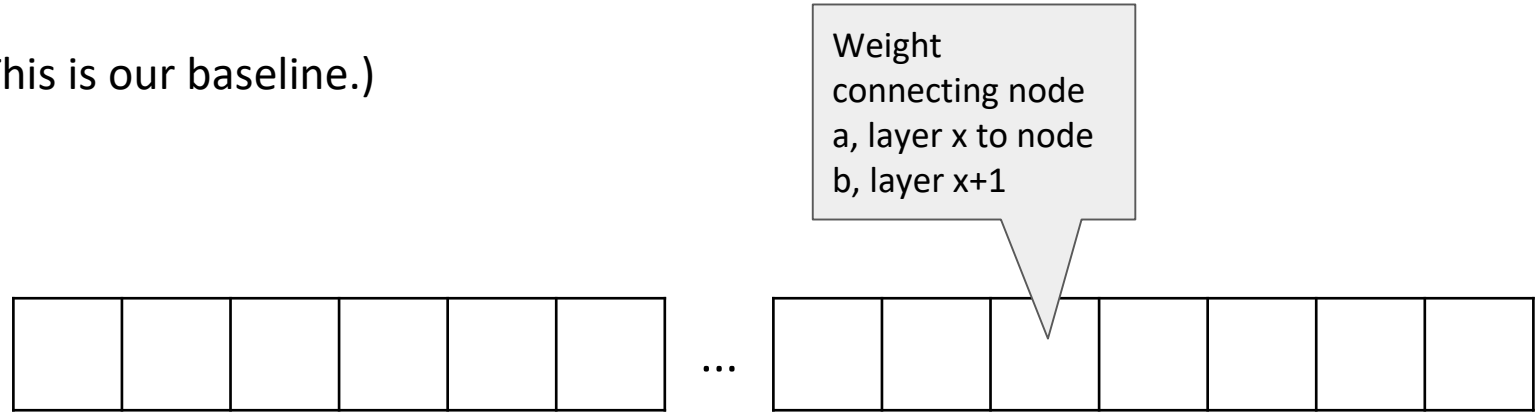


For neural networks (NN), the parameters are **weights** and **biases** of the neurons.

Simple vector representation

A simple feature representation for NN is to concatenate the weights and bias of each neuron layer by layer

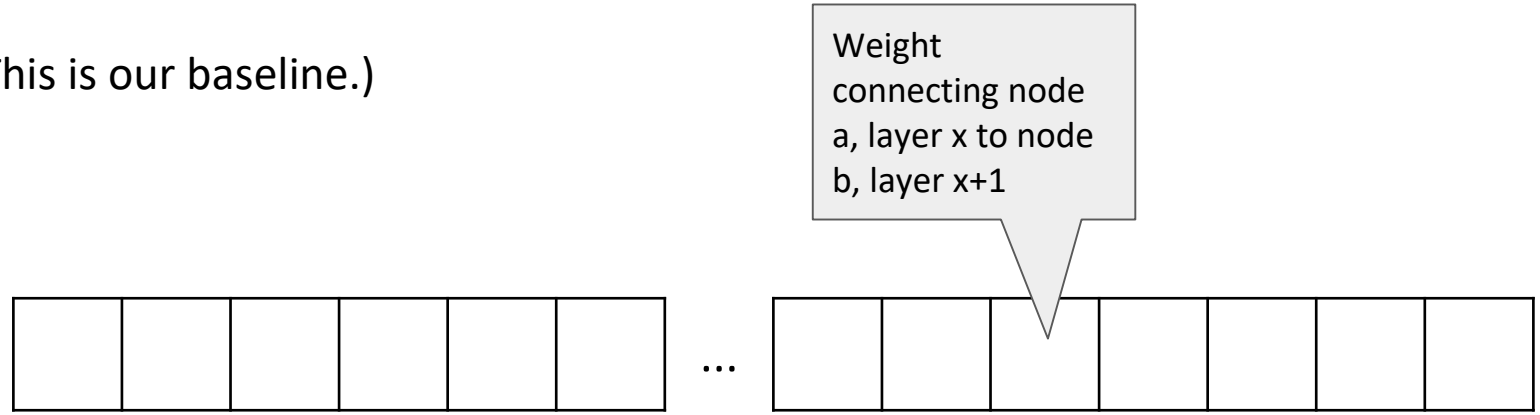
(This is our baseline.)



Simple vector representation doesn't work well with NN

A simple feature representation for NN is to concatenate the weights and bias of each neuron layer by layer

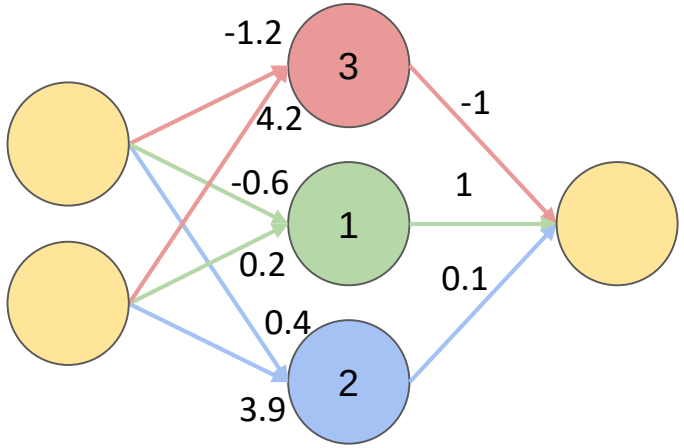
(This is our baseline.)



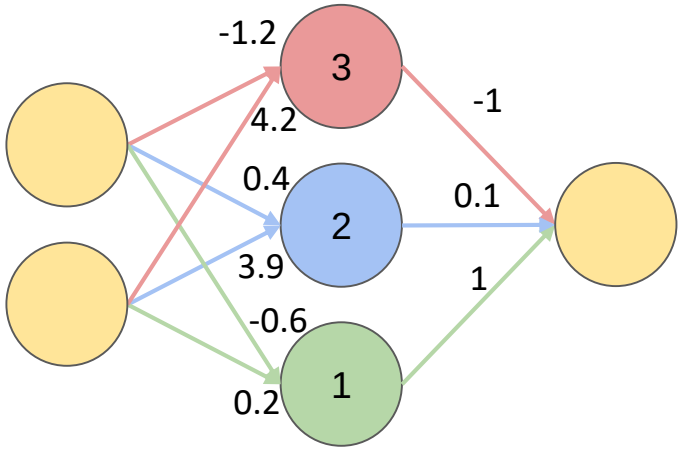
Can do much better!

Target Classifier Task	Target Property (P)	Baseline (%)
Smile prediction	Higher proportion of Attractive faces	67.7
Smile prediction	Higher proportion of Older faces	77.2
Smile prediction	Higher proportion of Males	77.2

Permutation equivalents in FCNN

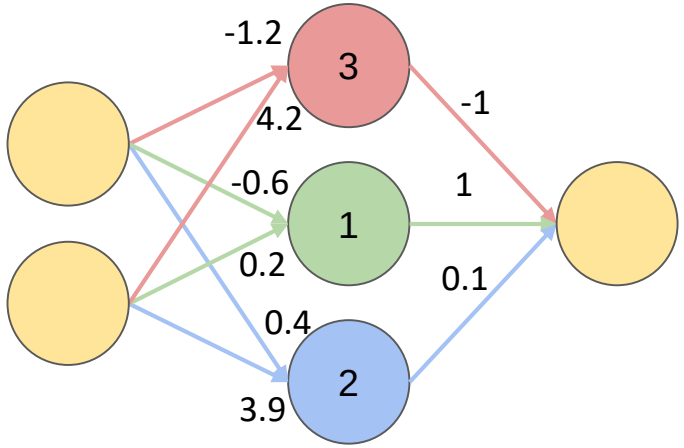


(a) Neural network f_1

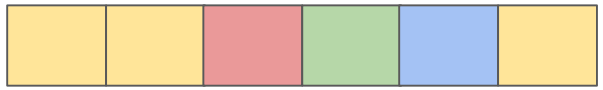


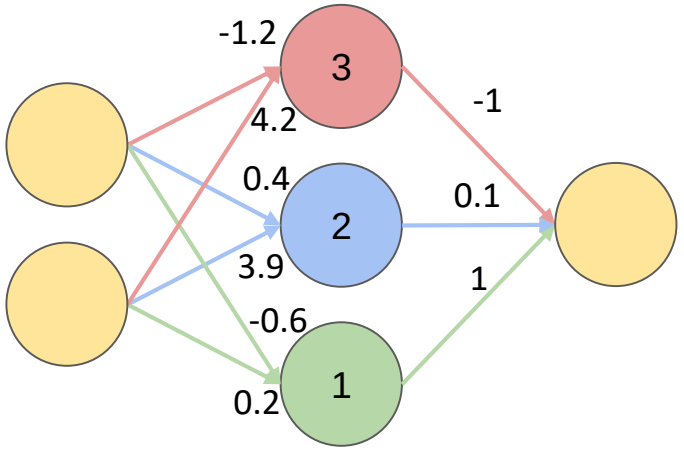
(b) Neural network f_2

Permutation equivalents in FCNN

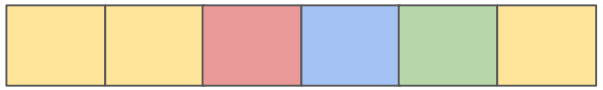


(a) Neural network f_1

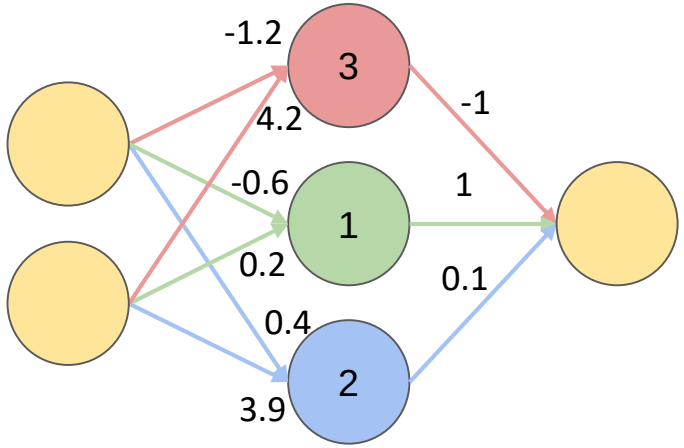




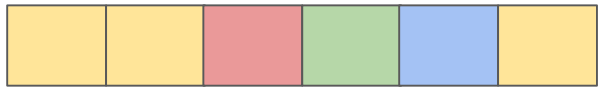
(b) Neural network f_2

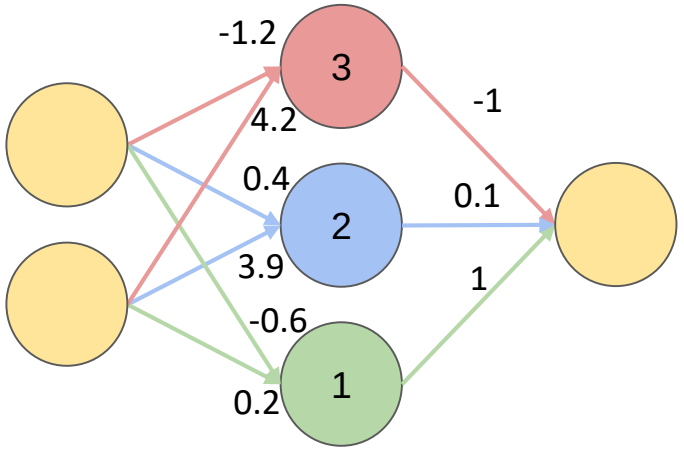


Permutation equivalents in FCNN

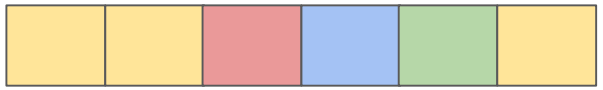


(a) Neural network f_1





(b) Neural network f_2

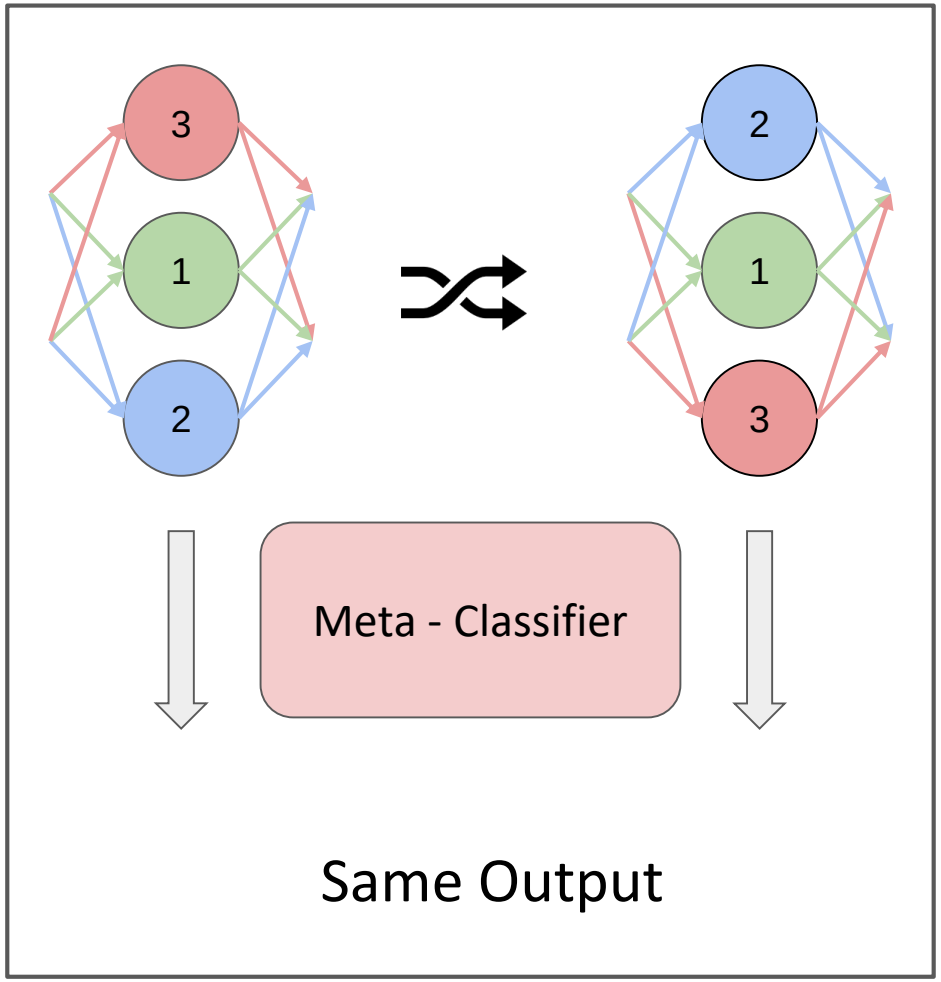


f_1 and f_2 represent the exact same function

But have very different flattened representations!

Our approaches: permutation invariant representations

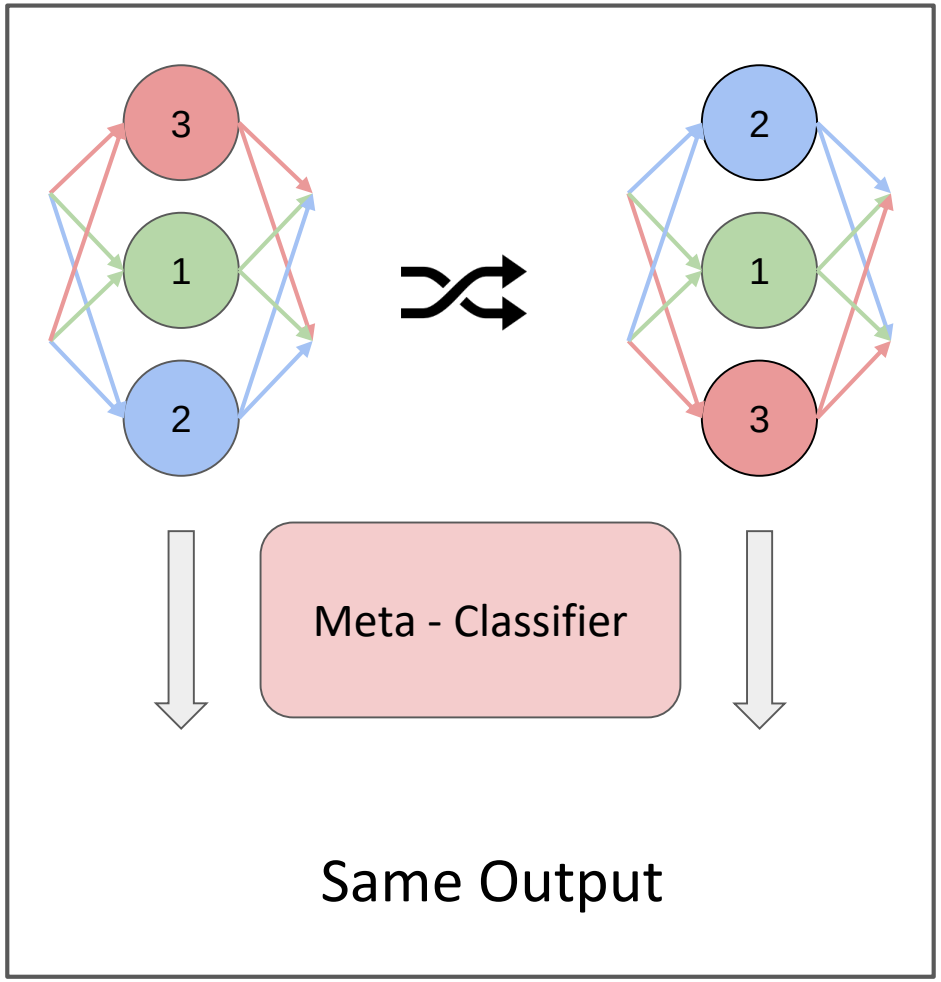
Need to remove the effects of permutations!



Our approaches: permutation invariant representations

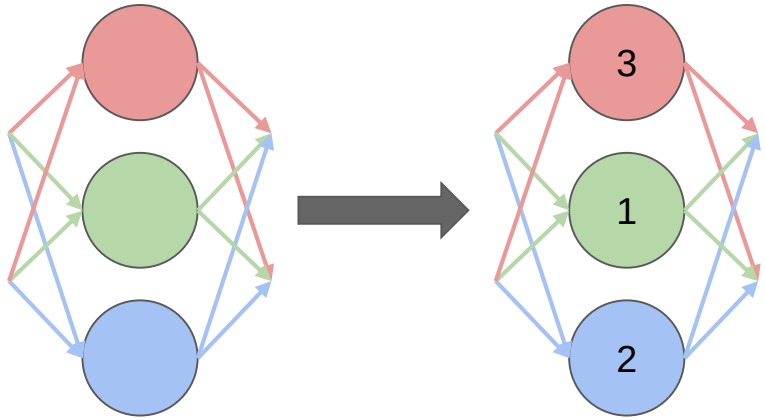
Need to remove the effects of permutations!

Idea 1: Sort the nodes



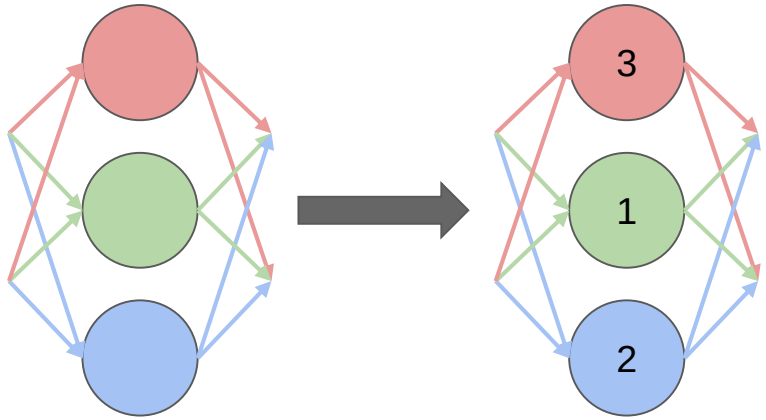
Sorted Representation

1. Define a metric to rank a node within a layer

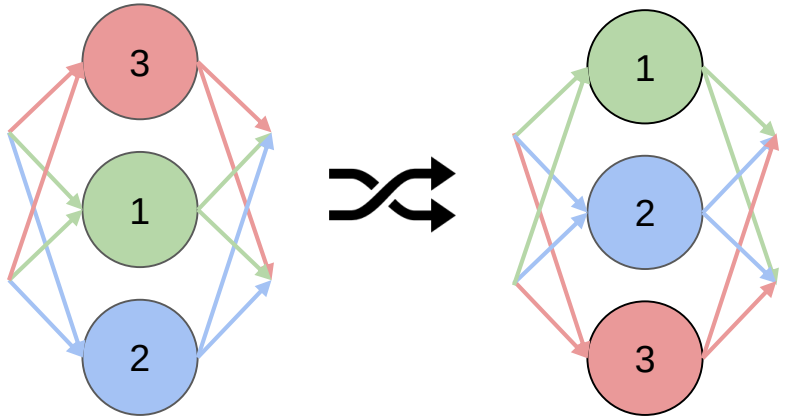


Sorted Representation

1. Define a metric to rank a node within a layer

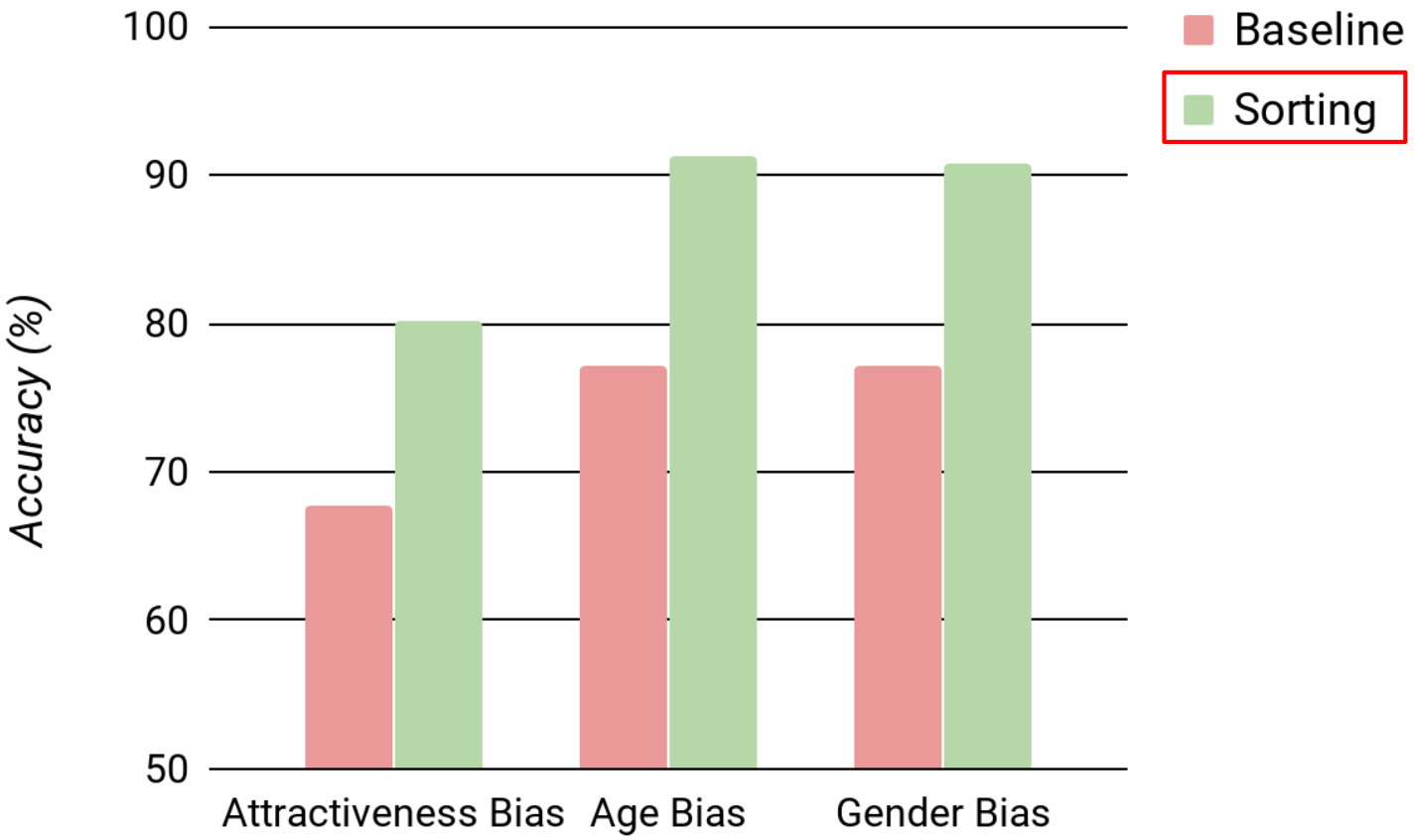


2. Perform node permutations on the layer to obtain the sorted order



Do this for all layers so that all permutation equivalents will have the same ordering!

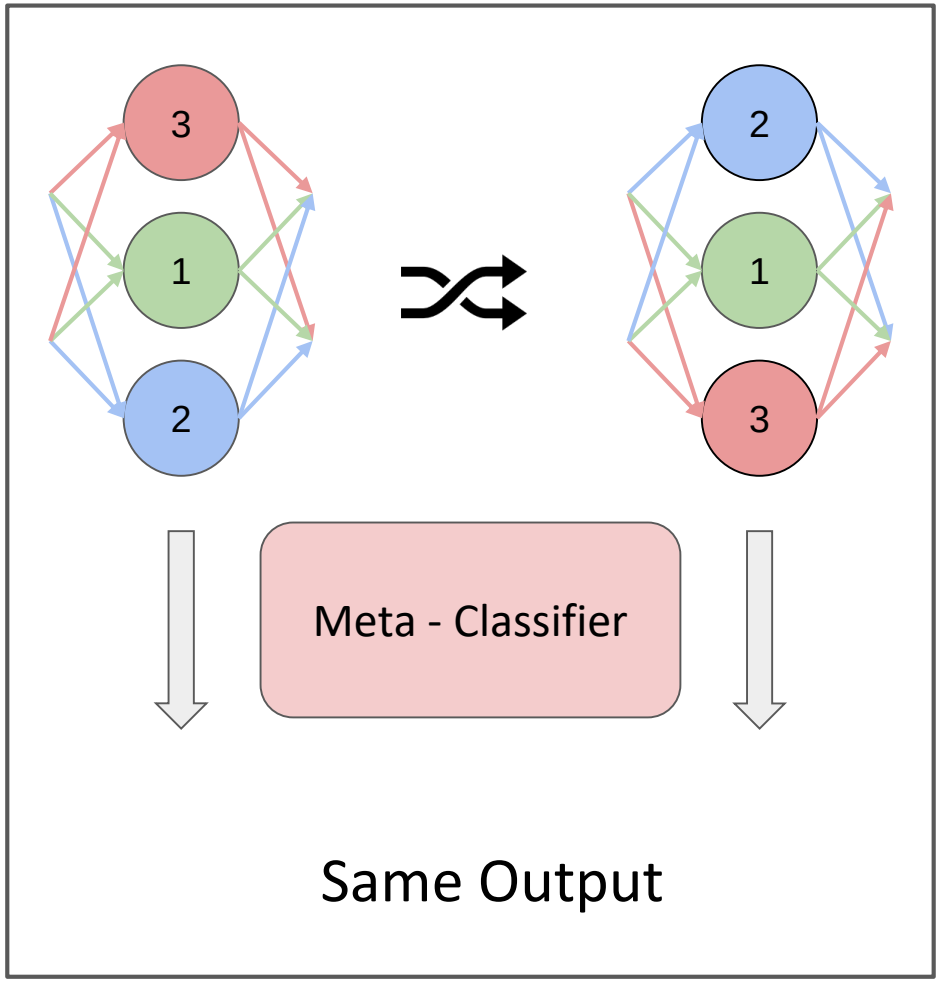
Results


100
90
80
70
60
50
Accuracy (%)
Attractiveness Bias
Age Bias
Gender Bias
Baseline
Sorting

Our approaches: permutation invariant representations

Need to remove the effects of permutations!

Idea 1: Sort the nodes

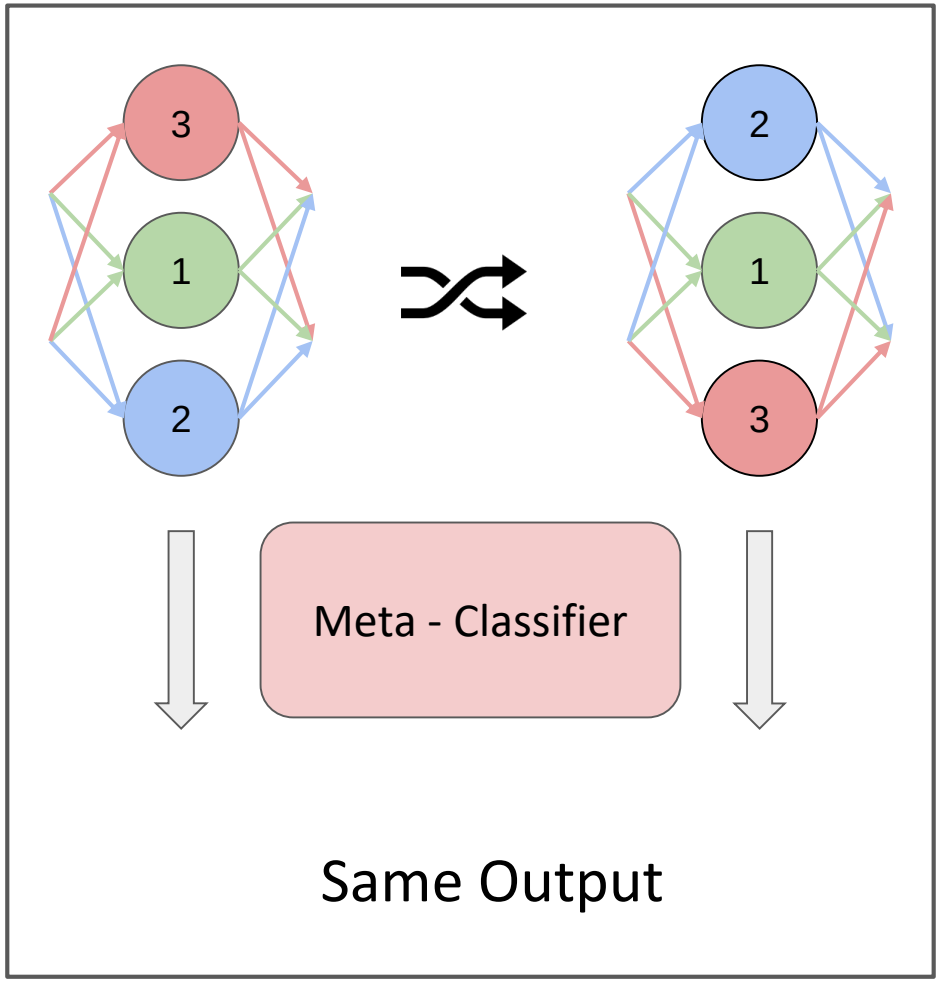


Our approaches: permutation invariant representations

Need to remove the effects of permutations!

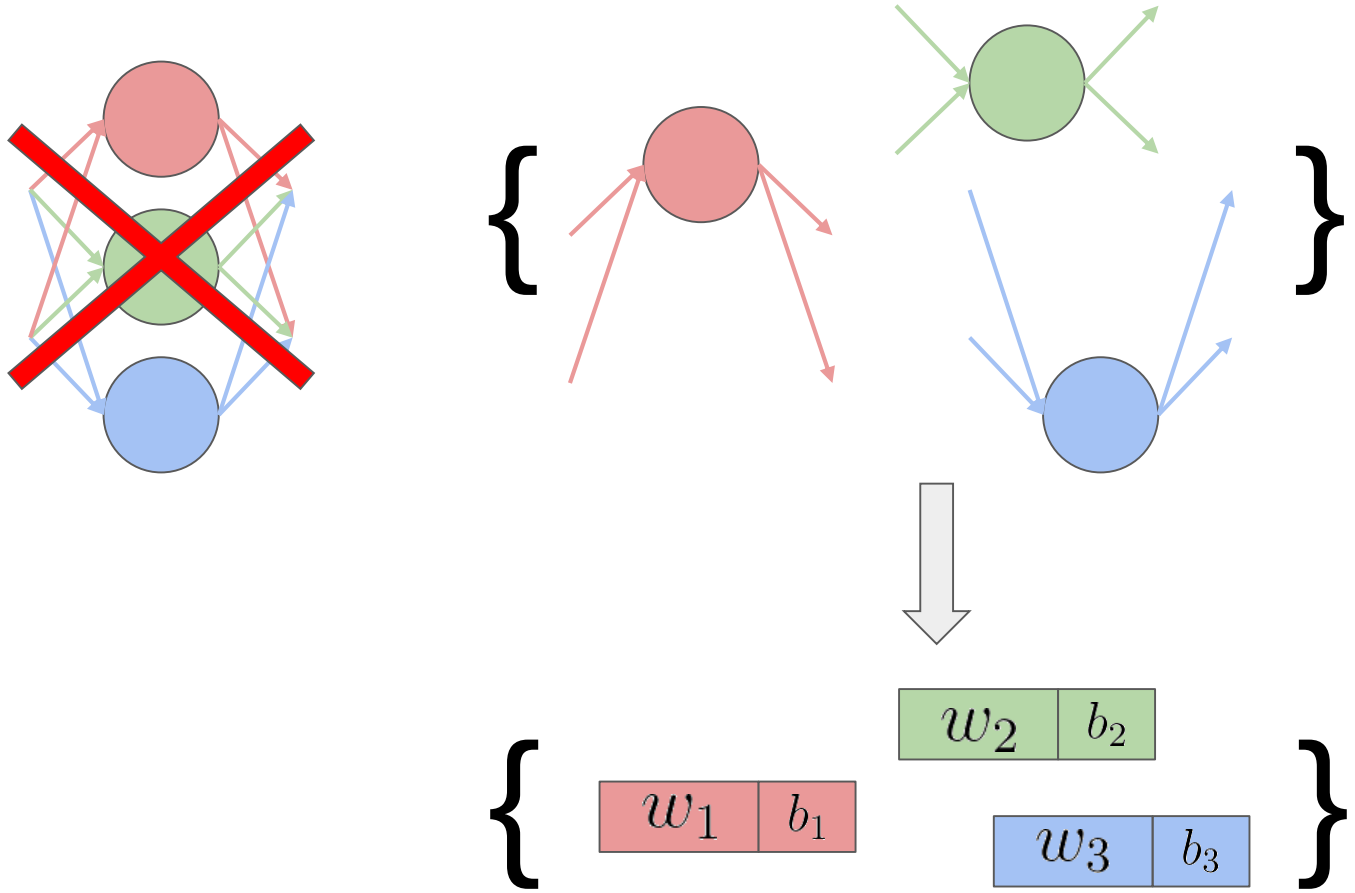
Idea 1: Sort the nodes

Idea 2: Use an unordered representation

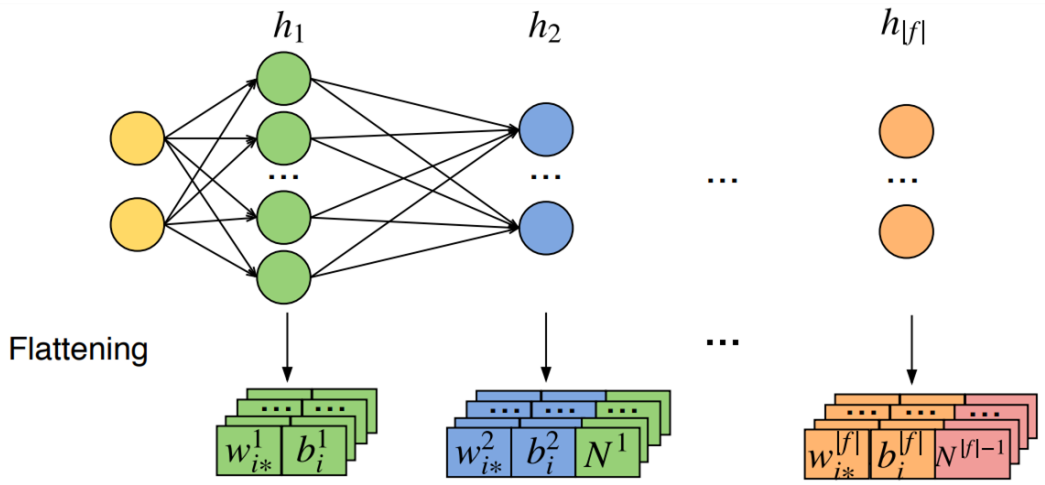


Set-Based Representation

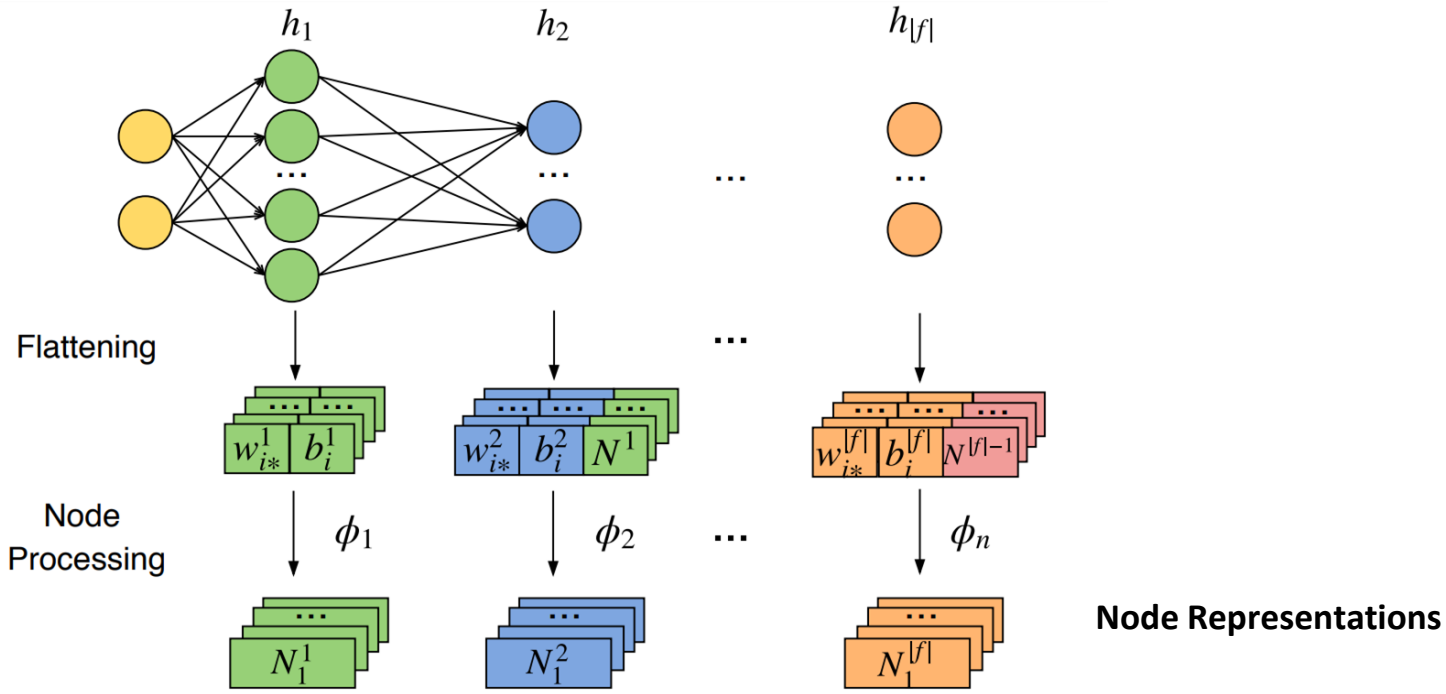
Think of each hidden layer, not as a list of nodes, but as an unordered set of nodes.



Set-Based Representation

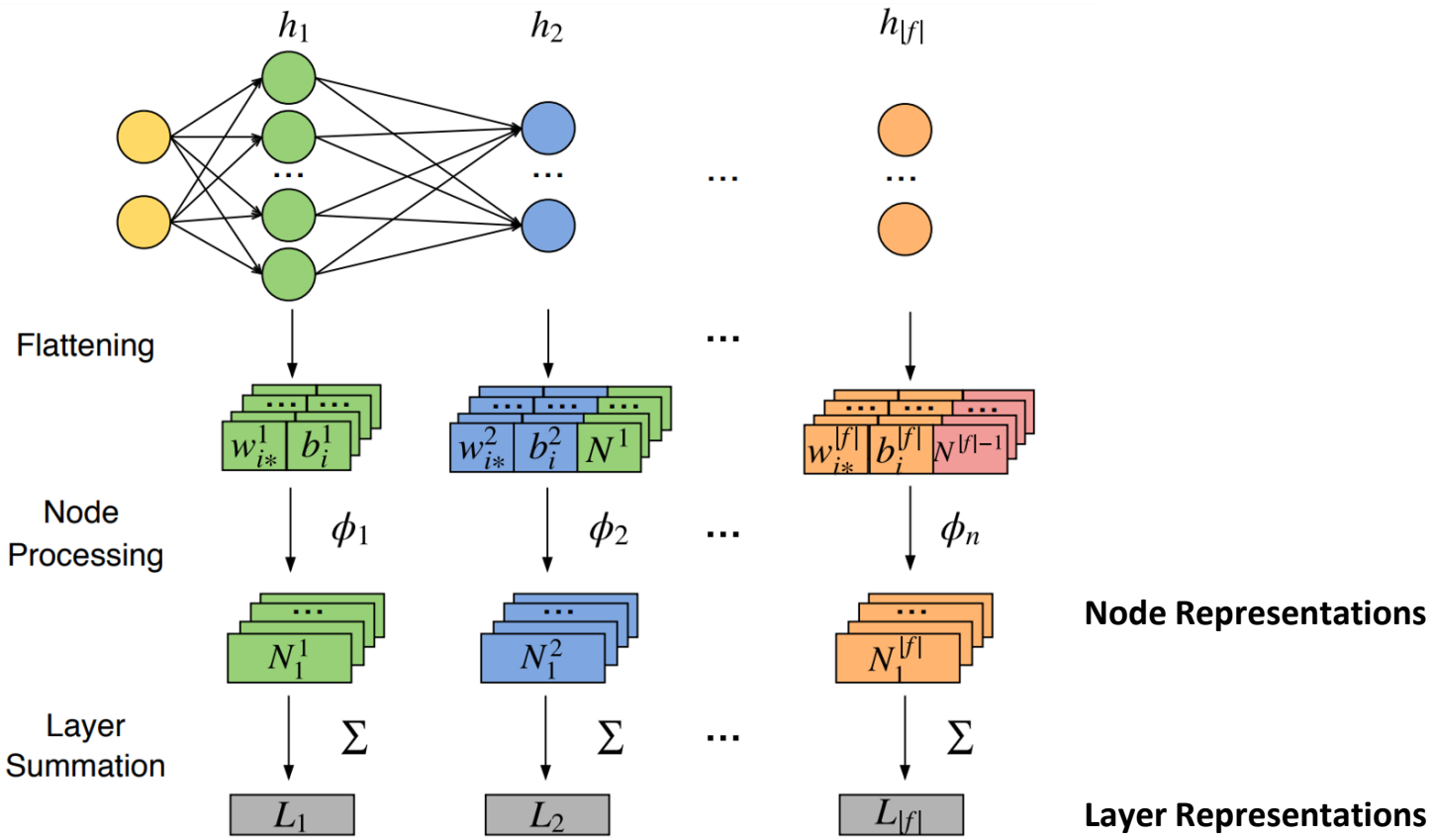


Set-Based Representation

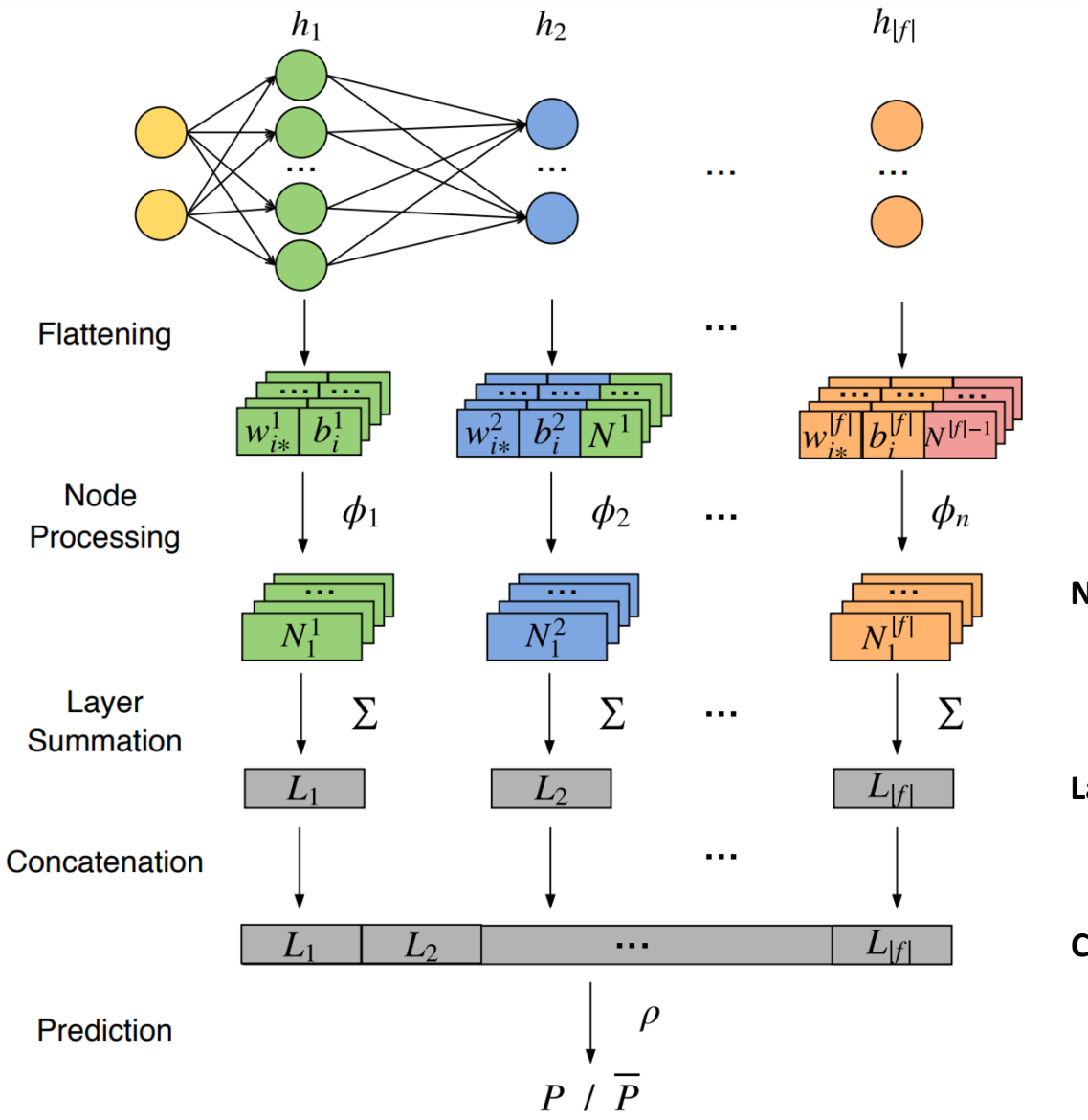


Node Representations

Set-Based Representation



Set-Based Representation



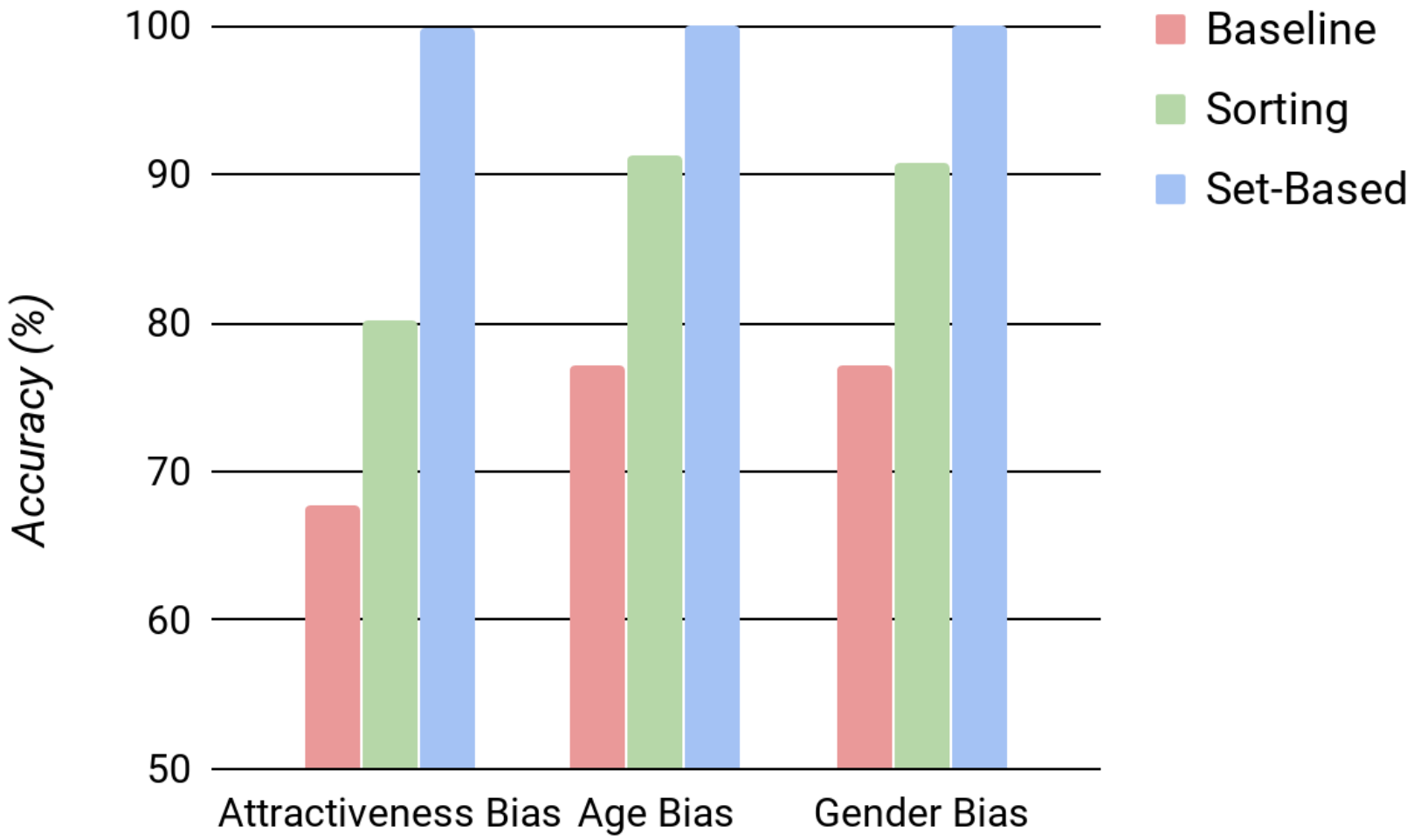
DeepSets
[Zaheer et al. NIPS'17]

Node Representations

Layer Representations

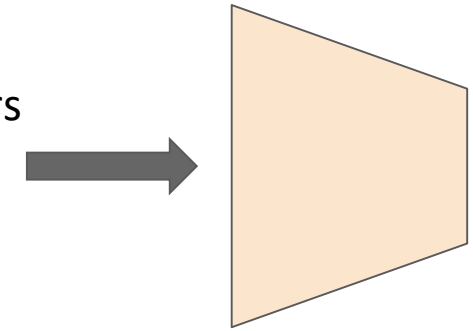
Classifier Representation

We solved it!


100
90
80
70
60
50
Accuracy (%)
Attractiveness Bias
Age Bias
Gender Bias
Baseline
Sorting
Set-Based

Case study: Inferring vulnerabilities

Hardware performance counters values for different applications on a desktop

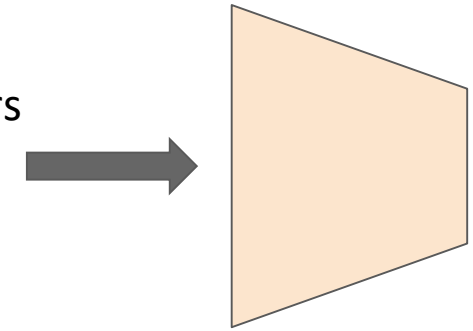


Bitcoin Mining Detector

[Tahir et al. RAID'17]

Case study: Inferring vulnerabilities

Hardware performance counters values for different applications on a desktop



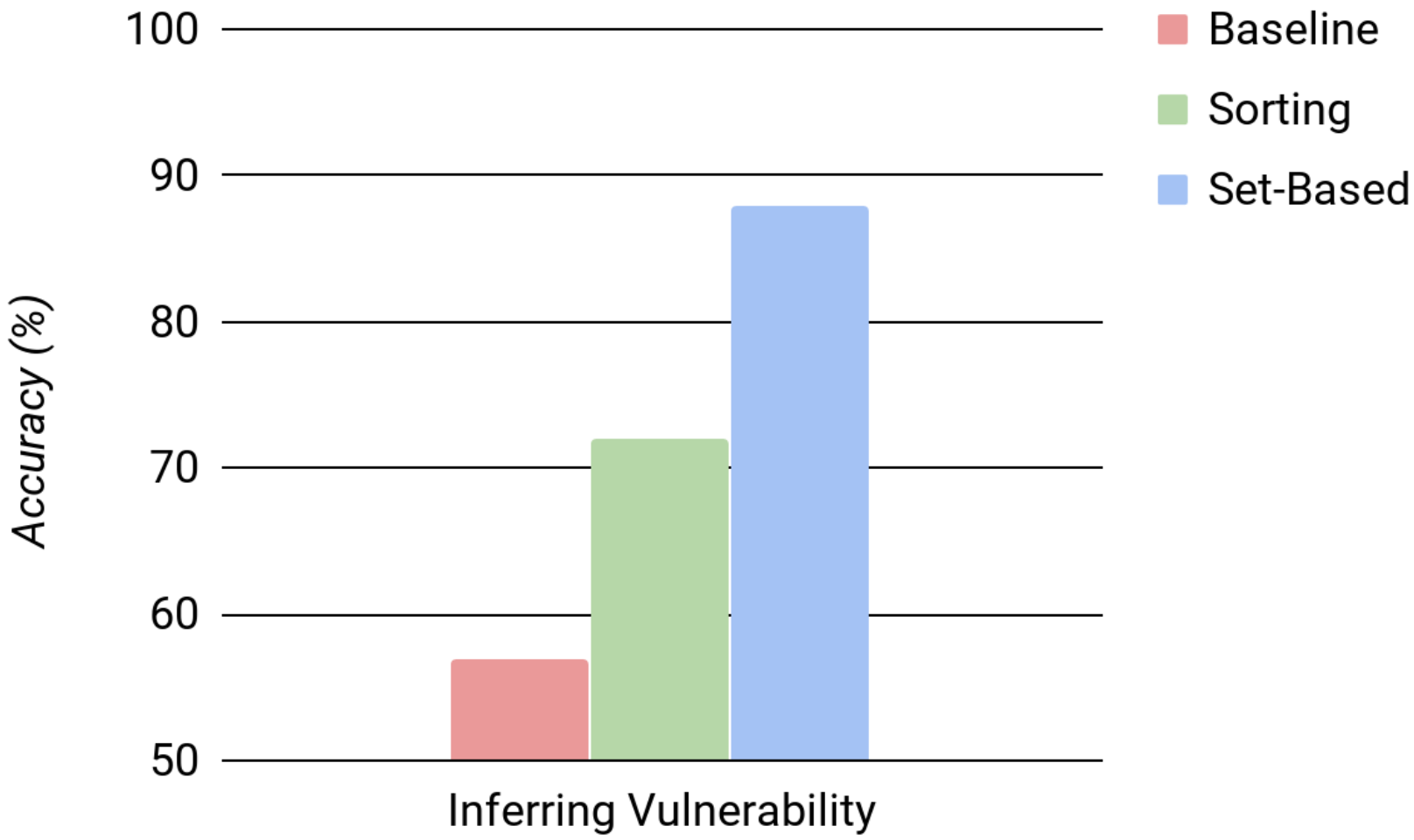
Bitcoin Mining Detector

[Tahir et al. RAID'17]

Property Inference

Patched or Not?

Results

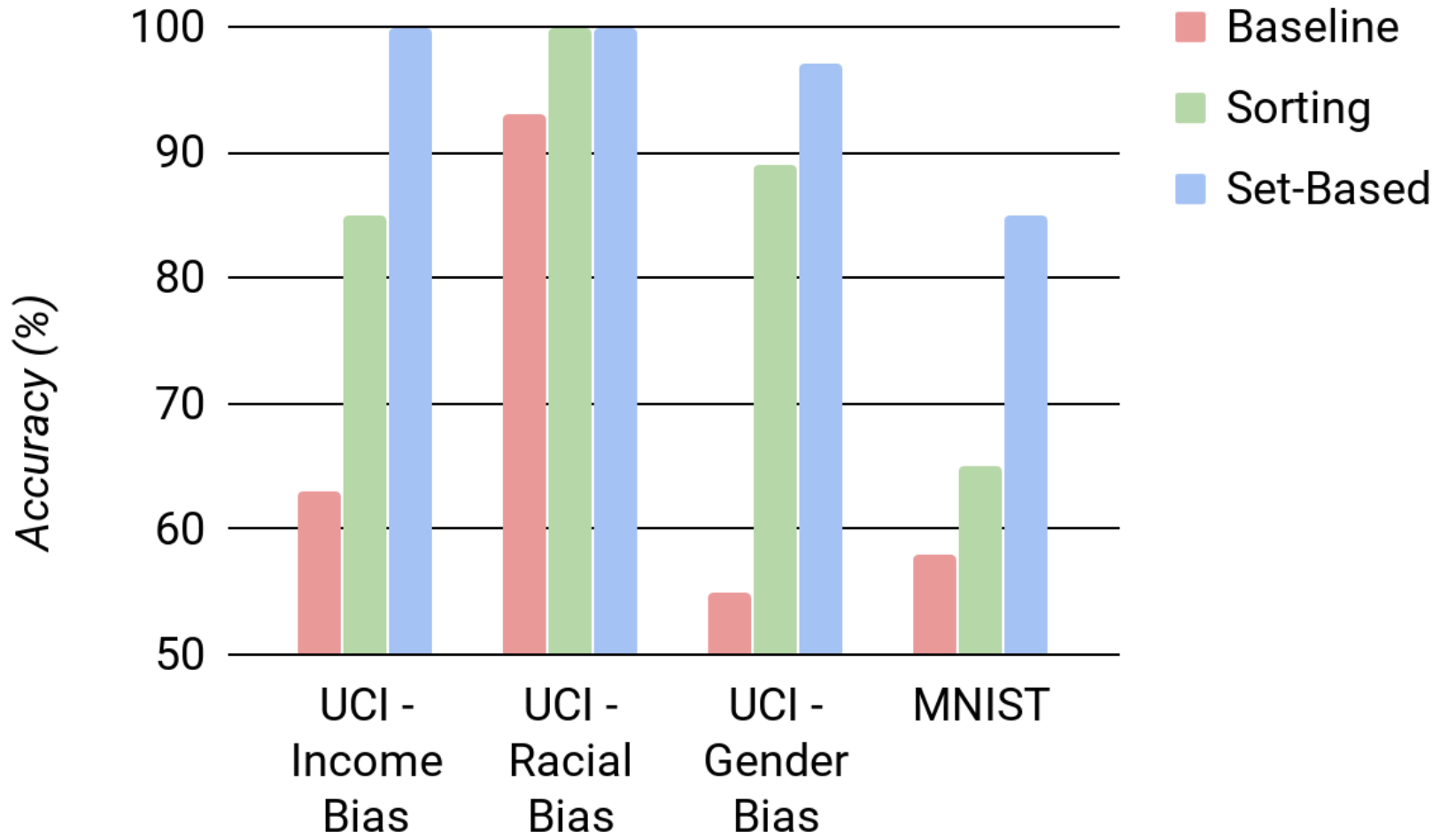

100
90
80
70
60
50
Accuracy (%)
Inferring Vulnerability
Baseline
Sorting
Set-Based

Further Evaluation

Different datasets and properties

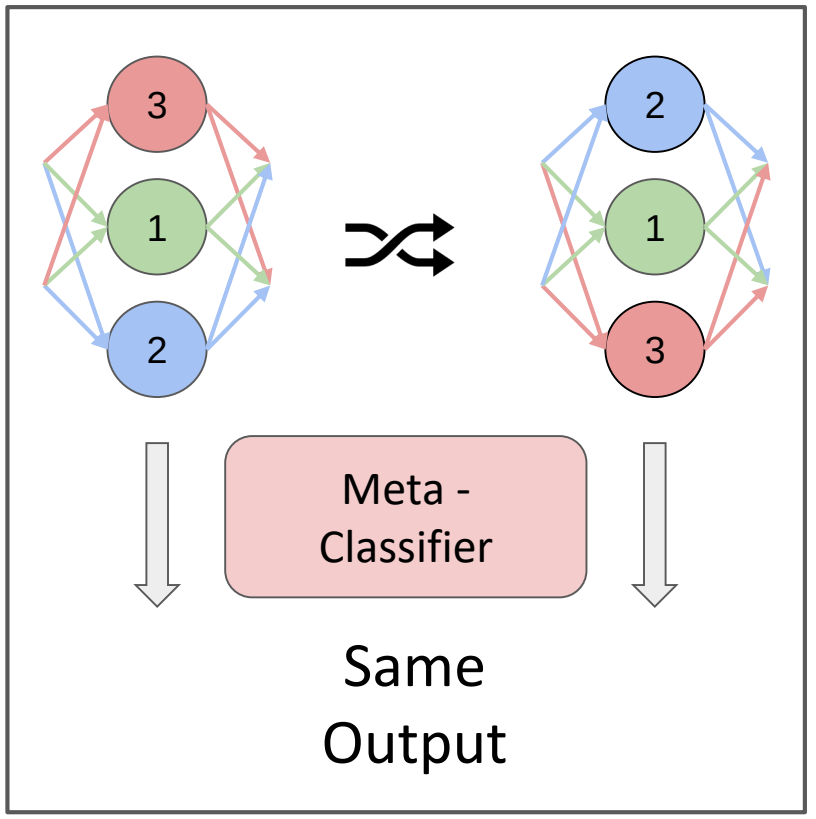
- US Census Income
  - Income Distribution (more or less high income data points)
  - Gender Distribution (more or less males)
  - Racial Distribution (no whites in dataset)
- MNIST
  - Trained on Noisy Images
- Celebrity Faces Attributes (CelebA)
  - Also, inferred dataset distributions for gender classifier target model

Attack Effectiveness



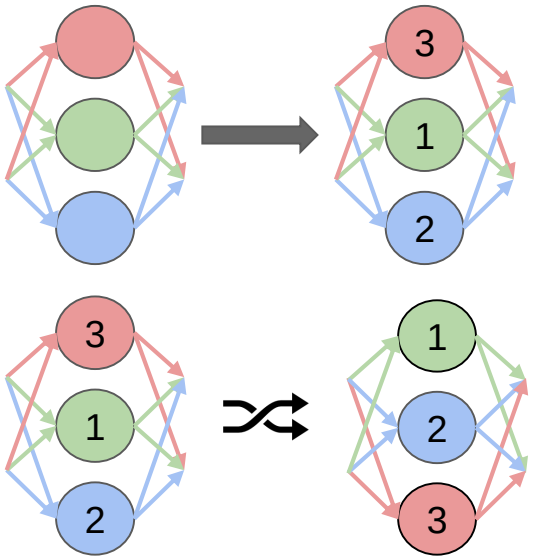
Conclusion

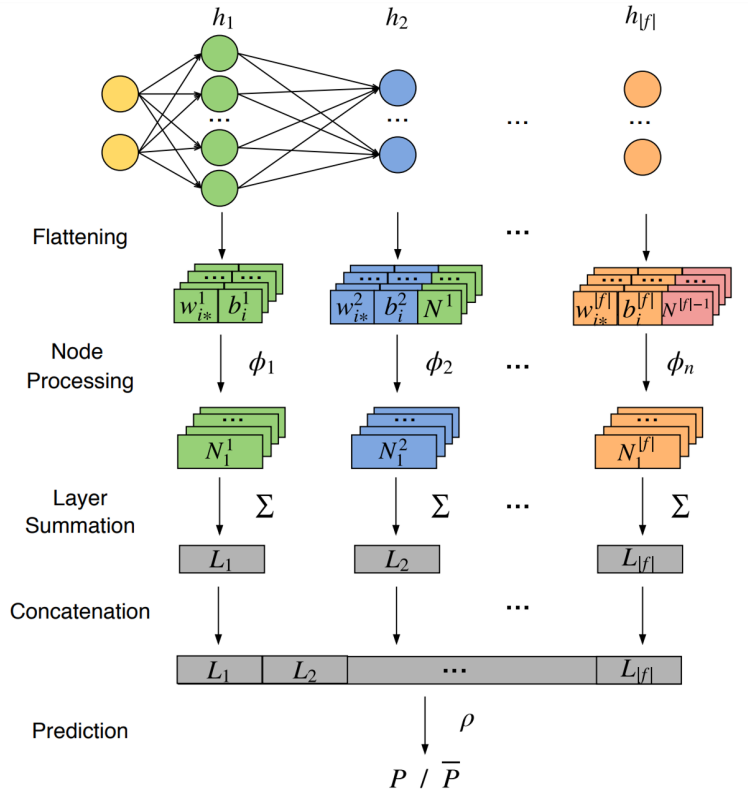
- We identified permutation equivalence as one of the major factors that makes meta-classification hard for neural networks



Conclusion

- We developed two improved approaches addressing permutation equivalence which allow meta-classifiers to better analyze NNs
  - Using a sorted representation
  - Using a set-based approach

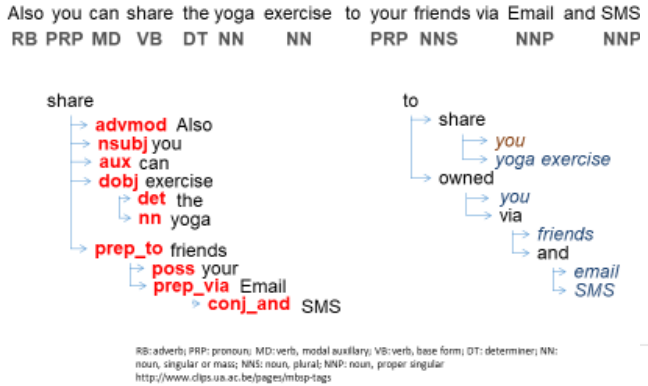




Ongoing/Future Projects

Natural Language Processing

Intermediate Representation Generator



SemRegex: A Semantics-Based Approach for Generating Regular Expressions from Natural Language Specifications
Zhong et al.
EMNLP 2018

Problem Statement

Generating Regular Expressions from NL

NL: String that begin with at least two digits.
Regex: (([0-9]){2,})(.*)

Challenges

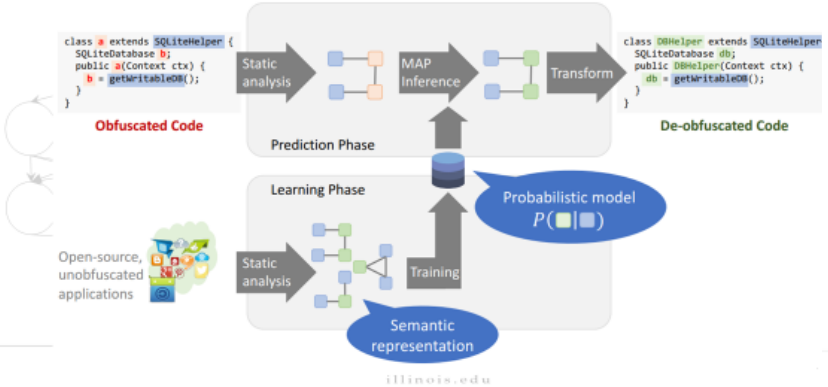
Program Aliasing

omain	NL Specification	Program 1	Program 2
Regex	Match lines that start with an uppercase vowel and end with 'X'	([AEIOUaeiou]&[A-Z]).*X	([AEIOU].*)&(.*X)
Bash	Rename file 'f1' to 'f1.txt'	mv 'f1' 'f1.txt'	cp 'f1' 'f1.txt'; rm 'f1'
Python	Assign the greater value of 'a' and 'b' to variable 'c'	c = a if a > b else b	c = [b, a][a > b]

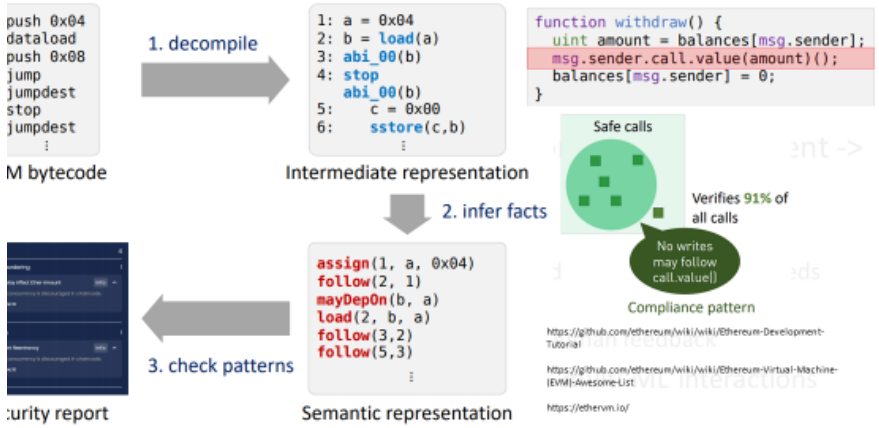
A semantically equivalent program may have various syntactically different forms.

Security

Future Work —— Code Obfuscation/ De-obfuscation/Transformation

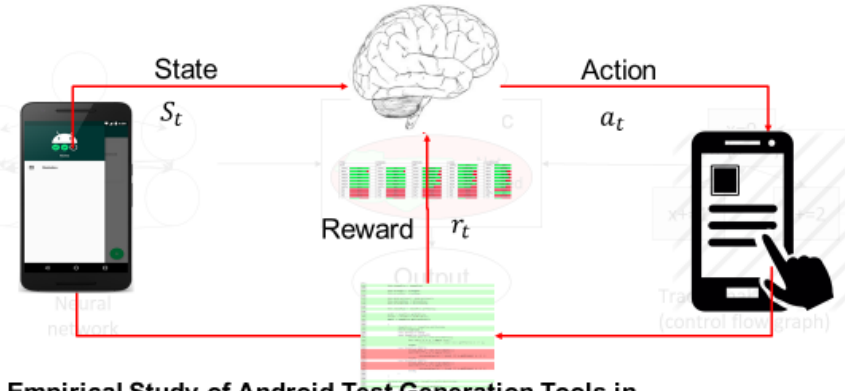


Future work – Smart contract analysis



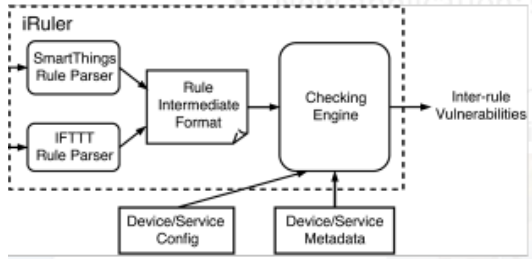
Software Engineering

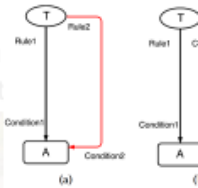
Future Work —— UI testing agent with reinforcement learning



An Empirical Study of Android Test Generation Tools in Industrial Cases
Wang et al. ASE 2018

Future work – IoT App Analysis





iRuler: Detection of Inter-Rule Vulnerabilities in the Internet of T
Wang et al.

Thanks!

wei.yang@utdallas.edu

http://youngwei.com/