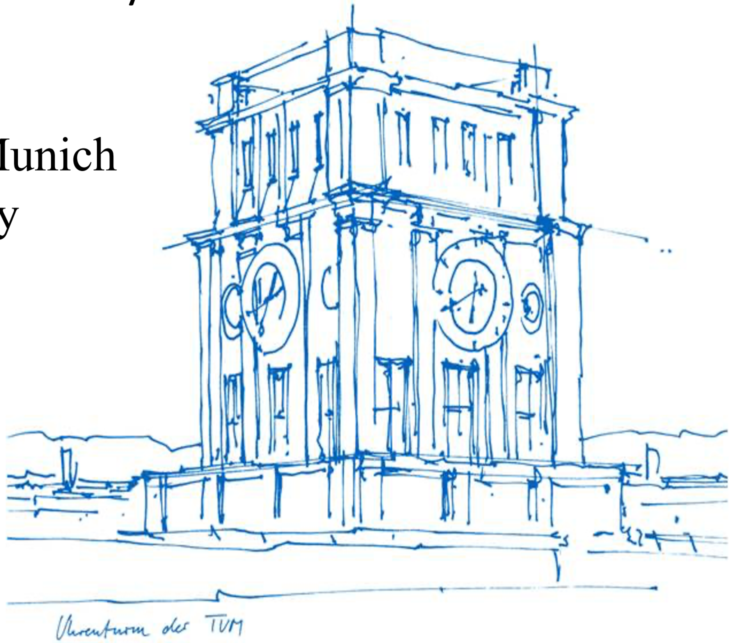
MANIS: Evading Malware Detection System on Graph Structure

Peng Xu[1], Bojan Kolosnjaji[1], Claudia Eckert[1], Apostolis Zarras[2]
{peng, kolosnjaji, eckert}@sec.in.tum.de,
apostolis.zarras@maastrichtuniversity.nl

[1] Technical University of Munich
[2] Masstricht University

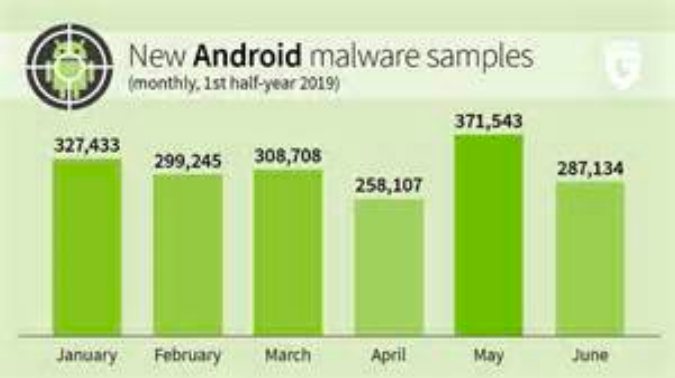


Motivation

- **Smartphones** are now a basic life necessity
- **Android** is the world‘s dominant mobile operating system
- According to **McAfee**, the number of discovered **Android malware** has touched **2.5 millions** in 2017, which led the overall mobile malware‘s tally to reach **25 millions**

Motivation

- **Smartphones** are now a basic life necessity
- **Android** is the world's dominant mobile operating system
- According to **McAfee**, the number of discovered **Android malware** has touched **2.5 millions** in 2017, which led the overall mobile malware's tally to reach **25 millions**

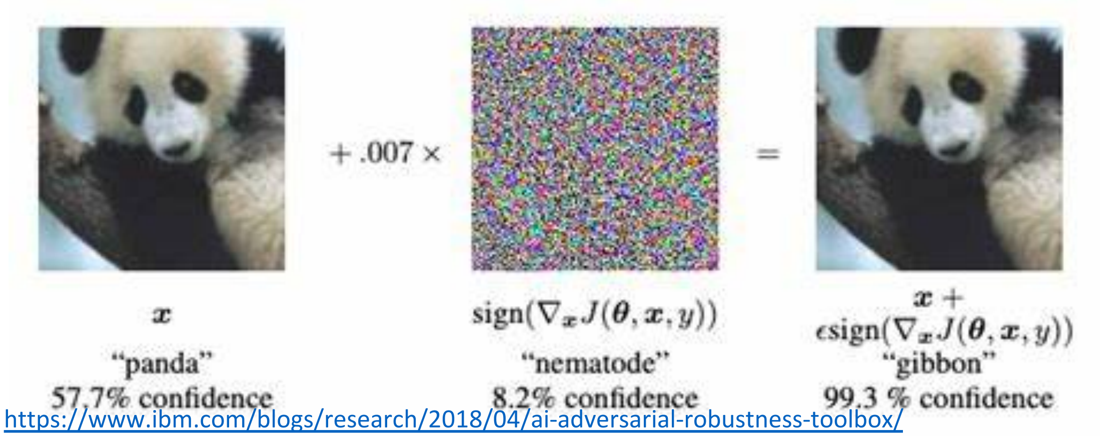


Motivation

- **Smartphones** are now a basic life necessity
- **Android** is the world‘s dominant mobile operating system
- According to **McAfee**, the number of discovered **Android malware** has touched **2.5 millions** in 2017, which led the overall mobile malware‘s tally to reach **25 millions**
- Existing Android Malware Detection techniques:
  - Signature-based and code matching techniques are obsolete
  - Context-based machine learning approaches are not adequate

Motivation

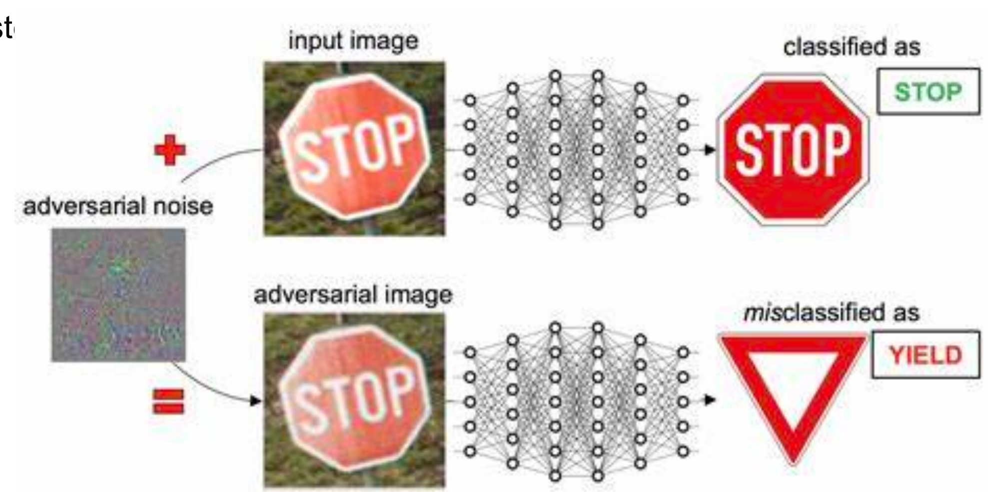
- **Adversarial Machine learning**
  - Attack the machine learning system
- **Poisoning attack**
  - To the training step
- **Evasion attack**
  - To the testing step： Fast gradient sign method(FGSM), Jacobian Saliency Map Approach(JSMA)



https://www.ibm.com/blogs/research/2018/04/ai-adversarial-robustness-toolbox/

Motivation

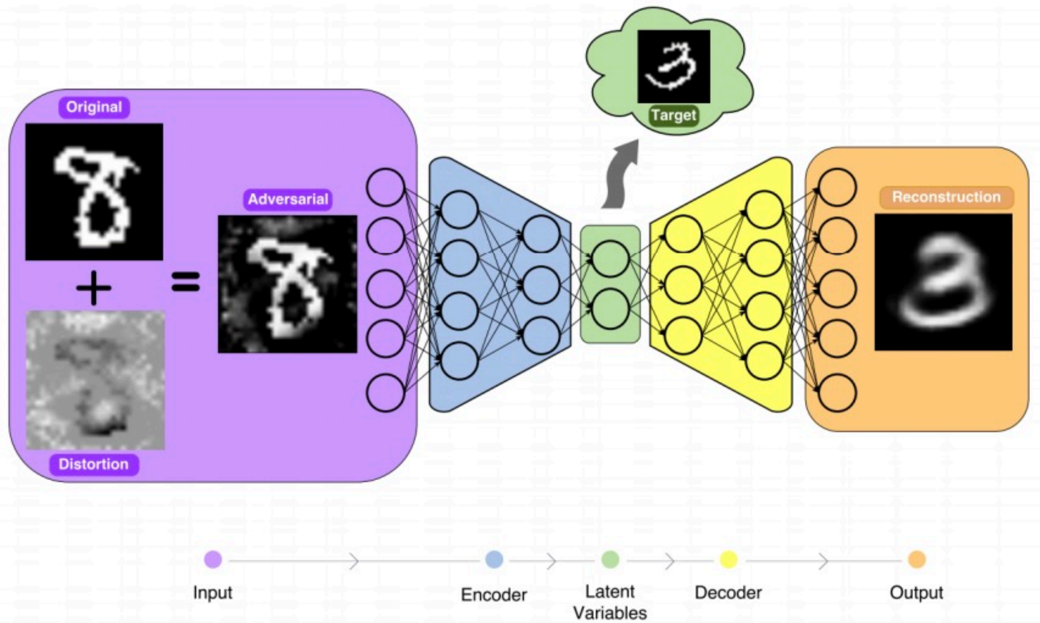
- **Adversarial Machine learning**
  - Attack the machine learning system
- **Poisoning attack**
  - To the training step
- **Evasion attack**
  - To the testing st … oach(JSMA)



https://blog.csiro.au/vaccinating-machine-learning-against-attacks/

Motivation

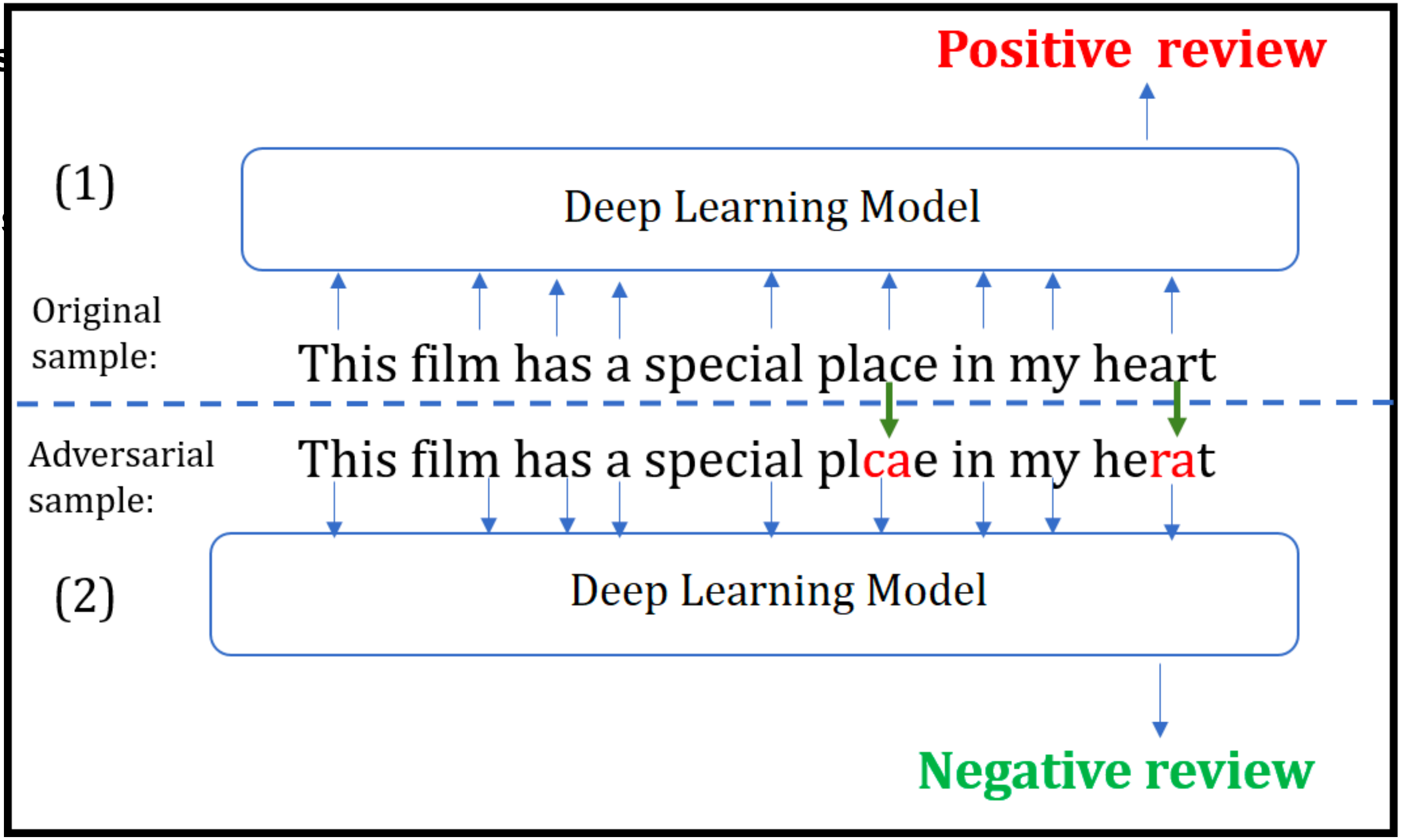
- **Adversarial Machine learning**
  - Attack the machine learning system
- **Poisoning attack**
  - To the training step
- **Evasion attack**
  - To the testing step



https://arxiv.org/pdf/1712.07107.pdf

Motivation

- **Adversarial Machine learning**
  - Attack the machine learning system
- **Pois**
  -
- **Eva**
  -



https://qdata.github.io/secureml-web/4VisualizeBench/

Motivation

Motivation

- **Robustness of ML-based malware detection under adversarial noise**

Motivation

- **Robustness of ML-based malware detection under adversarial noise**
- **Crafting the adversarial noise**

Motivation

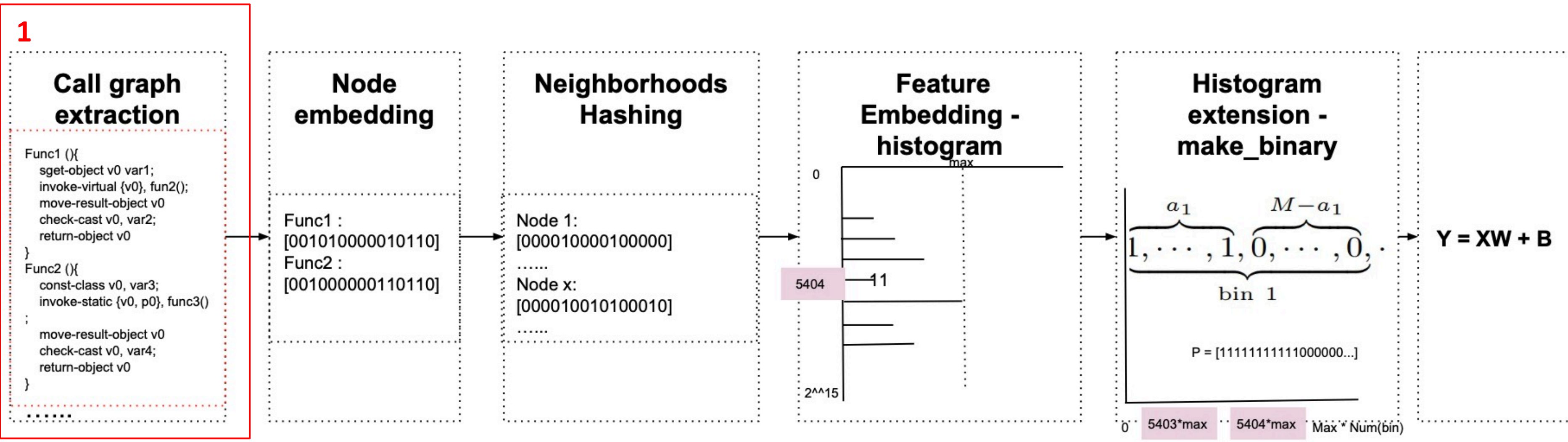
- **Robustness o**
- **Crafting the a**

oise

Android Malware Detection on Graph Structure

- **Manifest**
  - **Permission**
- **Structural Information**
  - **Control Flow Graph(CFG)**
  - **Function Call Graph(FCG)**
  - **Program Dependence Graph (PDG)**

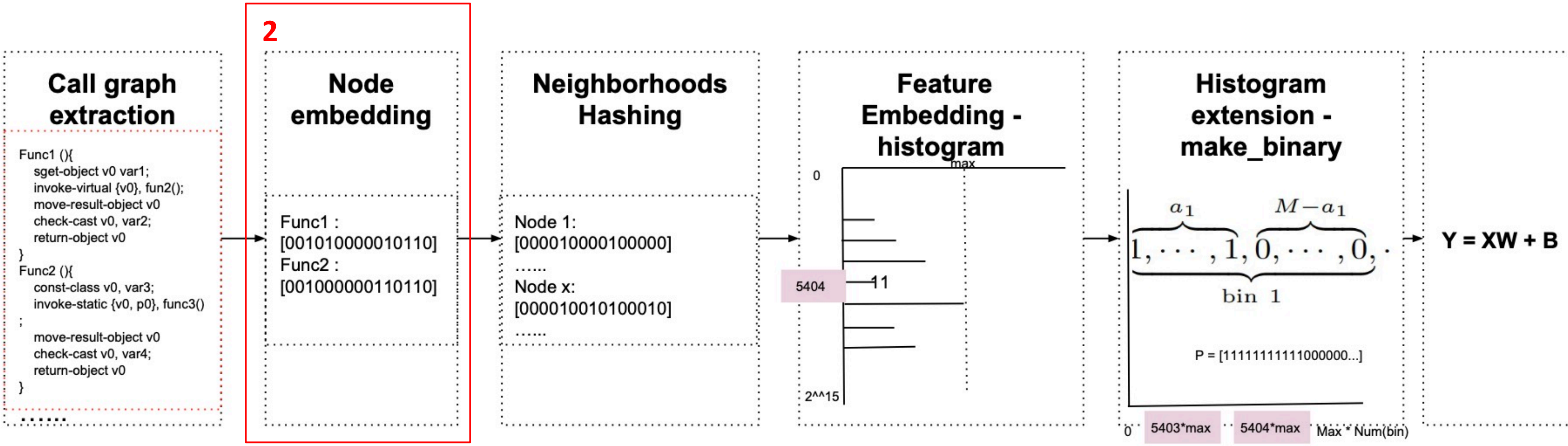
Android Malware Detection on Graph Structure



14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

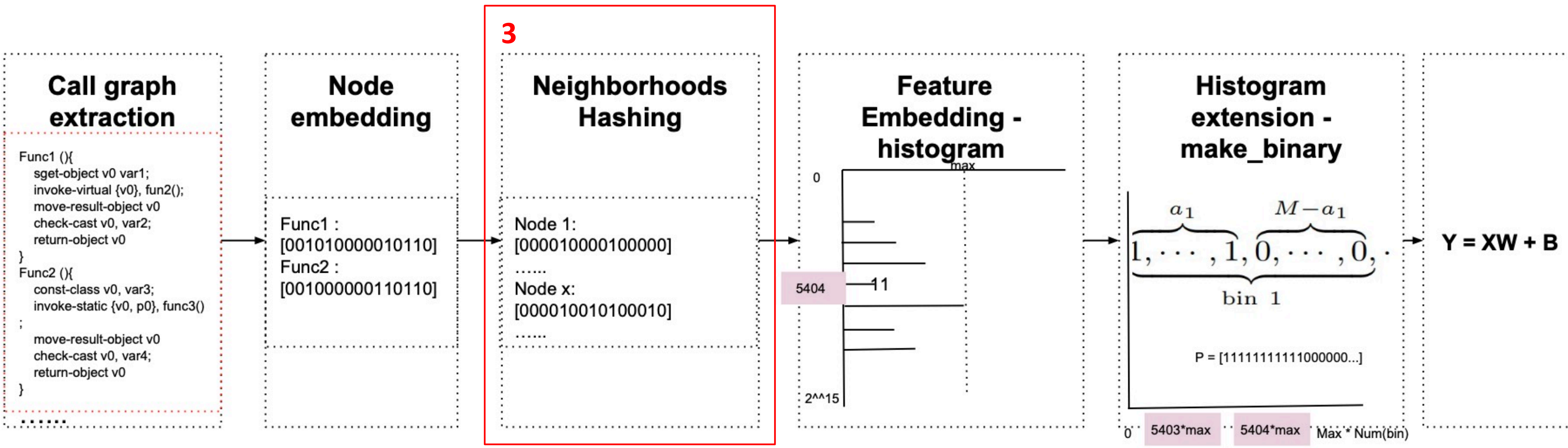
Android Malware Detection on Graph Structure



14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

Android Malware Detection on Graph Structure



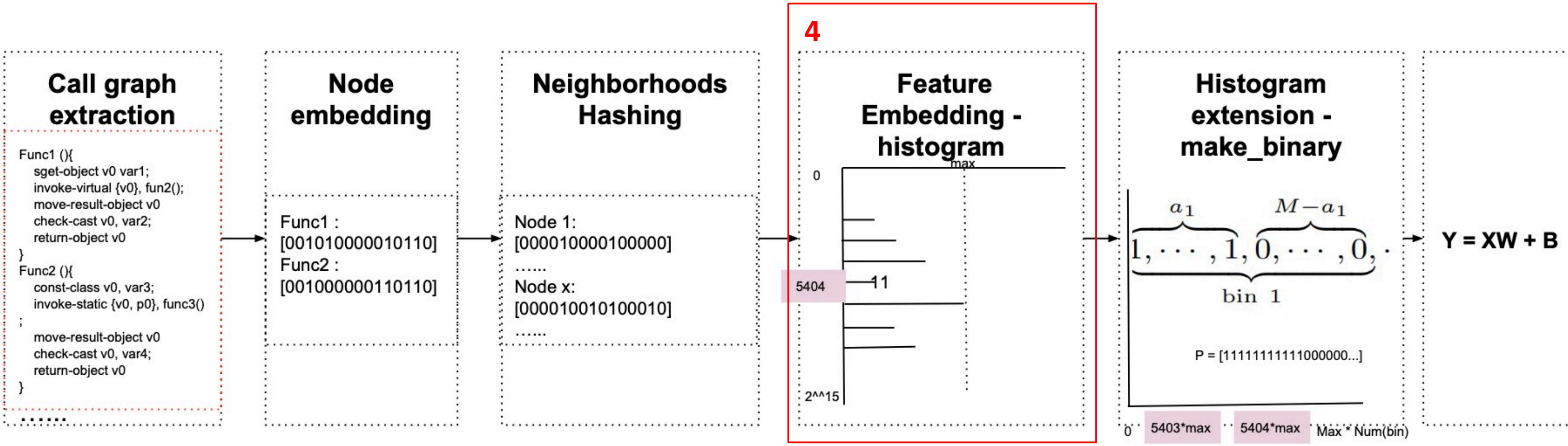
14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

$$h(v) = r(\ell(v)) \oplus \left(\bigoplus_{z \in V_v} \ell(z) \right)$$

http://www.prosec-project.org/docs/2013b-aisec.pdf

Android Malware Detection on Graph Structure

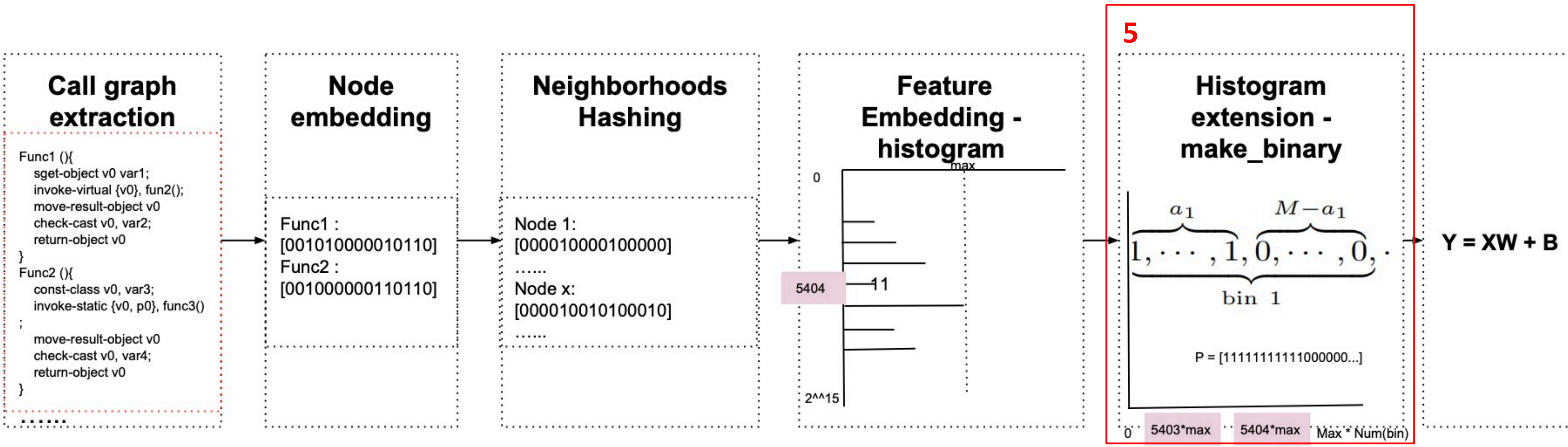


14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

5404 -> [001-0101-0001-1100]

Android Malware Detection on Graph Structure

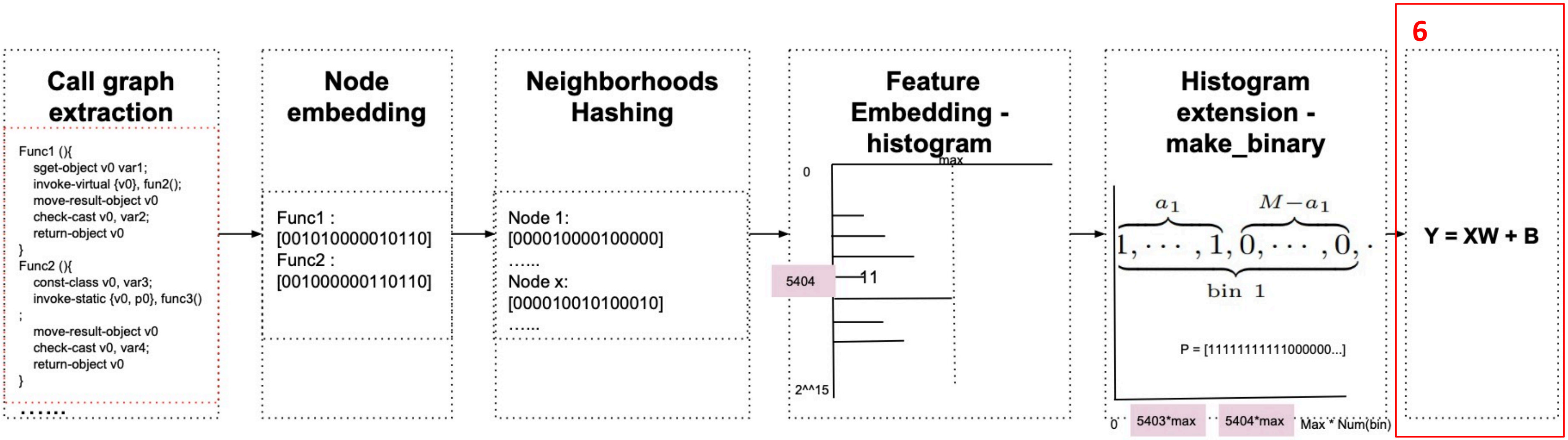


14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

$\{[11111111111]_{11}[0]_{(max*5404 - 11)}$

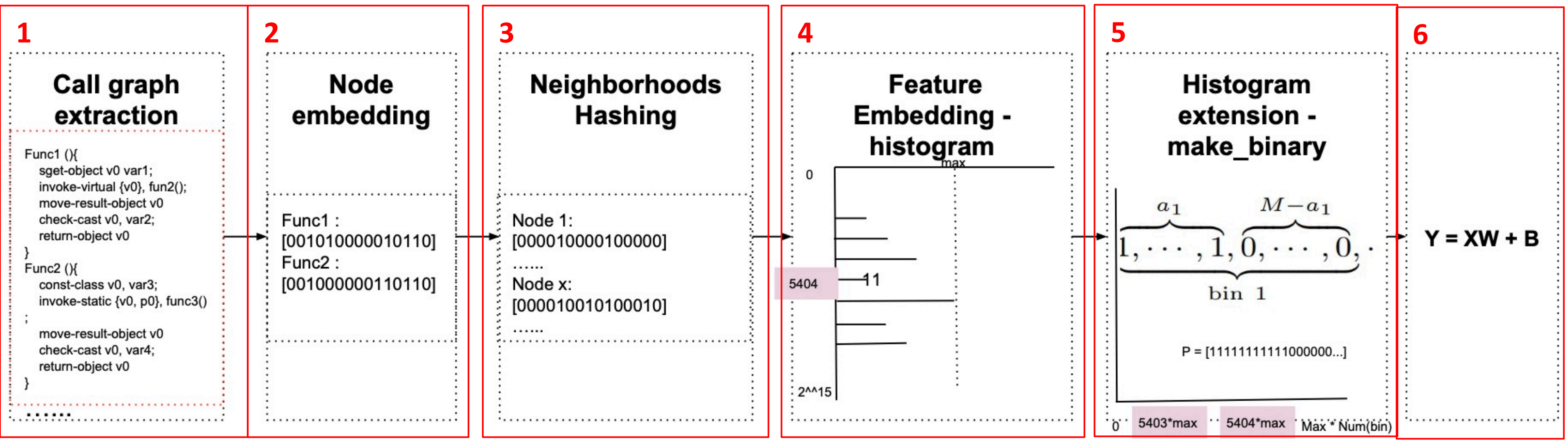
Android Malware Detection on Graph Structure



14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

Android Malware Detection on Graph Structure



14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binop	unop	invoke	staticop	instanceop	arrayop	branch	jump	throw	new	test	monitor	return	move	nop

Figure 1: Adagio malware detection. Top: Detection system includes six steps. Botton: 15-Dalvik instruction categories.

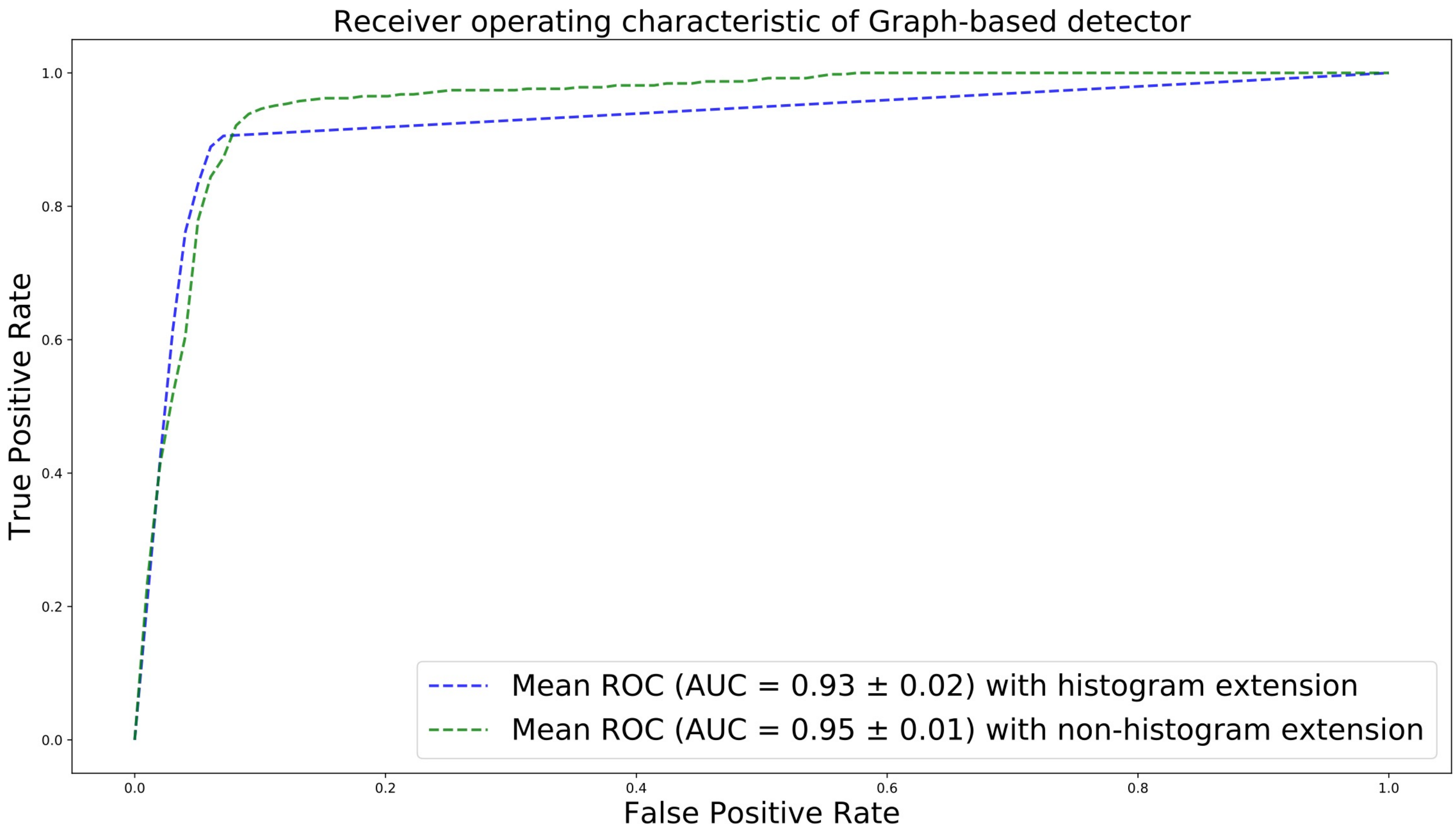
5404 -> [001-0101-0001-1100]

$$h(v) = r(\ell(v)) \oplus \left(\bigoplus_{z \in V_v} \ell(z) \right)$$

$\{[11111111111]_{11}[0]_{(max*5404-11)}$

http://www.prosec-project.org/docs/2013b-aisec.pdf

Android Malware Detection on Graph Structure



Evading Malware Detection on Graph Structure

- ## Mathematic Model
  - **Detection: Y = X*W + B = f(G)*W + B**
  - **Loss Function:** $Loss = 1/N \sum_{i=1}^{N} Loss(f(G_i), y_i),$
  - **Adversarial noise:** $|G_i^* - G_i| < \mathcal{B}$

$$\max_{G^*} Loss^*$$

$$Loss^* = 1/N \sum_{i=1}^{N} Loss(f(G_i^*), y_i)$$

$$G_i^* = G_i + \alpha * \xi^{adv}(G_i),$$

- ## N-strongest Nodes
  - **Finding the nodes which has the largest influence**
  - **Injecting those nodes multiple times**
- ## Gradient-Based Approach
  - **Gradient computation**
  - **Direction vector**

Evading Malware Detection on Graph Structure

- **N-strongest Nodes**
  - **Finding the nodes which has the largest influence**
  - **Injecting those nodes multiple times**
- **Gradient-Based Approach**
  - **Using gradient’s direction**

Evading Malware Detection on Graph Structure

- **N-strongest Nodes**
  - **Initialization: Prepare the weights and find the node(s) which have the minimum weight value**
  - **Injection operation**
    - **Boolean Representation of the n-strongest node(s)**
    - **Injecting the bool representation of n-strongest node(s) at neighborhood hashing step**
    - **Feature embedding histogram with the injected n-strongest node(s)**
    - **Histogram and non-histogram extension with the injected n-strongest node(s)**
  - **Classifier**
    - $|G_i - G^*_i| < \mathcal{B}$ **and** $Y = -1$

Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements:**
    - **R1: The occurrence of graph's node cannot be expressed less than zero**
    - **R2: In histogram extension mode, all "1" should align at the beginning of P-dimensional vector**
    - **R3: Cannot reduce the original occurrence of graph's nodes in order to keep the functionalities**

Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector**

Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector**

$$\min_{n} f(G_h)$$

$$s.t. d(G_h, G'_h) \leq m,$$

$$f(X) = (X * W + B - Y)^2,$$

Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector**

$$\min_{n} f(G_h)$$

$$s.t. d(G_h, G'_h) \leq m,$$

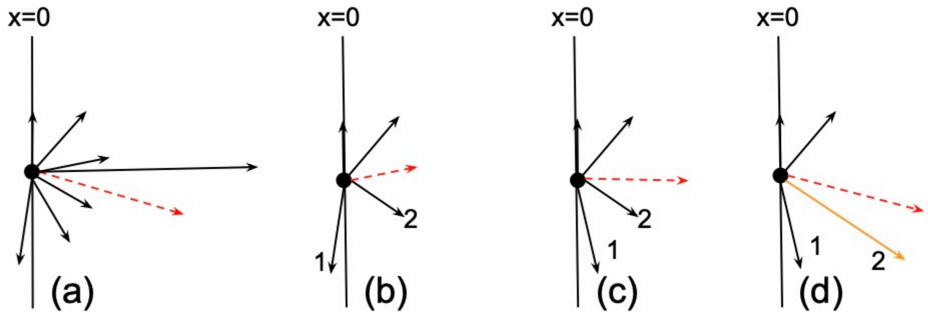
$$f(X) = (X * W + B - Y)^2,$$

$$\nabla f(X)/X$$

Evading Malware Detection on Graph Structure

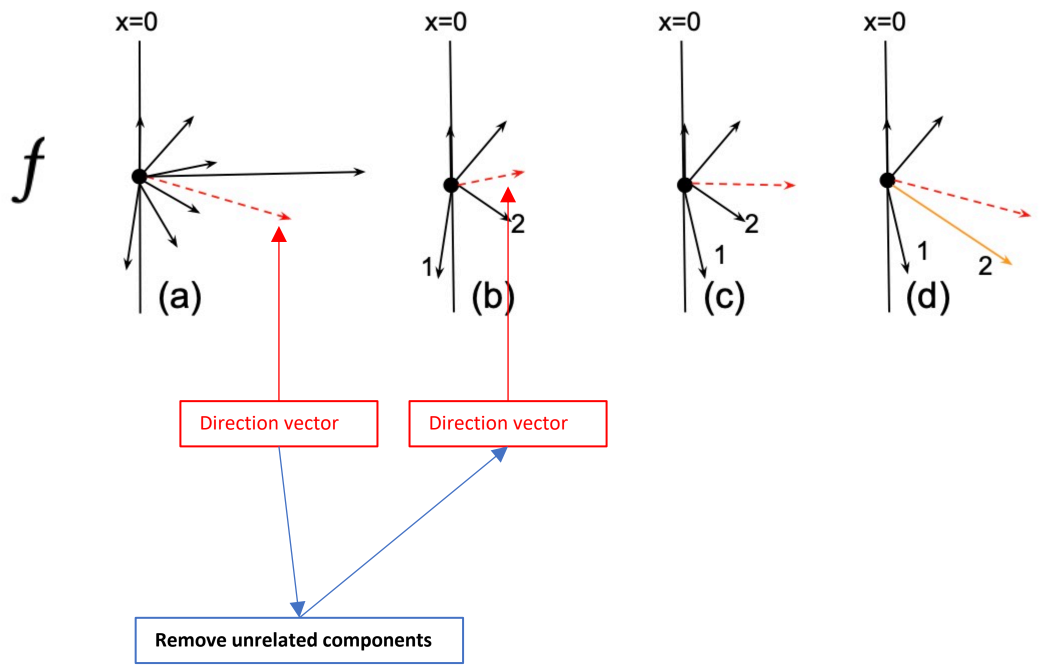
- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector** $$\nabla f(X)/X$$
    - **Gradient vector adjusting**

f



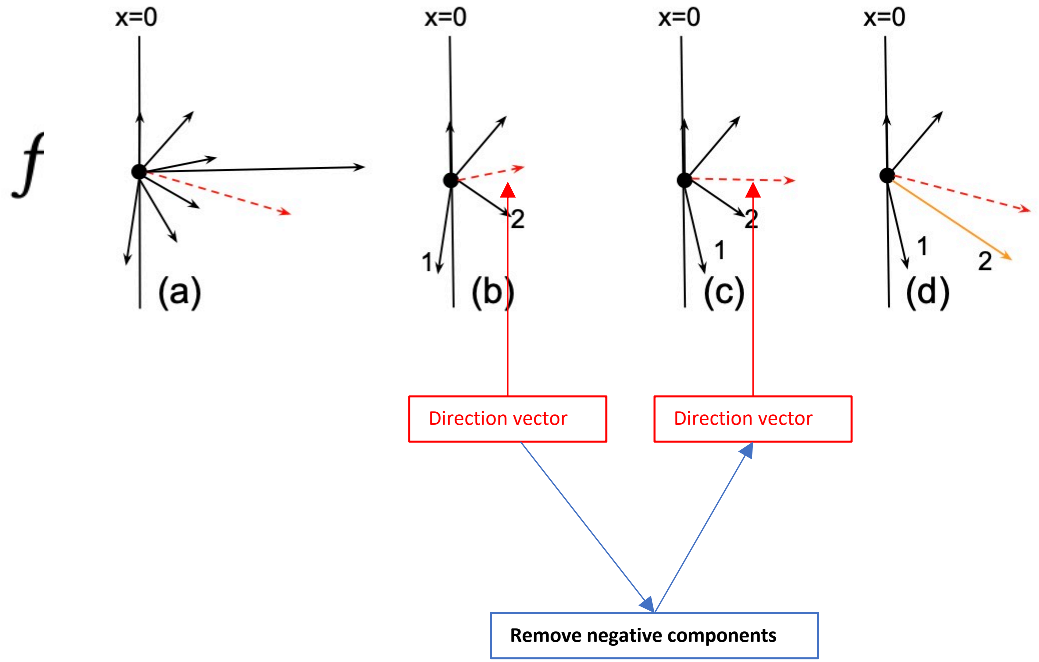
Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector** $\nabla f(X)/X$
    - **Gradient vector adjusting**
    - **Node Projection**



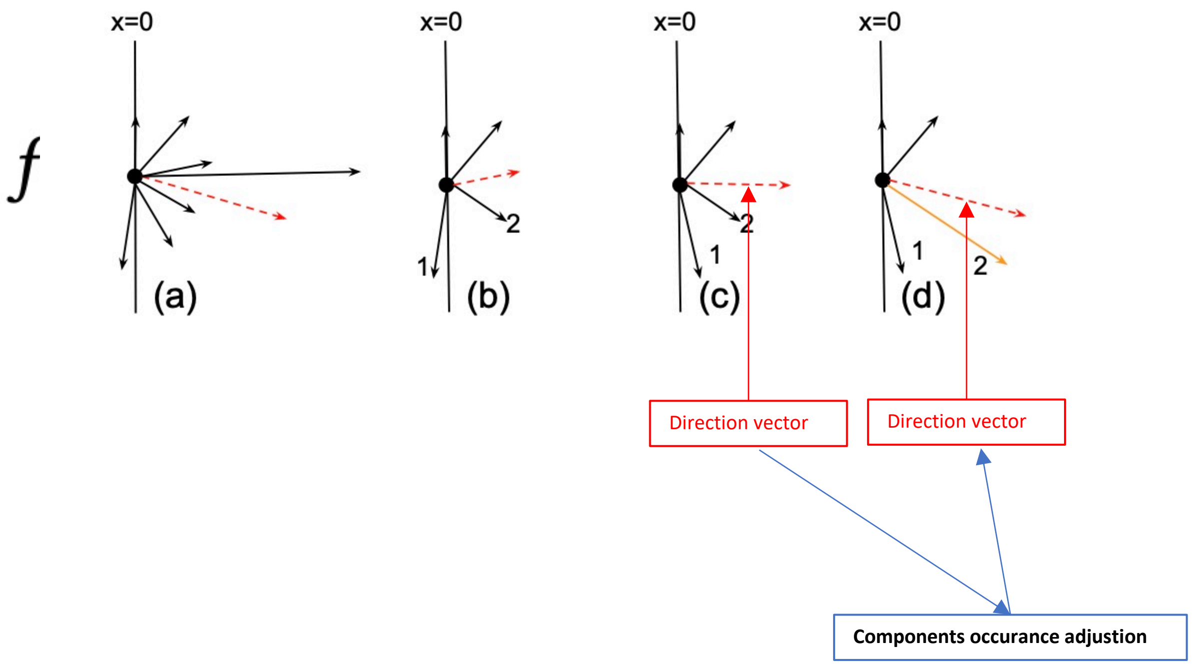
Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector** $\nabla f(X)/X$
    - **Gradient vector adjusting**
    - **Node Projection**



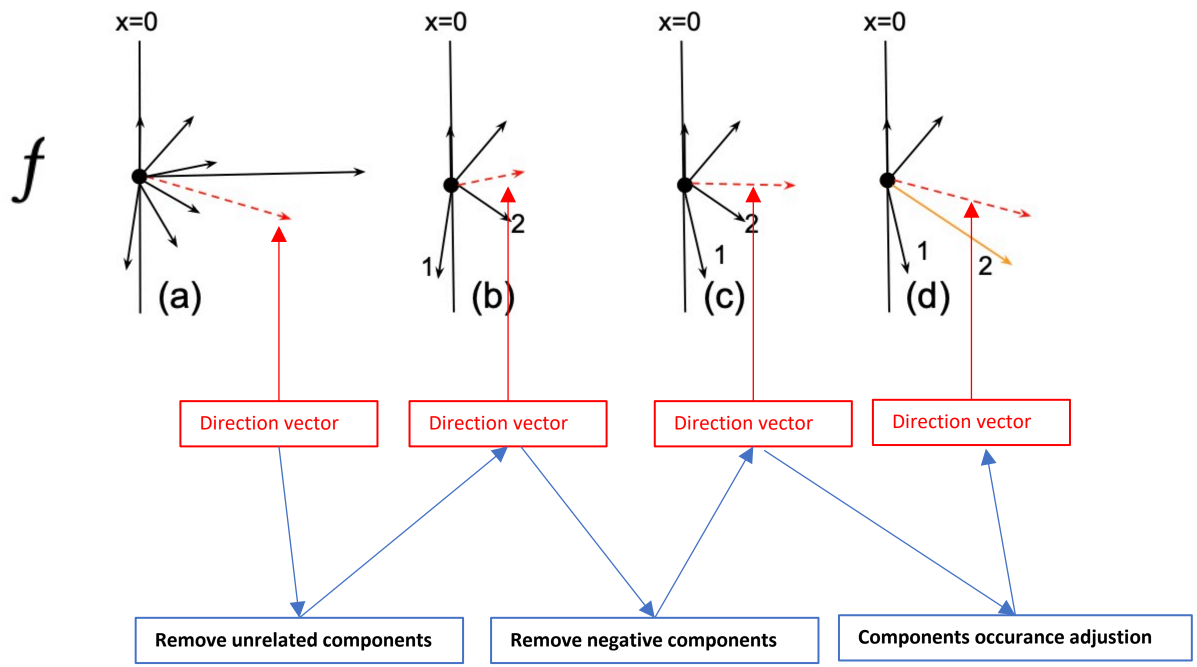
Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector** $\nabla f(X)/X$
    - **Gradient vector adjusting**
    - **Node Projection**



Evading Malware Detection on Graph Structure

- **Gradient-based Approach**
  - **Requirements**
  - **Crafting methods**
    - **Gradient Computation : direction vector** $\nabla f(X)/X$
    - **Gradient vector adjusting**
    - **Node Projection**



Evading Malware Detection on Graph Structure

Gradient-based Approach

- **Requirements**
- **Crafting methods**
  - **Gradient Computation : direction vector**
  - **Gradient vector adjusting**
  - **Node Projection**

$$[1, 2, 1, 1, -1, 0, 0, -1, 2, -4, \ldots, 1, -4, 2, \ldots]_{250}$$

R1

$$[1, 1, 1, 1, 1, 0, 0, 1, 1, 1, \ldots, 1, 1, 1, \ldots]_{250}$$

R2

$$[\mathbf{1,1,1,1,1}, \mathit{1,1,1,1,1}, 0, 0, 0, \ldots]_{250}$$

Evaluation

- **Dataset**
  - **Benign: 49,947(AndroZoo + VirusTotal)**
  - **Malware: 5,560(Drebin)**
  - **Five folds**
- **White-box and gray-box**
  - **While-box: access all of information**
  - **Gray-box: can access limited information for the targeting classifier, but can get other classifier's information, which trained by other folds**
- **Histogram extension and Non-histogram extension**
  - **Including the histogram extension or not**

Evaluation

- **Strongest nodes Number: 1, 2, 3, 4, 5**
- **White-box and gray-box**

Table 1: N-strongest nodes (non-histogram extension)

	Non-histogram extension(white-box)					Non-histogram extension(gray-box)				
Strongest nodes	1	2	3	4	5	1	2	3	4	5
Injected nodes ratio ($\overline{x}$)	22.7%	23.7%	17.6%	15.6%	25.7%	20.69%	21.25%	32.26%	17.76%	18.98%
Injected nodes ratio (σ)	5.8%	22.3%	22.7%	22.1%	22.4%	5.19%	20.67%	27.93%	30.76%	32.87%
Misclassified Rate ($\overline{x}$)	72.2%	26.7%	26.6%	32.8%	40.8%	80.79%	49.43%	35.62%	21.14%	19.8%
Misclassified Rate (σ)	15.4%	28.4%	34.1%	43.7%	42.5%	4.9%	42.9%	25.9%	25.6%	26.8%

- **Results:**
  - **Injecting 22.7%(20.69%) one strongest node will get 72.2%(80.79%) misclassification with white-box(gray-box) setting**
  - **Injecting around 23% and 17% two- and three-strongest nodes will get 26% misclassification with white box, and 21.25%,32.26% injection two- and three nodes with get 49.43% and 35.62% misclassification**
  - **Misclassification rates depend on the number of n-strongest nodes**

Evaluation

- **Strongest nodes Number: 1, 2, 3, 4, 5**
- **White-box and gray-box**

Table 2: N-strongest nodes (histogram extension)

	histogram extension (white-box)					histogram extension(gray-box)				
Strongest nodes	1	2	3	4	5	1	2	3	4	5
Injected nodes ratio ($\overline{x}$)	24.30%	37.27%	18.51%	46.95%	40.24%	23.68%	22.38%	14.41%	25.88%	23.35%
Injected nodes ratio (σ)	8.6%	8.2%	10.4%	11.3%	19.1%	24.94%	17.04%	13.17%	15.13%	14.52%
Misclassified Rate ($\overline{x}$)	6.01%	29.77%	6.97%	35.81%	17.65%	5.46%	21.33%	4.39%	4.70%	5.03%
Misclassified Rate (σ)	1.89 %	22%	3.3%	27%	14%	3.71%	29.31%	1.21%	0.82%	0.77%

- **Results:**
  - **Misclassification rates with histogram extension are significantly lower than non-histogram extension**
  - **Injecting two strongest nodes with 37.37%(21.33%) will cause around 30%(21.22%) misclassification with white-box(gray-box) setting**
  - **White-box attacking gets better misclassification than gray-box setting**

Evaluation

- **Gradient sign method**
- **Parameter:** α $G_i^* = G_i + \alpha * \xi^{adv}(G_i)$
- *threshold* limit the number of injected nodes
- **White-box and gray-box**

Table 3: Gradient sign method (non-histogram extension)

α	0.1			0.2			0.3		
threshold	< 0.1	<= 0.1	<= 0.2	< 0.2	<= 0.2	<= 0.4	< 0.3	<= 0.3	<= 0.6
	non-histogram extension (white-box)								
Injected nodes ratio ($\overline{x}$)	56.5x	17.48%	17.06%	51.8x	15.46%	14.82%	46.9x	14.22%	13.89%
Injected nodes ratio (σ)	1.98	1.1%	1.1 %	2.74	1%	0.6%	59.74 %	1.4%	0.5%
Misclassified rate ($\overline{x}$)	79.6%	45.3%	39.68%	94.21%	47.46%	43.69%	98.07%	41.20%	42.17%
Misclassified rate (σ)	5.2%	5.6%	1.9%	0.6%	3.7%	6.7%	2.6%	7%	0.2%
	non-histogram extension (gray-box)								
Injected nodes ratio ($\overline{x}$)	59.62x	15.61%	15.35%	48.57x	14.72%	14.7%	50.09x	15.07%	15.26%
Injected nodes ratio (σ)	4.71	1.18%	0.78 %	3.48	0.99%	0.98%	1.89	1.55%	1.86%
Misclassified rate ($\overline{x}$)	79.76%	37.59%	38.95%	96.06%	43.69%	44.18%	97.43%	44.65%	43.05%
Misclassified rate (σ)	3.78%	7.95%	8.29%	2.08%	8.11%	8.23%	1.33%	8.43%	8.07%

Evaluation

Table 3: Gradient sign method (non-histogram extension)

α	0.1			0.2			0.3		
threshold	< 0.1	<= 0.1	<= 0.2	< 0.2	<= 0.2	<= 0.4	< 0.3	<= 0.3	<= 0.6
	non-histogram extension (white-box)								
Injected nodes ratio ($\overline{x}$)	56.5x	17.48%	17.06%	51.8x	15.46%	14.82%	46.9x	14.22%	13.89%
Injected nodes ratio (σ)	1.98	1.1%	1.1 %	2.74	1%	0.6%	59.74 %	1.4%	0.5%
Misclassified rate ($\overline{x}$)	79.6%	45.3%	39.68%	94.21%	47.46%	43.69%	98.07%	41.20%	42.17%
Misclassified rate (σ)	5.2%	5.6%	1.9%	0.6%	3.7%	6.7%	2.6%	7%	0.2%
	non-histogram extension (gray-box)								
Injected nodes ratio ($\overline{x}$)	59.62x	15.61%	15.35%	48.57x	14.72%	14.7%	50.09x	15.07%	15.26%
Injected nodes ratio (σ)	4.71	1.18%	0.78 %	3.48	0.99%	0.98%	1.89	1.55%	1.86%
Misclassified rate ($\overline{x}$)	79.76%	37.59%	38.95%	96.06%	43.69%	44.18%	97.43%	44.65%	43.05%
Misclassified rate (σ)	3.78%	7.95%	8.29%	2.08%	8.11%	8.23%	1.33%	8.43%	8.07%

- **Results:**
  - **More than 1x injection ratio should be removed**
  - **With threshold <=0.1 and <=0.2, we get around 40% misclassification ratio with 17%~14% node injection with white-box and gray-box**

Evaluation

- **N: the different number of adjusted occurrence of a node**
- **White-box and gray-box**

Table 4: Gradient sign method (histogram extension)

	histogram extension (white-box)					histogram extension(gray-box)				
N	1	2	3	4	5	1	2	3	4	5
Injected nodes ratio ($\overline{x}$)	10.47%	22.05%	35.40%	35.42%	36.3%	9.52%	21.01%	33.11%	33.11%	33.11%
Injected nodes ratio (σ)	0.04%	2.23%	0.22%	1.49%	2.04%	0.4%	0.8%	1.05%	1.05%	1.05%
Misclassified rate ($\overline{x}$)	21.12%	20.56%	22.3%	36.30%	33.49%	18.79%	18.79%	18.79%	18.79%	18.79%
Misclassified rate (σ)	5.8%	6.52%	6.63%	8.21%	11.93%	2.04%	2.04%	2.04%	2.04%	2.04%

- **Results:**
  - **Misclassification rates with histogram extension are significantly lower than non-histogram extension**
  - **Misclassification rates are around 20% with 10%~30% node injection with adjusted 1,2,3 nodes with white-box setting**
  - **Misclassification rates are 18.8% with different injected number under gray-box setting**
  - **White-box attacking gets better misclassification than gray-box setting**

Evaluation

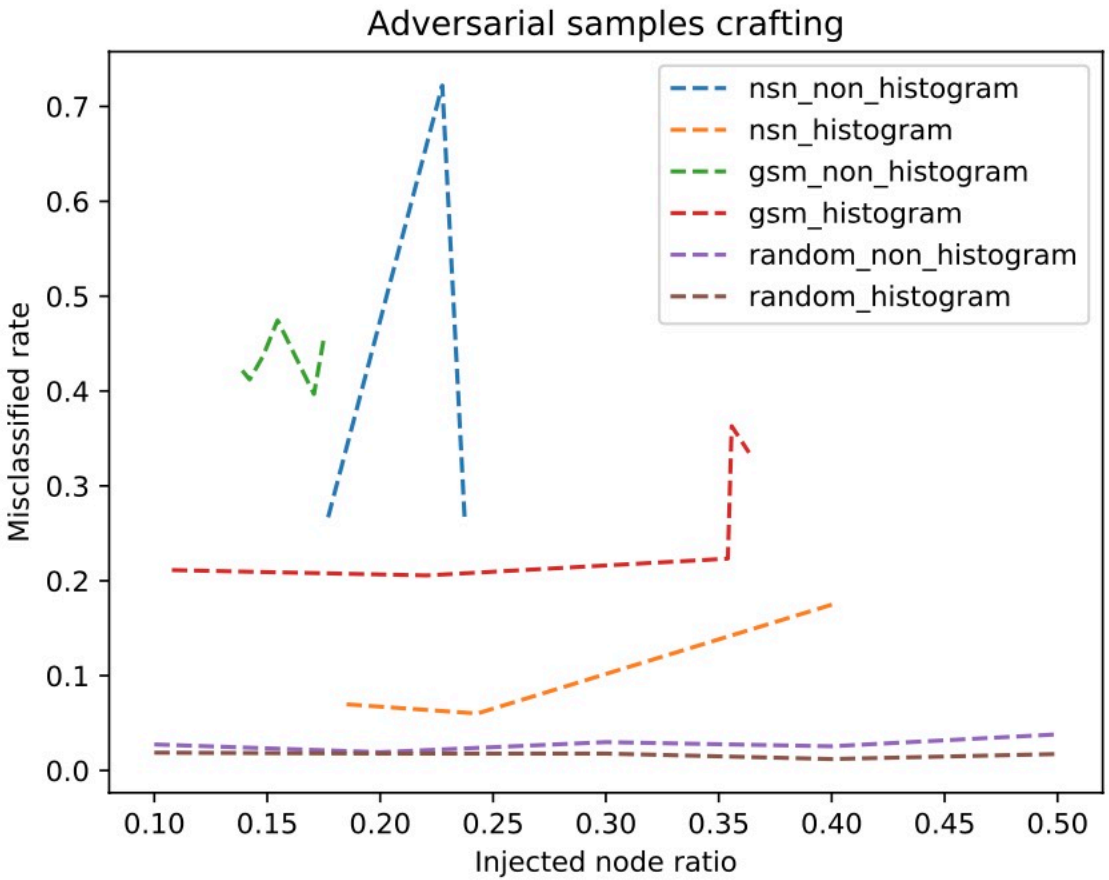
- **Randomly selected nodes injections**
- **White-box and gray-box**

Table 5: Randomization method.

	non-histogram extension (white-box)					histogram extension(gray-box)				
Injected nodes ratio ($\overline{x}$)	10%	20%	30%	40%	50%	10%	20%	30%	40%	50%
Misclassified rate ($\overline{x}$)	2.74%	1.92%	2.98%	2.55%	3.81%	1.87%	1.78%	1.78%	1.2%	1.73%
Misclassified rate (σ)	0.93%	0.85%	1.5%	1.09%	1.83%	1.31%	0.99%	1.25%	0.73%	1.13%

- **Results:**
  - **Misclassification rates randomly selected node injected are very low, from 1.9%~3.8% with white-box setting, and from 1.2%~1.87% with gray-box setting**
  - **The injected nodes ratios, from 10%~50%, do not affect misclassification rate significantly**

Evaluation



Adversarial samples crafting
Misclassified rate
Injected node ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
nsn_non_histogram
nsn_histogram
gsm_non_histogram
gsm_histogram
random_non_histogram
random_histogram

Conclusion

- **Android Malware detection on graph structure**
- **Adversarial example crafting for Android malware detection**
  - **N-Strongest nodes**
  - **Gradient-sign method**
- **Limitation**
  - **Only evaluate for the call function graph**
  - **Graph kernel-hashing embedding**