CE 815 – Secure Software Systems

ML-Based Vulnerability Detection Methods (Hoppity)

Mohammad Haddadian/Mehdi Kharrazi
Department of Computer Engineering
Sharif University of Technology

Acknowledgments: Some of the slides are fully or partially obtained from other sources. A reference is noted on the bottom of each slide, when the content is fully obtained from another source. Otherwise a full list of references is provided on the last slide.

Introduction

- Vulnerability detection as first step
- Then, Vulnerability repair

HOPPITY: Learning Graph Transformations to Detect and Fix Bugs in Programs, E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, K. Wang, ICLR 2020.

Problem

Source-code analysis is:

- Undecidable
- Noisy
- Rules are hand written
- Tailored to specific code bases / bug patterns

Javascript Challenges

- Incorrect operators
- Incorrect identifiers
- Accessing undefined properties
- Mishandling variable scopes
- Type incompatibilities

Example

```
function clearEmployeeListOnLinkClick(){
  document.querySelector("a").addEventListener("click",
    function(event){
      document.querySelector("ul").InnerHTML = "";
    }
  );
}
```

(a) `InnerHTML` should have been `innerHTML`.

```
if (matches) {
  return {
    episode: Number(matches.groups.episode),
    hosts: matches.groups.hosts.split(/([,&]+|\sand\s)/).
              map(el => S(el).trim().s)
  };
}
```

(b) Highlighted parentheses should have been removed.

```
module.exports = function (grunt) {
  grunt.initConfig({
    execute: {...}, copy: {...}, checktextdomain: {...}
    wp_readme_to_markdown: {...}, makepot: {...}})
  ...
  grunt.registerTask('default',['wp_readme_to_markdown'
    ,'makepot','execute','checktextdomain'])
};
```

(c) `copy` function should have also been included in the highlighted list.

```
export default {
  computed: {
    level () {
      return dictMap.skillLevel[
        parseInt((this.value === 0 ? 1 : this.value)/20)];
    }
  },...
}
```

(d) `parseInt` should have been removed because `===` implies `this.value` is an integer.

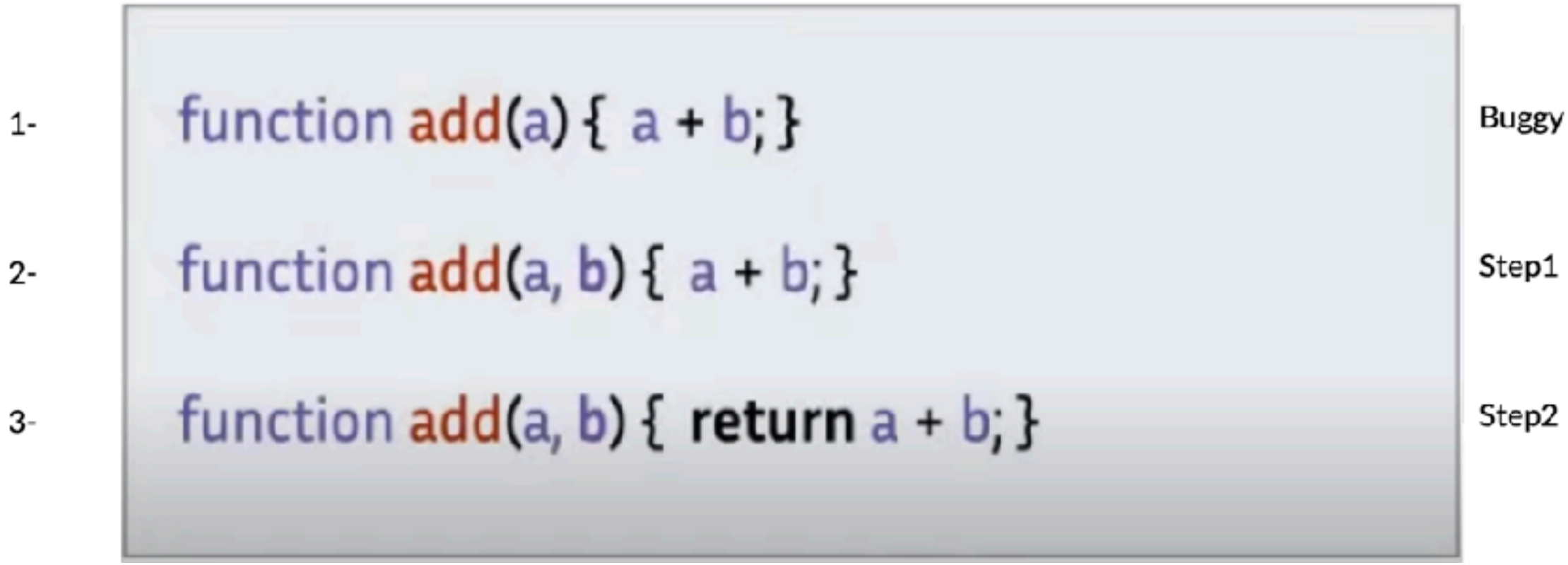
Solution

Leverage large amounts of Javascript fixes on Github to locate and repair bugs

Steps

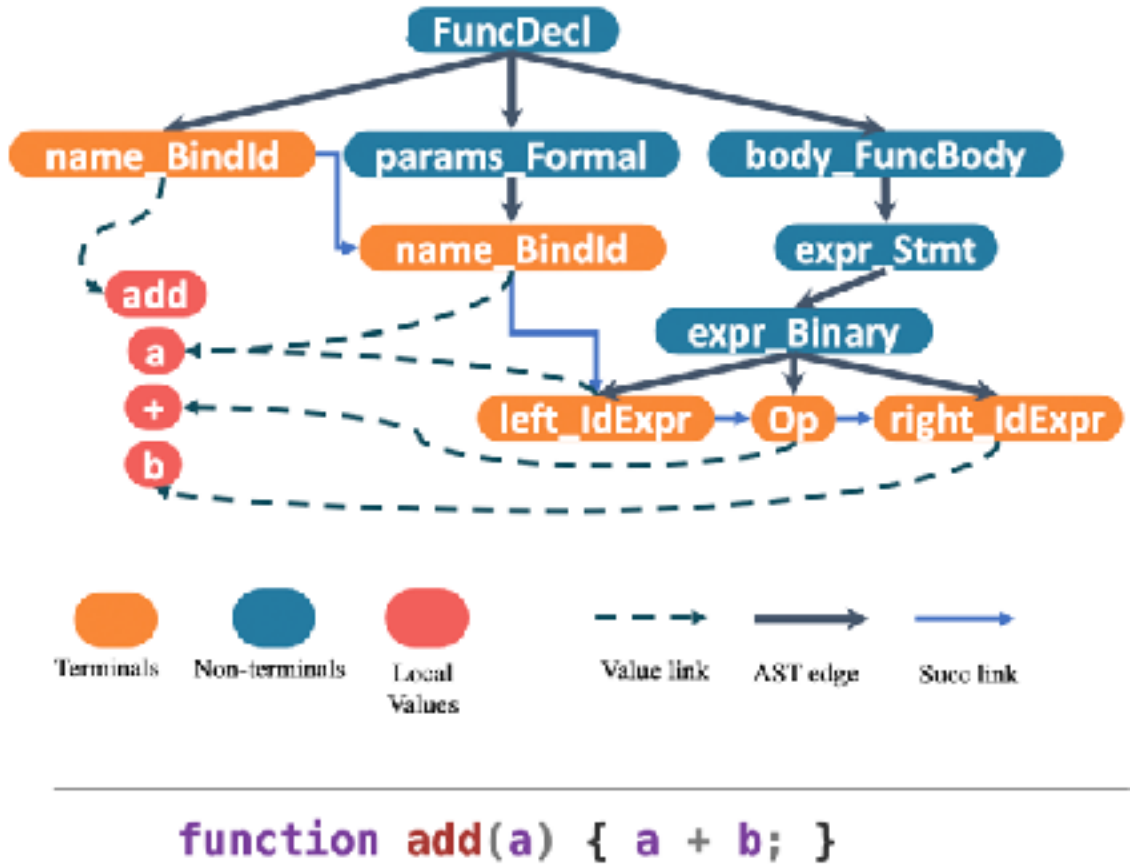
- Represent source code
- Represent fixes
- Learning

Goal


1-
function add(a) { a + b; }
Buggy
2-
function add(a, b) { a + b; }
Step1
3-
function add(a, b) { return a + b; }
Step2

Source code representation

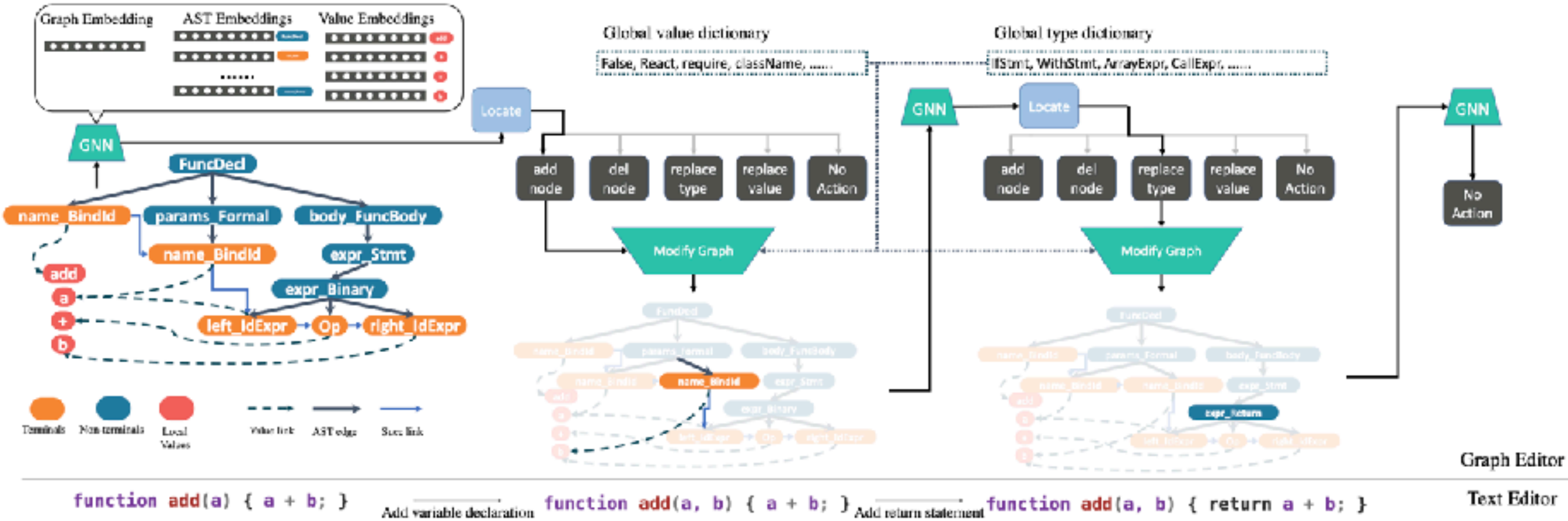
- AST
- ValueLink

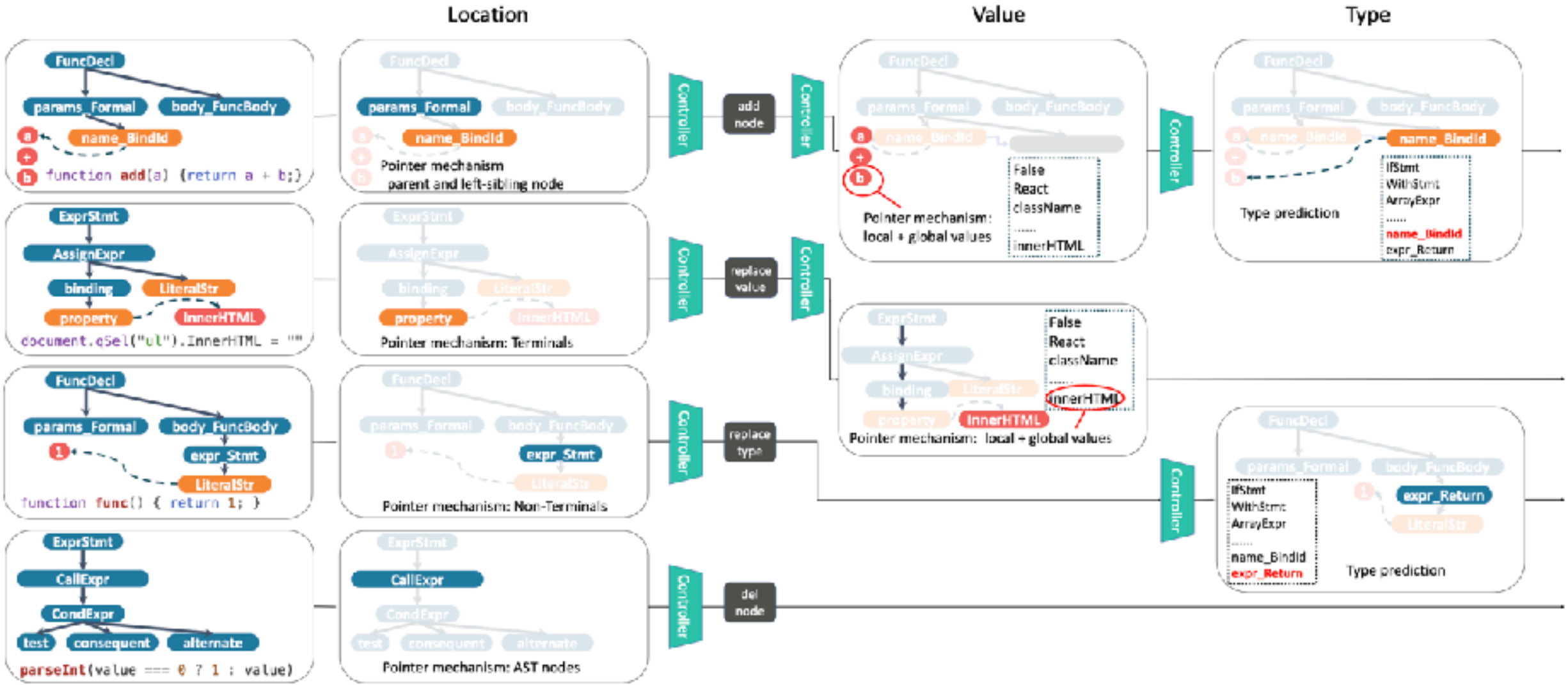



```
function add(a) { a + b; }
```

Fix representation

- Graph Edits




Location
Value
Type
FuncDecl
params_Formal
body_FuncBody
name_BindId
function add(a) {return a + b;}
Pointer mechanism
parent and left-sibling node
add node
Pointer mechanism:
local + global values
False
React
className
innerHTML
Type prediction
IfStmt
WithStmt
ArrayExpr
expr_Return
ExprStmt
AssignExpr
binding
LiteralStr
property
innerHTML
document.qSel("ul").InnerHTML = ""
Pointer mechanism: Terminals
replace value
Pointer mechanism: local + global values
FuncDecl
params_Formal
body_FuncBody
expr_Stmt
LiteralStr
function func() { return 1; }
Pointer mechanism: Non-Terminals
replace type
ExprStmt
CallExpr
CondExpr
test
consequent
alternate
parseInt(value === 0 ? 1 : value)
Pointer mechanism: AST nodes
del node
Controller

[Hoppity]

Low level primitives

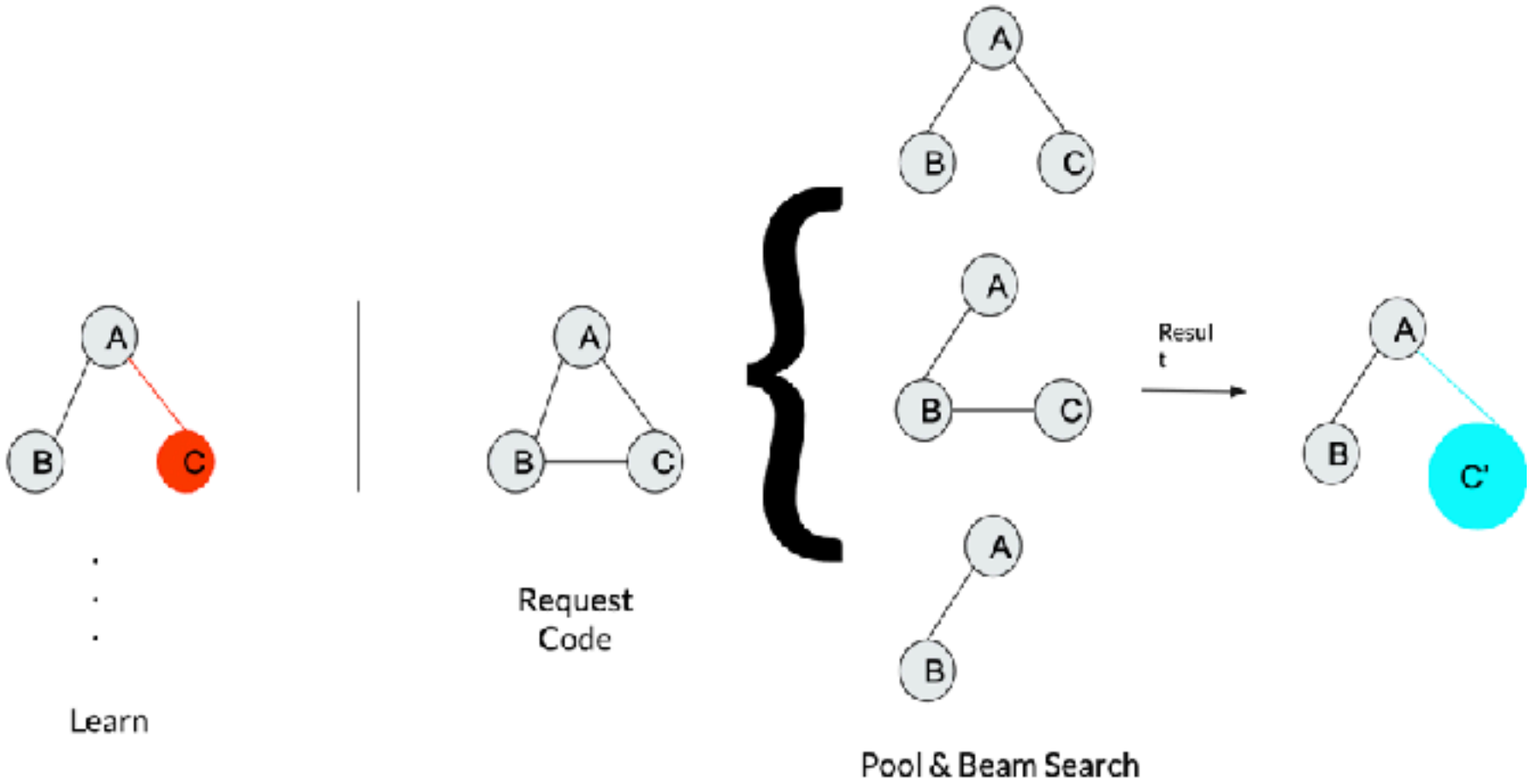
- Location
- Value
- Type

Graph edit operators

Graph transformation

1 2 3

Inference


A
B
C
Learn
A
B
C
Request
Code
A
B
C
A
B
C
A
B
Pool & Beam Search
Resul
t
A
B
C'

Dataset

- OneDiff (just one change)
- ZeroOneDiff (zero or one edit)
- ZeroOneTwoDiff (zero, one or two edits)

	ADD	REP_TYPE	REP_VAL	DEL	total
train	6,473	1,864	251,097	31,281	290,715
validate	790	245	31,357	3,957	36,349
test	796	233	31,387	3,945	36,361

Table 1: Statistic of `OneDiff` dataset. See appendix for more information of other dataset.

Evaluation

	Total		*Location*		*Operator*	*Value*		*Type*	
	Top-3	Top-1	Top-3	Top-1	Top-1	Top-3	Top-1	Top-3	Top-1
TOTAL	**26.1**	14.2	35.5	20.4	34.4	52.3	29.1	76.1	66.7
ADD	52.9	39.2	69.6	51.4	70.6	65.7	55.1	76.8	68.5
REP_VAL	23.4	11.9	33.3	18.5	31.7	53.0	28.8	-	-
REP_TYPE	71.7	52.4	73.0	52.8	79.4	-	-	74.7	61.0
DEL	39.6	24.8	44.0	27.5	45.8	-	-	-	-
Random	.08	.07	2.28	1.4	27.7	.01	.01	.27	0

Table 2: Evaluation of model on the `OneDiff` dataset: accuracy (%).

Evaluation (cont.)

Type	GGNN-Rep	GGNN-Cls	HOPPITY
Top-1	53.2%	**99.6%**	90.0%
Top-3	85.8%	**99.6%**	94.8%

Table 3: `REP_TYPE` accuracies with location+op.

Value	GGNN-Rep	GGNN-RNN	HOPPITY
Top-1	63.8%	60.3%	**69.1%**
Top-3	67.6%	63.6%	**73.4%**

Table 4: `REP_VAL` accuracies with location+op.

	Top-1	Top-3
HOPPITY	**67.7%**	**73.3%**
SequenceR	64.2%	68.6%

Table 5: Overall OneDiff accuracy with location.

Bug Type	Amount	TAJS	HOPPITY
Undefined Property	7	0	1
Functional Bug	11	0	3
Refactoring	12	0	1
Total	30	0	5

Table 6: Comparison with TAJS.

Dos and Don'ts of Machine Learning in Computer Security, D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, K. Rieck, Usenix 2022.

Machine Learning already solved many problems in computer security

[Arp]

Unfortunately not… 🥺

[Arp]

Motivation—Historical Examples

Network intrusion detection: The base rate fallacy

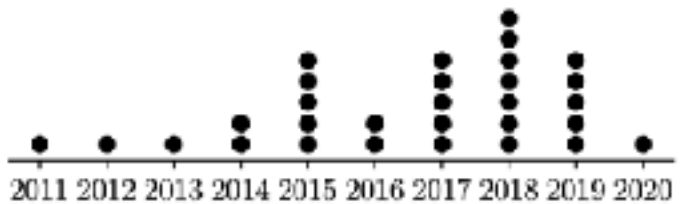
- Intrusion detectors should have low false positive rates (FPR)
- 'Low' FPR often still corresponds to large number of false positives

Android malware detection: Spatio-temporal bias inflating performance

- Models trained with access to 'future' information
- Unrealistic class balance inflates performance

[Arp]

Overview



1. **Identification of common pitfalls**
   - 10 subtle issues affecting ML for security
   - Recommendations for avoiding them
2. **Survey on the prevalence of pitfalls**
   - Review of 30 top papers in security
   - Pitfalls are widespread
3. **Case studies demonstrating impact of pitfalls**
   - Mobile malware detection
   - Vulnerability discovery
   - Source code authorship attribution
   - Network intrusion detection

[Arp]

ML Pipeline and Pitfalls

Data Collection and Labeling

- **P1** Sampling bias
- **P2** Label Inaccuracy

System Design and Learning

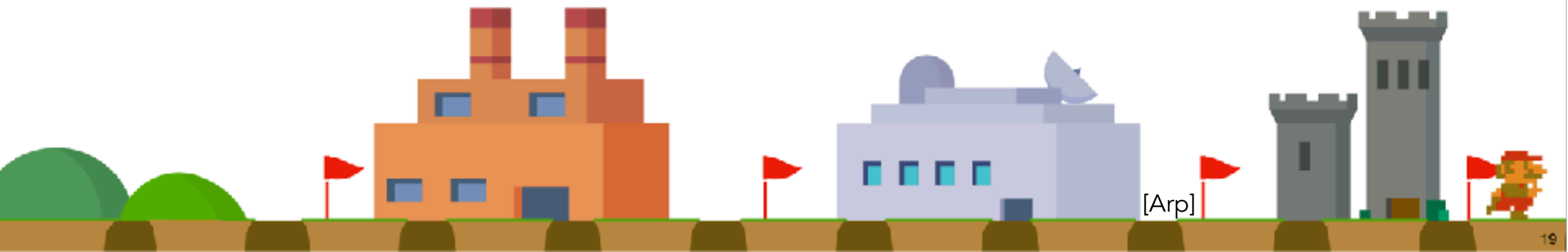
- **P3** Data snooping
- **P4** Spurious correlations
- **P5** Biased parameters

Performance Evaluation

- **P6** Inappropriate baselines
- **P7** Inappropriate measures
- **P8** Base rate fallacy

Deployment and Operation

- **P9** Lab-only evaluation
- **P10** Inappropriate threat model



[Arp]

P1 – Sampling Bias. The collected data does not sufficiently represent the true data distribution of the underlying security problem [1, 30, 33].

P2 – Label Inaccuracy. The ground-truth labels required for classification tasks are inaccurate, unstable, or erroneous, affecting the overall performance of a learning-based system [85, 144].

P3 – Data Snooping. A learning model is trained with data that is typically not available in practice. Data snooping can occur in many ways, some of which are very subtle and hard to identify [1].

P4 – Spurious Correlations. Artifacts unrelated to the security problem create shortcut patterns for separating classes. Consequently, the learning model adapts to these artifacts instead of solving the actual task.

P5 – Biased Parameter Selection. The final parameters of a learning-based method are not entirely fixed at training time. Instead, they indirectly depend on the test set.

P6 – Inappropriate Baseline. The evaluation is conducted without, or with limited, baseline methods. As a result, it is impossible to demonstrate improvements against the state of the art and other security mechanisms.

P7 – Inappropriate Performance Measures. The chosen performance measures do not account for the constraints of the application scenario, such as imbalanced data or the need to keep a low false-positive rate.

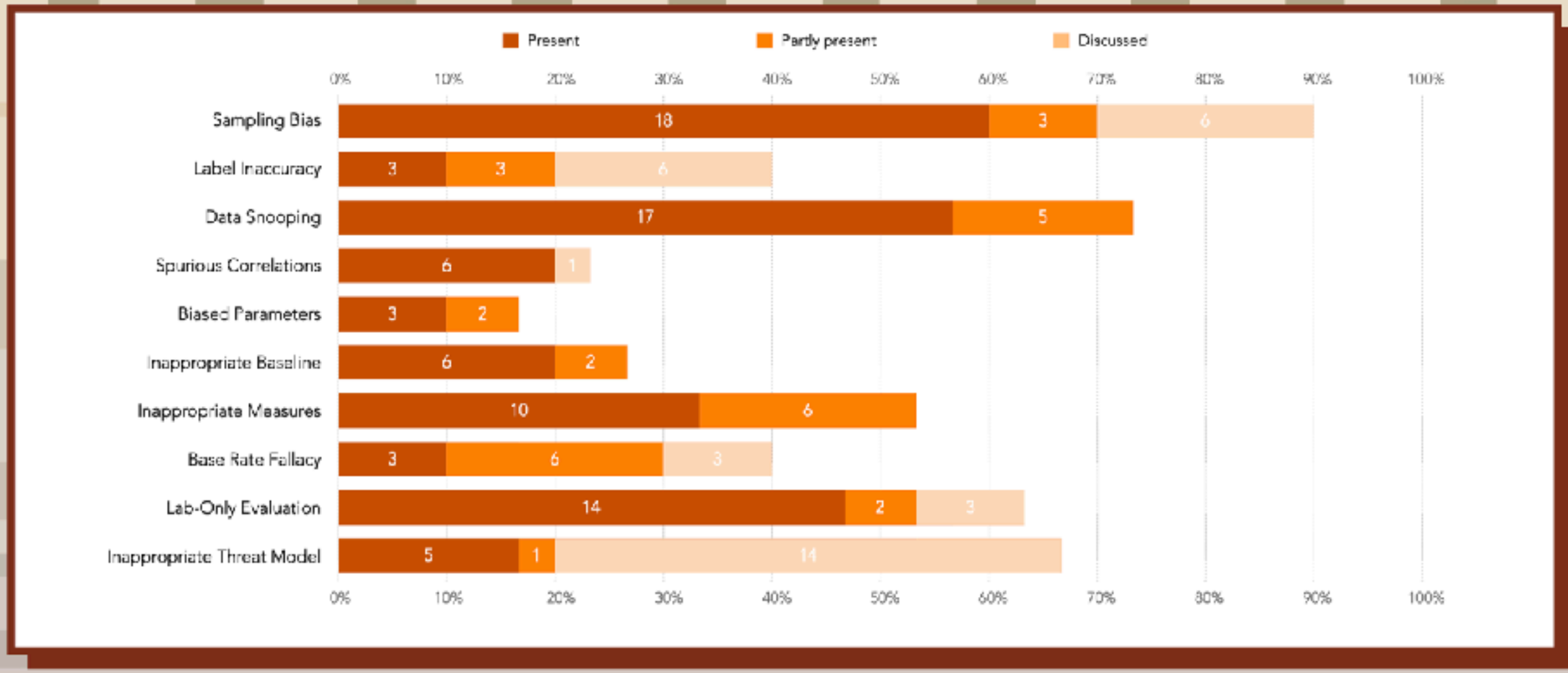
P8 – Base Rate Fallacy. A large class imbalance is ignored when interpreting the performance measures leading to an overestimation of performance.

P9 – Lab-Only Evaluation. A learning-based system is solely evaluated in a laboratory setting, without discussing its practical limitations.

P10 – Inappropriate Threat Model. The security of machine learning is not considered, exposing the system to a variety of attacks, such as poisoning and evasion attacks.

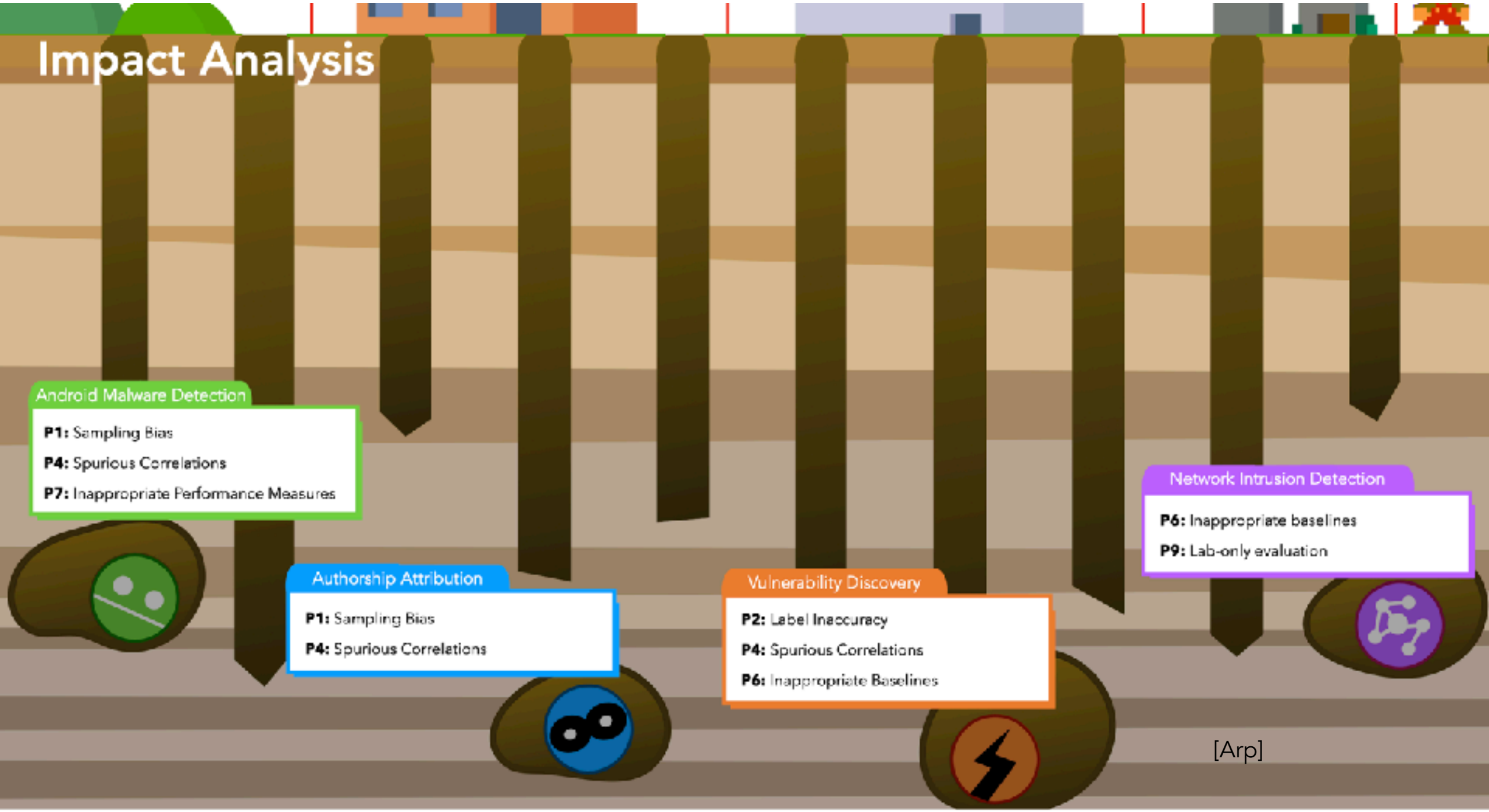
[Arp]

Prevalence Study



[Arp]

Impact Analysis



[Arp]

Acknowledgments

- [Hoppity] HOPPITY: Learning Graph Transformations to Detect and Fix Bugs in Programs, E. Dinella, H. Dai, Z. Li, M. Naik, L. Song, K. Wang, ICLR 2020.
- [Arp] Dos and Don'ts of Machine Learning in Computer Security, D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, K. Rieck, Usenix 2022.