ReMoS: Reducing Defect Inheritance in Transfer Learning via Relevant Model Slicing

Ziqi Zhang, Yuanchun Li, Jindong Wang, Bingyan Liu,
Ding Li, Yao Guo, Xiangqun Chen, Yunxin Liu

Software Reuse Is a Common Practice

- Copy-pasting a piece of code

Original Code

```
void more_variables(){
    int idx, old_count, old_var[];

    /* Save the old values. */
    old_count =
    old_var = va

    /* Incremen
    v_count +=
    variables =

    /* Copy the
    for (idx=3;
        variabl

    /* Initiali
    for (; idx
        variabl
}
```

Pasted Code

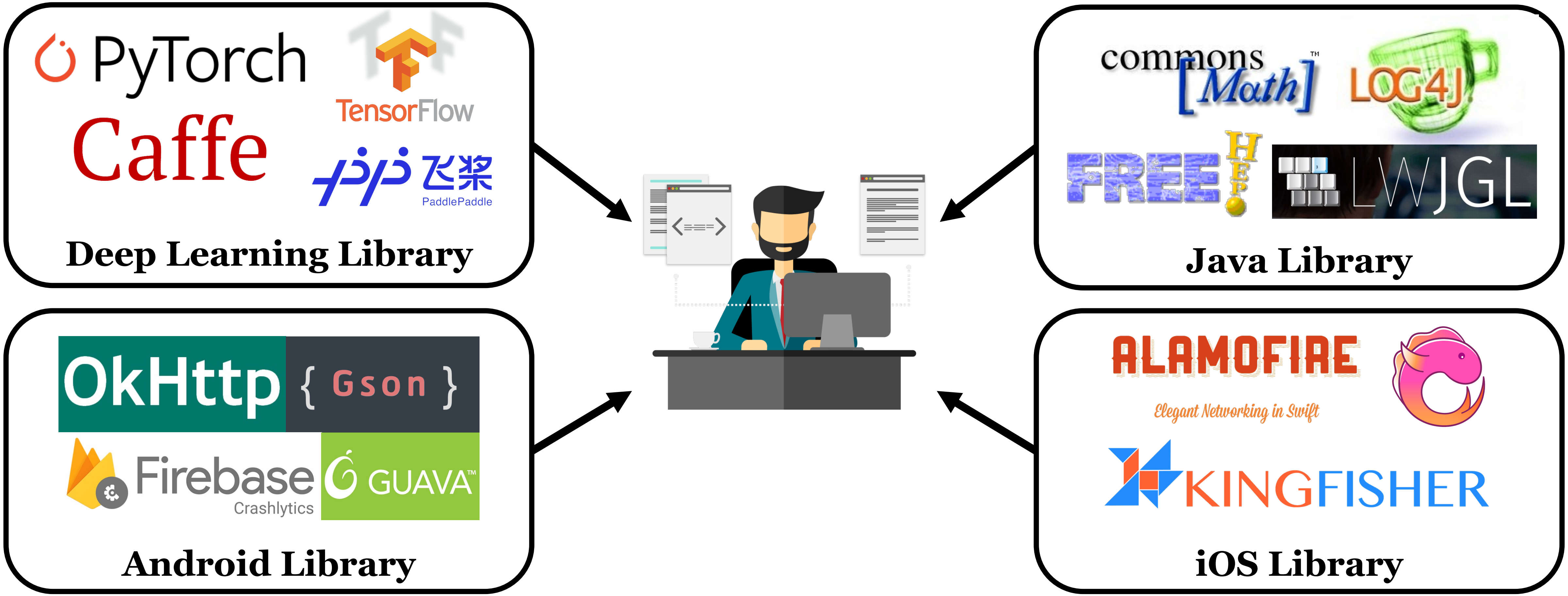
```
void more_functions(){
    int idx, old_count, old_f[];

    /* Save the old values. */
    old_count =
    old_f = fun

    /* Incremen
    f_count +=
    functions =

    /* Copy the
    for (idx=3;
        functio

    /* Initiali
    for (; idx
        functio
}
```

```
void more_arrays(){
    int idx, old_count, old_array[];

    /* Save the old values. */
    old_count = a_count;
    old_ary = arrays;

    /* Increment by a fixed amount. */
    a_count += STORE_INCR;
    arrays = new int[100];

    /* Copy the old variables. */
    for (idx=3; idx<old_count; idx++)
        arrays[idx] = old_ary[idx];

    /* Initialize the new element. */
    for (; idx < a_count; idx++)
        arrays[idx] = 0;
}
```

- Jablonski et al "Aiding software maintenance with copy-and-paste clone-awareness." ICPC 2010

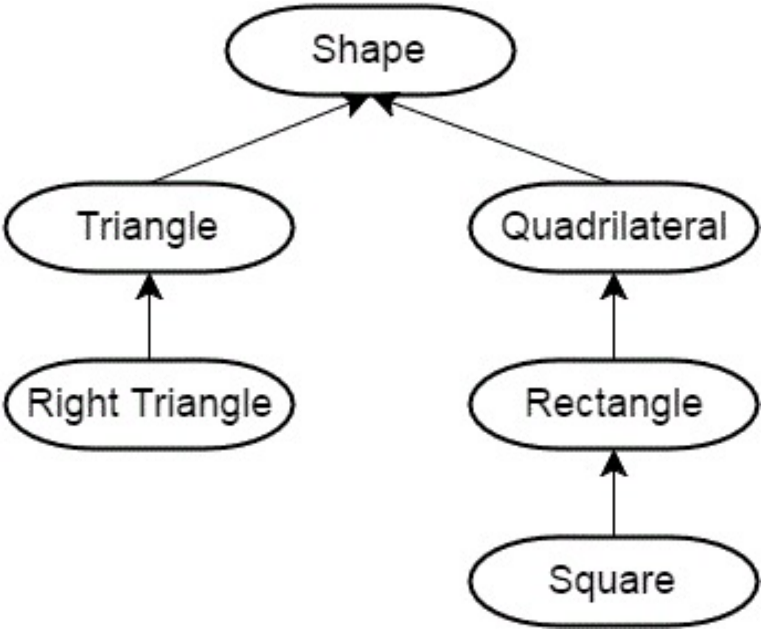
Software Reuse Is a Common Practice

- Third-party library



Software Reuse Is a Common Practice

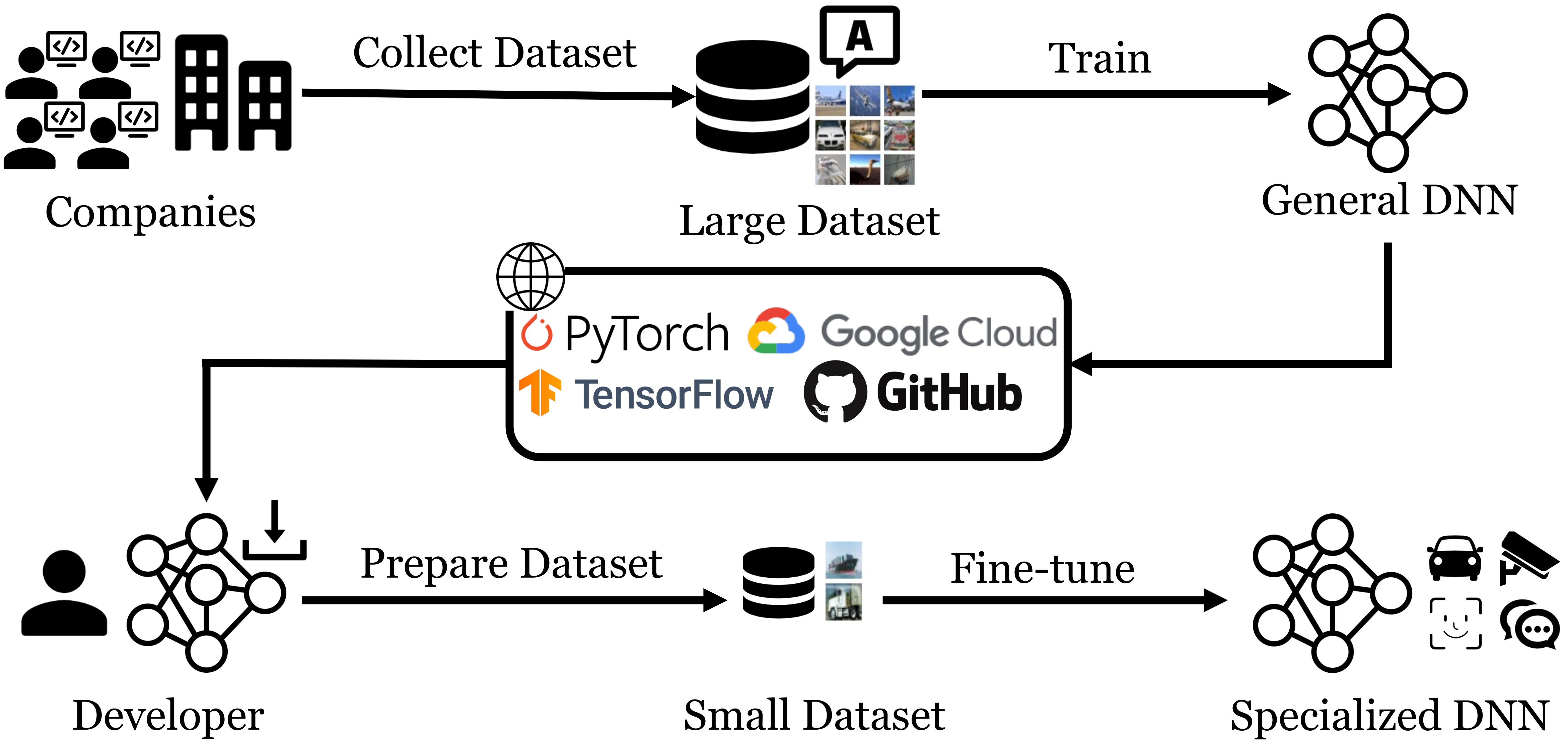
- Class Inheritance




```python
class Shape(object):
    # Constructor
    def __init__(self, size):
        self.size = size

    # To get size
    def getSize(self):
        return self.size

    def getArea(self):
        ...

    def getPerimeter(self):
        ...
```

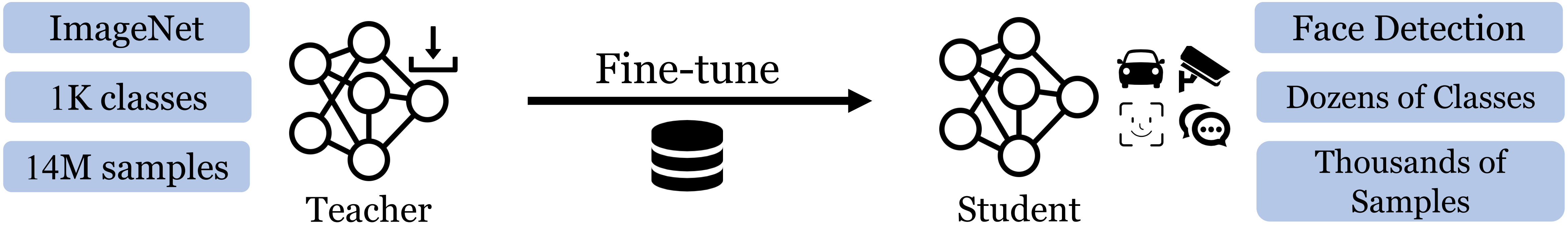
```python
class Triangle(Shape):
    def getArea(self):
        # Heron's Formula
        p = self.size[0]+self.size[1]\
            +self.size[2]
        return sqrt(p*(p-self.size[0])\
            *(p-self.size[1])\
            *(p-self.size[2]))

    def getPerimeter(self):
        return self.size[0]\
            +self.size[1]+self.size[2]

class Quareilateral(Shape):
    def getPerimeter(self):
        return self.size[0]+self.size[1]\
            +self.size[2]+self.size[3]

class Rectangle(Quareilateral):
    def getArea(self):
        return self.size[0]*self.size[1]

    def getPerimeter(self):
        return 2*(self.size[0]+self.size[1])
```

DNN Model Reuse: Transfer Learning



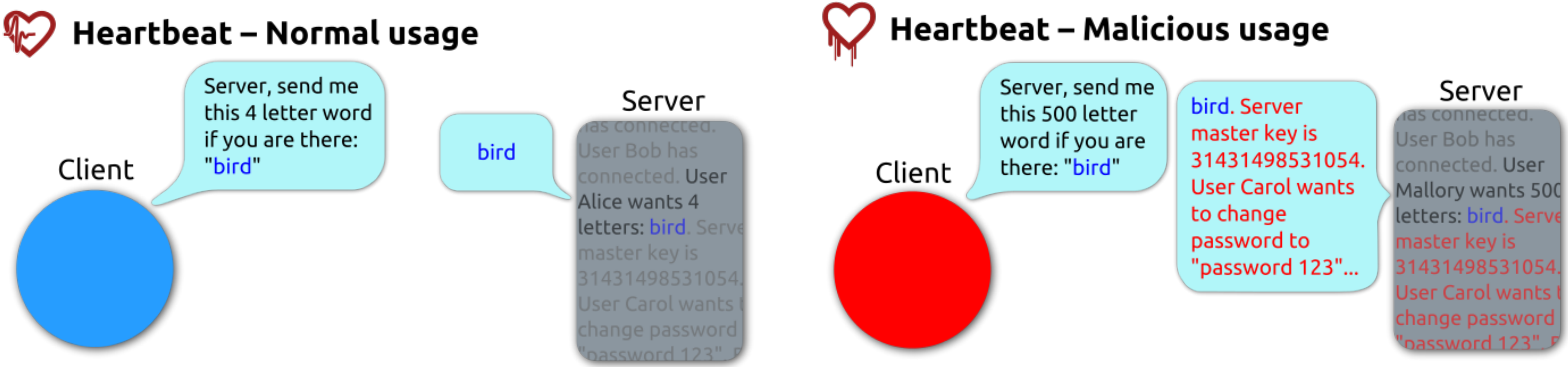
DNN Model Reuse: Transfer Learning

- (Pre-trained) Teacher Model
  - Trained by large-scale dataset, to complete complex task
  - Published on the Internet to be downloaded
- Student Model
  - Fine-tuned on small-scale private dataset, to complete simple task
- Advantage of transfer learning
  - High performance
  - Fast convergence and less training time
  - Less task-specific data

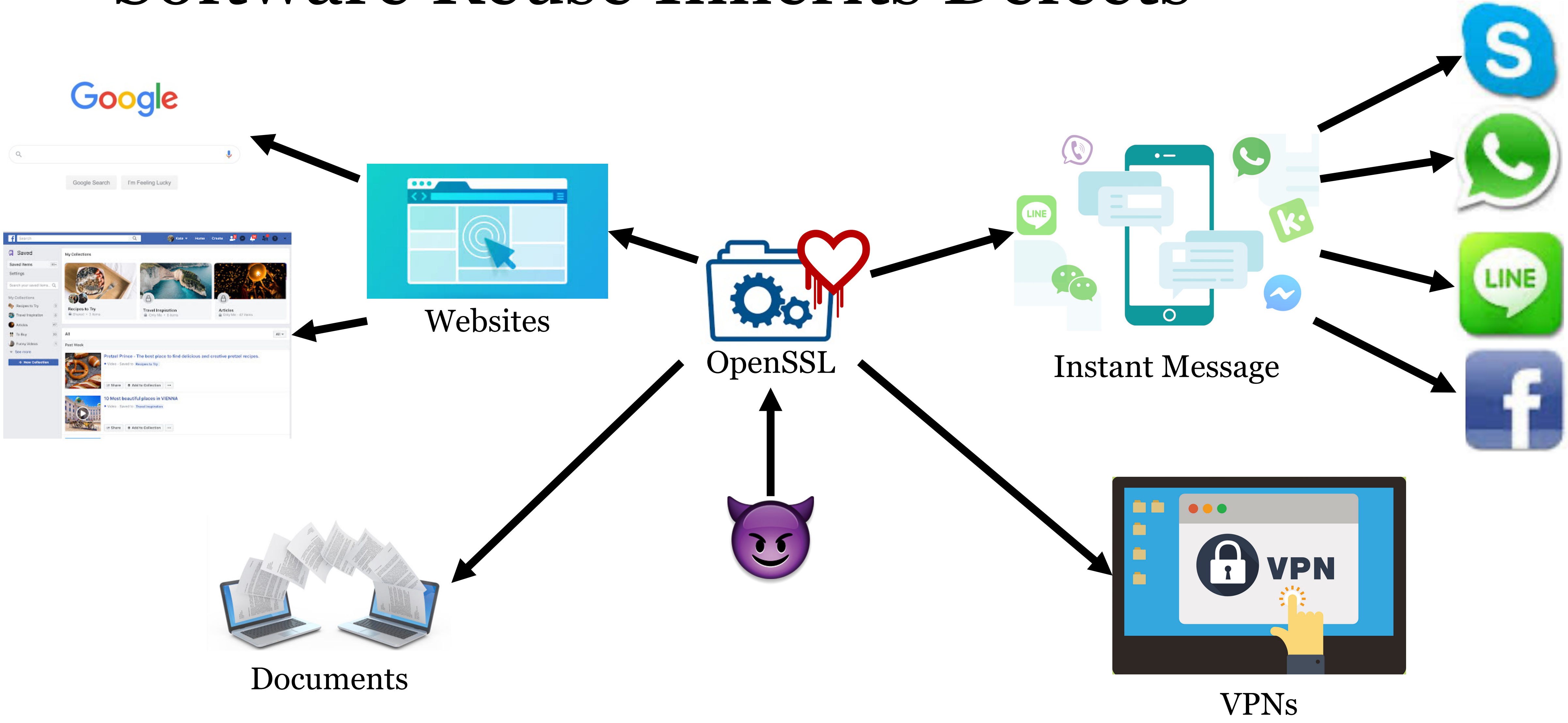


Software Reuse Inherits Defects

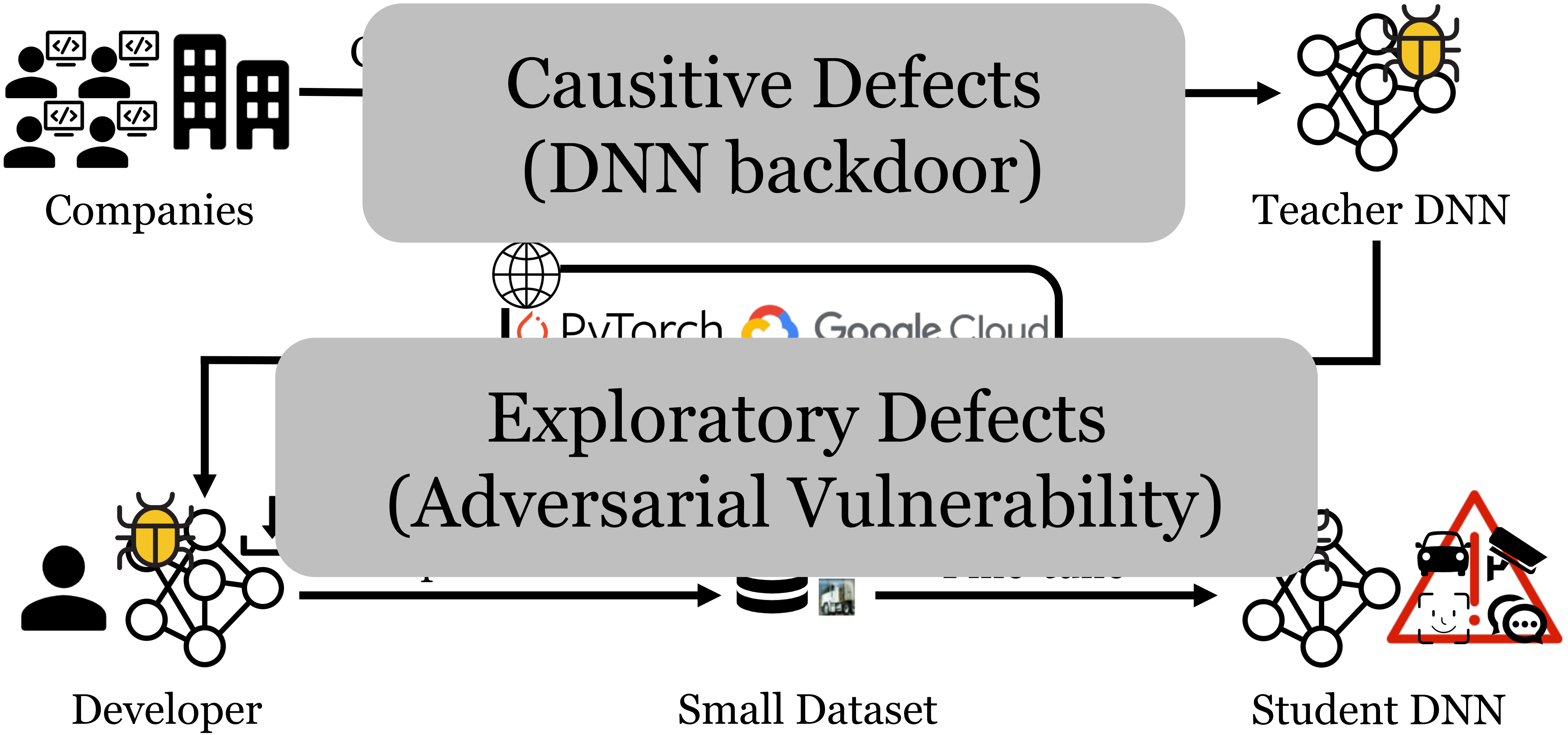
- The famous HeartBleed bug
  - A serious vulnerability in the popular OpenSSL cryptographic library.



Software Reuse Inherits Defects



Transfer Learning Inherits Defects



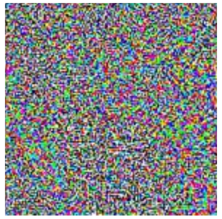
DNN defects

- DNN Defects are the deviation of the actual and expected results of a DNN model produced by certain input samples.

DNN defects

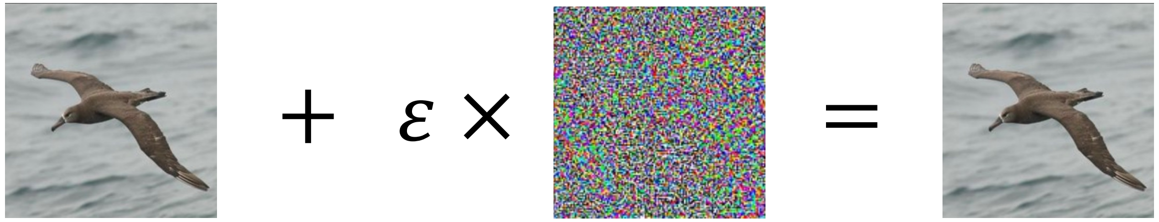
- DNN Defects are the deviation of the actual and expected results of a DNN model produced by certain input samples.

- Adversarial samples

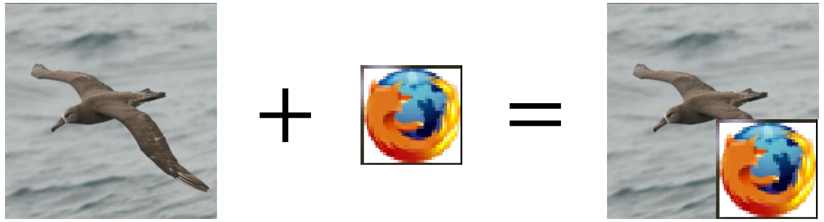
 $+ \ \varepsilon \times$  $=$ 

DNN defects

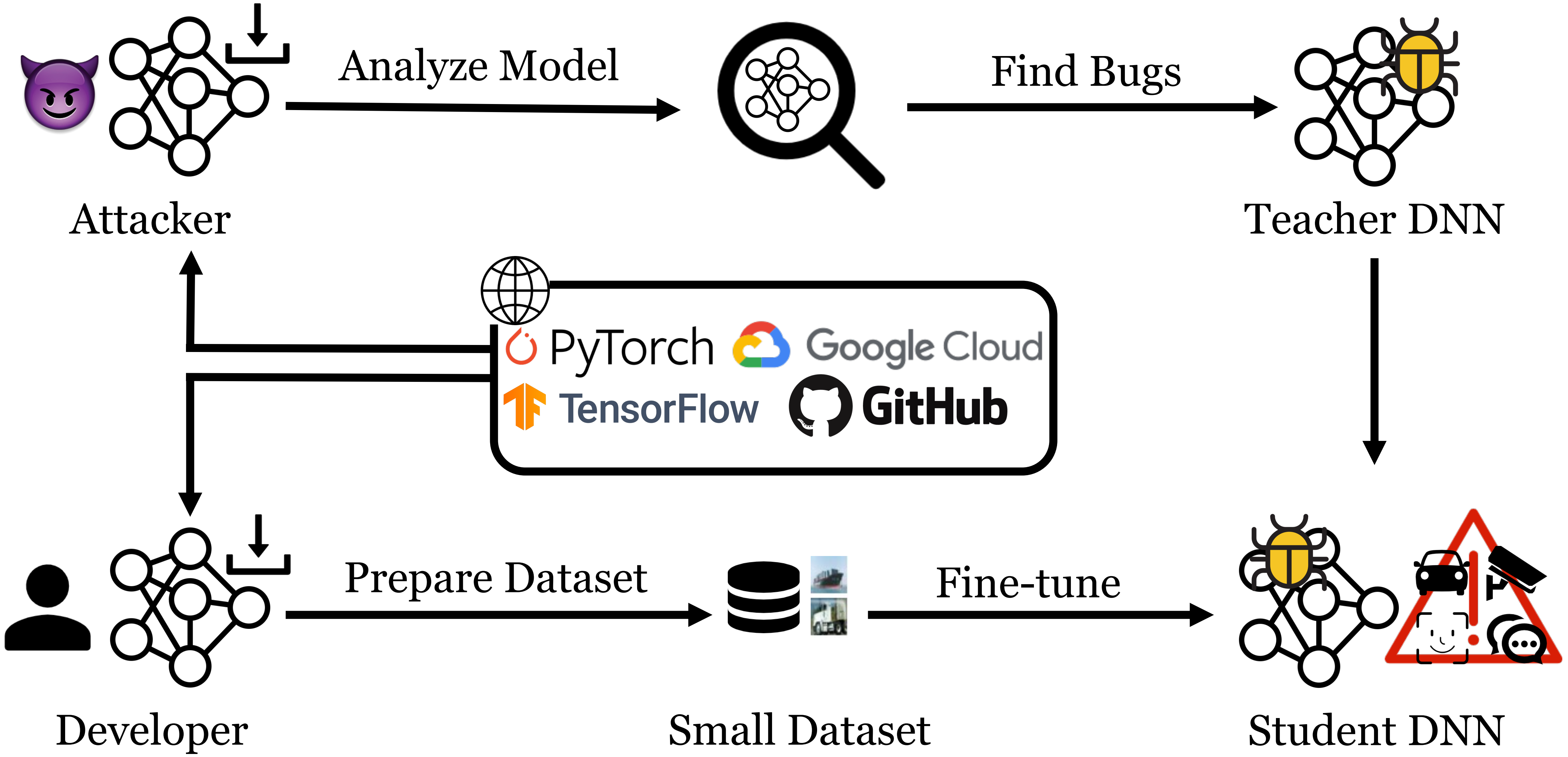
- DNN Defects are the deviation of the actual and expected results of a DNN model produced by certain input samples.
- Adversarial samples



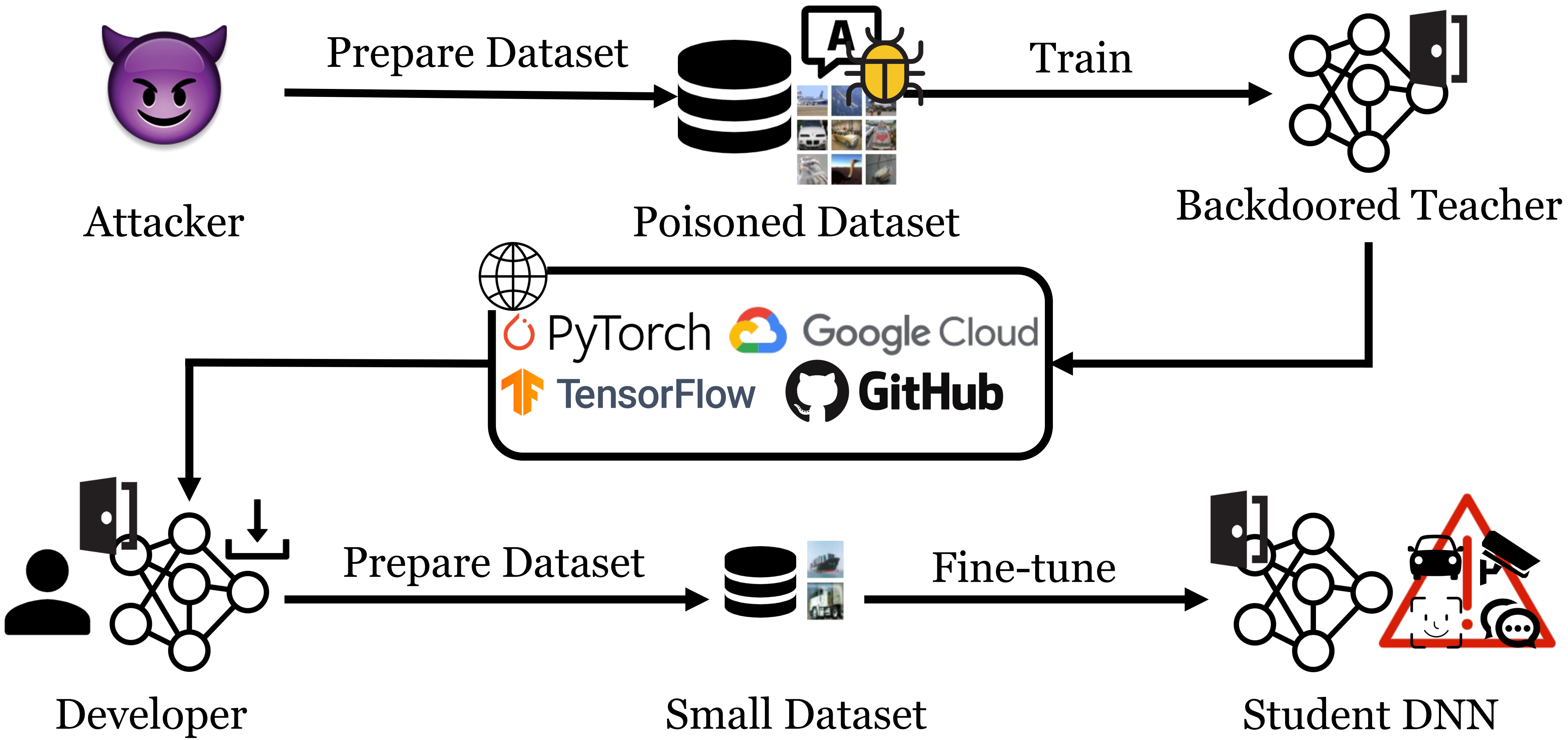
- Backdoor



TL Defect: Exploratory Defect

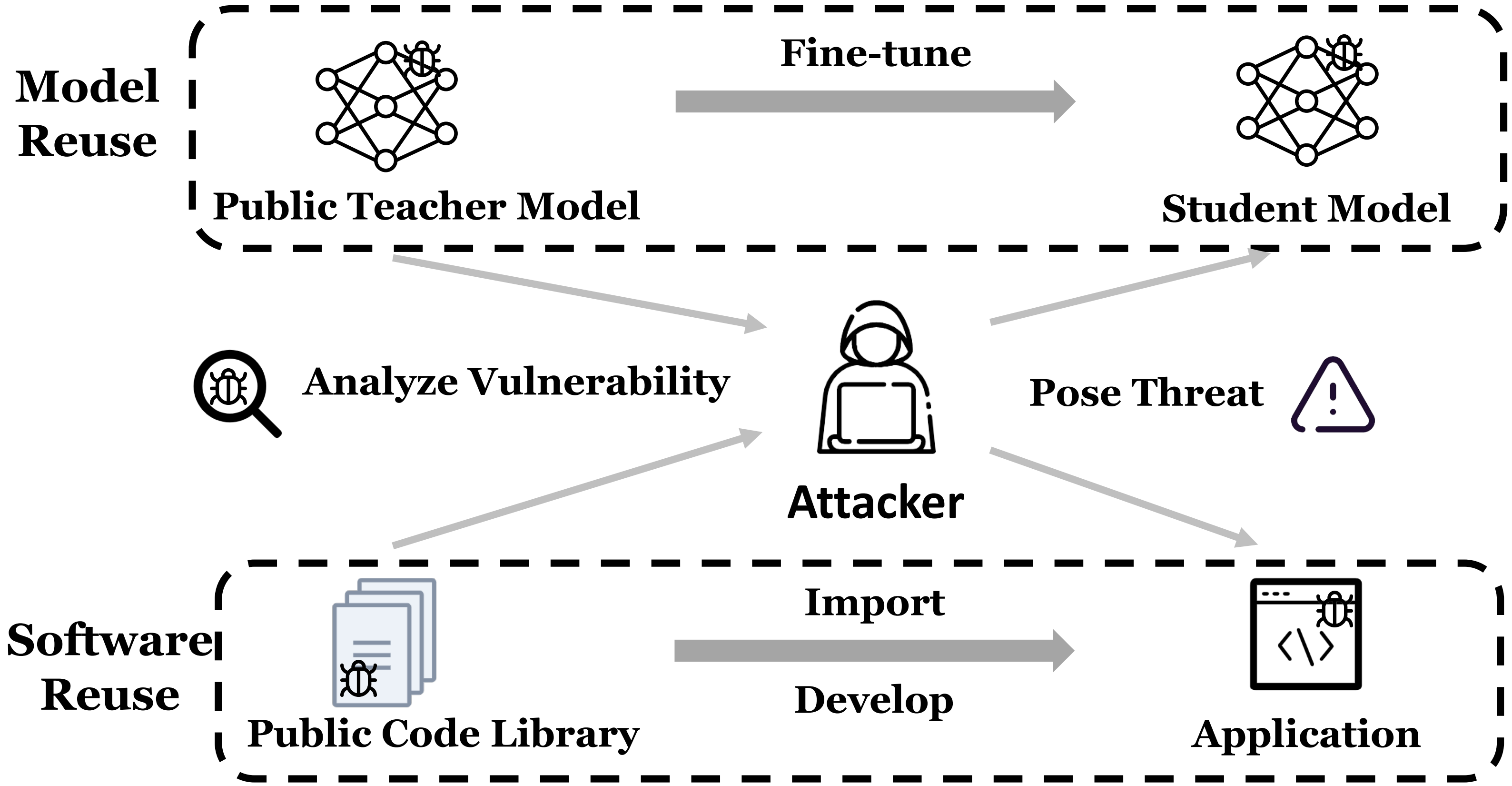


TL Defect: Causative Defect



Transfer Learning Inherits Defects

- Model reuse VS. software reuse



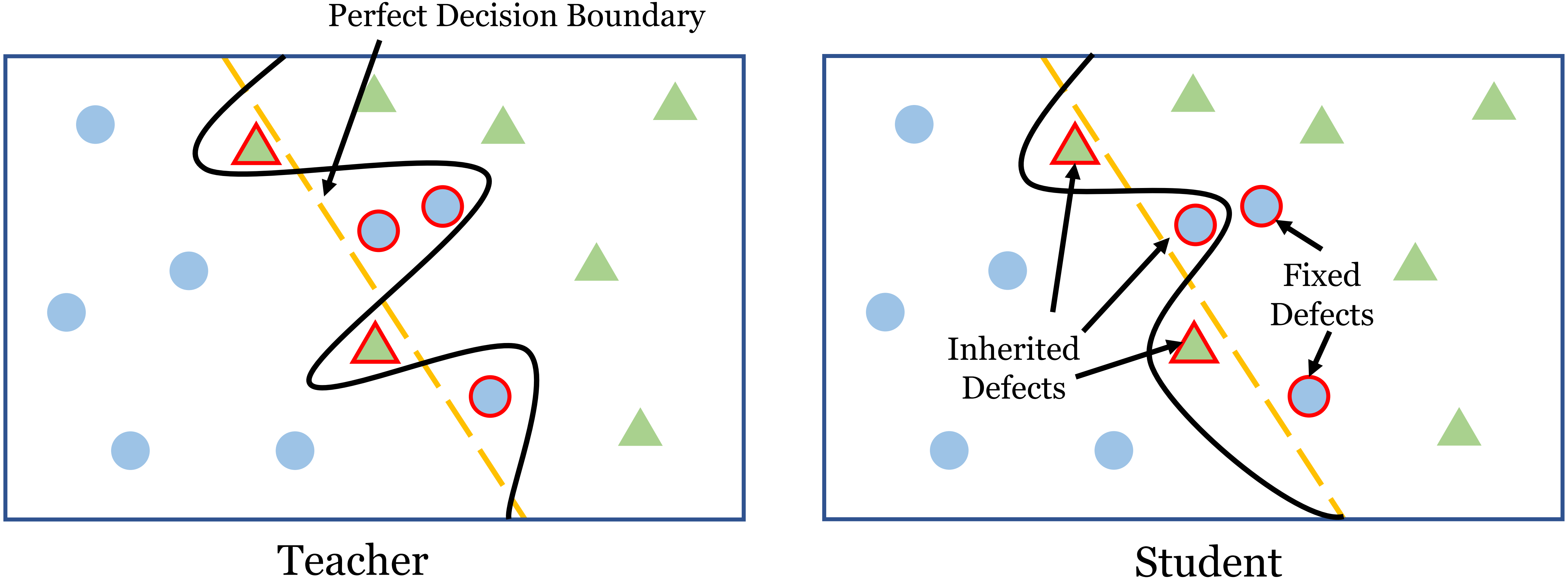
Transfer Learning Inherits Defects

- Potential defects in the prior literature that may inherit during model reuse

Task	Defect Type		Inheritance Rate
CV	Adversarial Vulnerability	Penultimate-Layer Guided [58]	58.01%
		Neuron-Coverage Guided [21, 55]	52.58%
	Backdoor	Latent Data Poison [70]	72.91%
NLP	Adversarial Vulnerability	Greedy Word Swap [31]	64.86%
		Word Importance Ranking [29]	94.73%
	Backdoor	Data Poison [20]	96.72%
		Weight Poison [32]	97.85%

Cause of Defect Inheritance

- The student model has similar decision boundary as the teacher model.



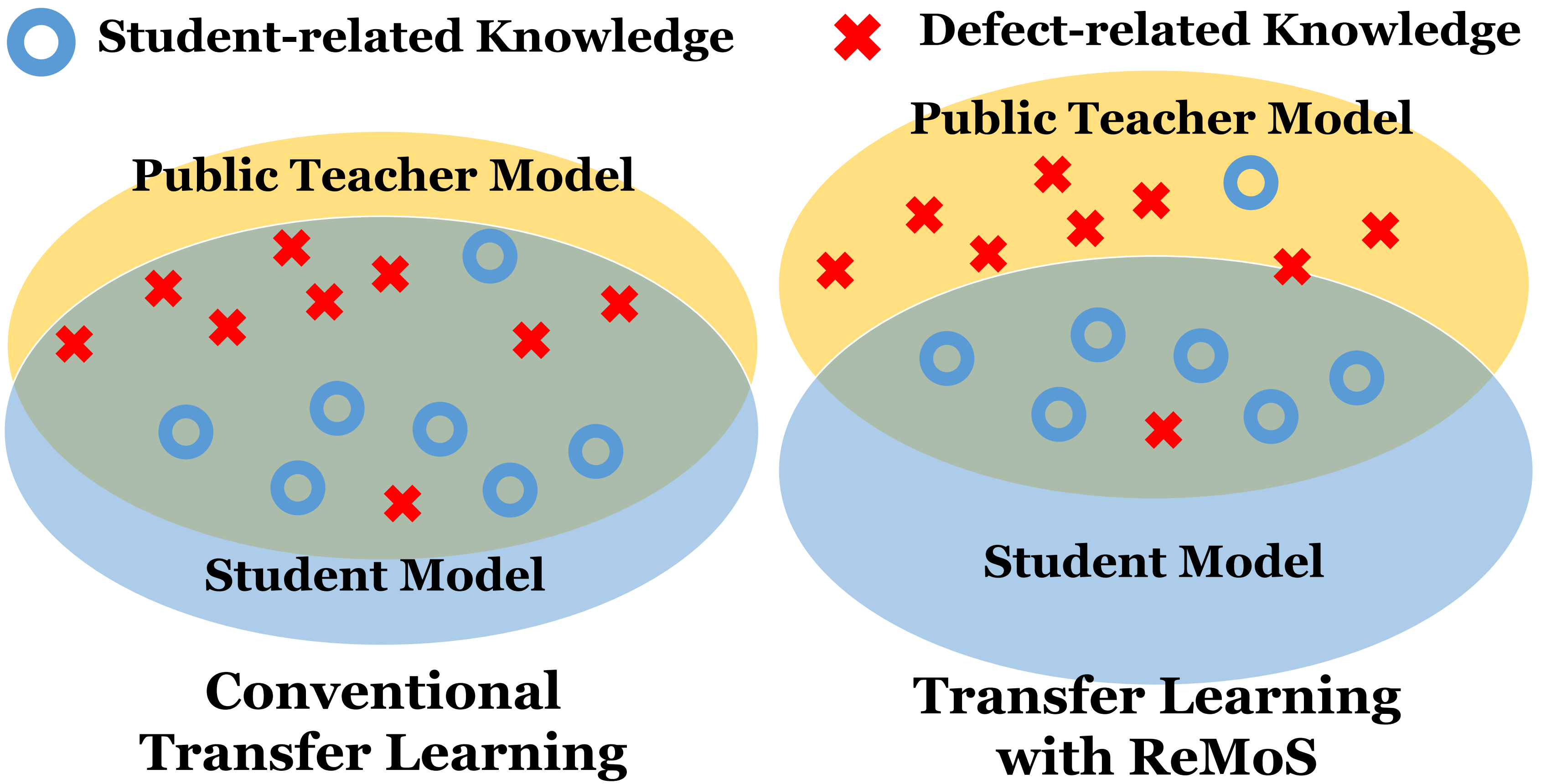
Defender’s Goal

Effectiveness

Accuracy

Efficiency

Utility



ReMoS: Relevant Model Slicing

- Relevant Slicing for Traditional Programs
  - Given a program *P* and a slicing criterion (a test case t and a target statement *s*), relevant slicing is to compute a subset of program statements that influence or **have the potential to** influence the statement *s* during the execution of *t*.

ReMoS: Relevant Model Slicing

- Relevant Slicing for Traditional Programs
  - Given a program P and a slicing criterion (a test case t and a target stateme … ram stateme … nce the stateme

```
1 read( a, b )
2 int x=0, y=0
3 x = a + 1
4 y = b + 1
5 int w = 0
6 if x > 3 then
7     if y > -3 then
8         w = w / b
```

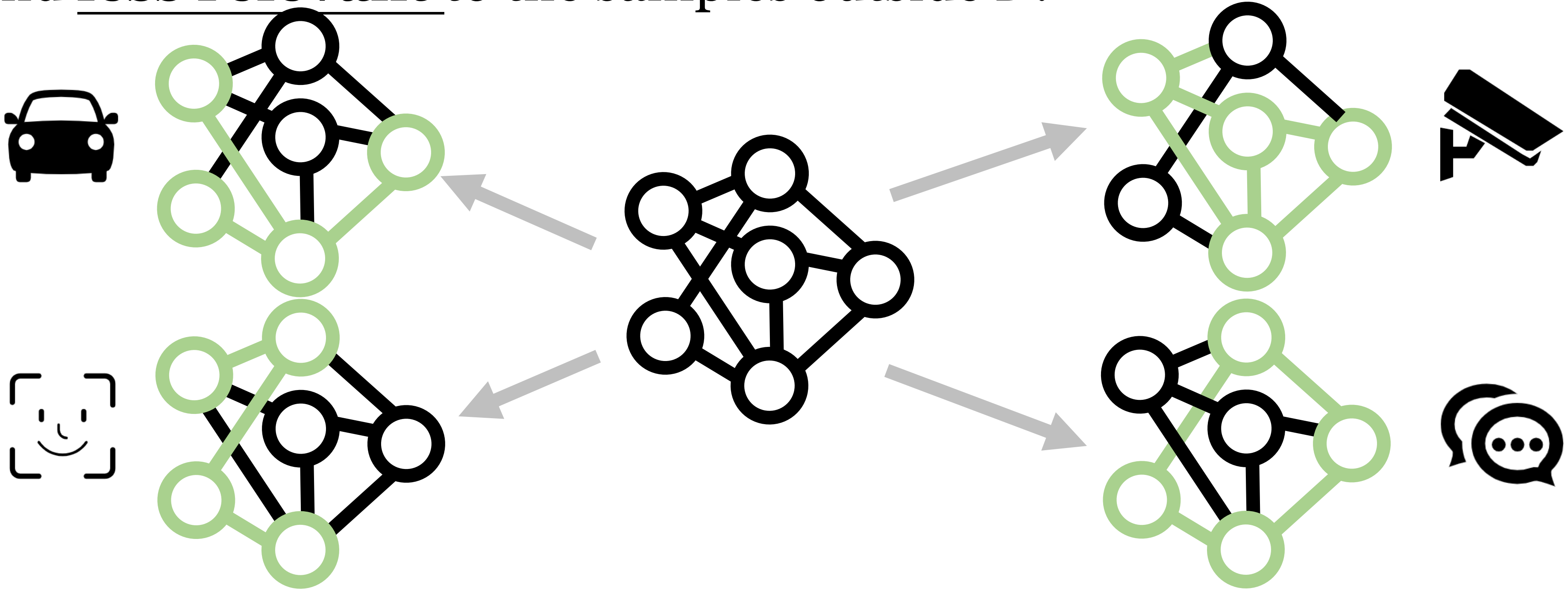
Input domain of the downstream application
$a < 1$

```
1 read( a, b )
2 int x=0, y=0
3 x = a + 1
4
5 int w = 0
6 if x > 3 then
7
8
9
10 endif
11 if y > 5 then
12     w = w + 1
13 endif
14 write( w )
```

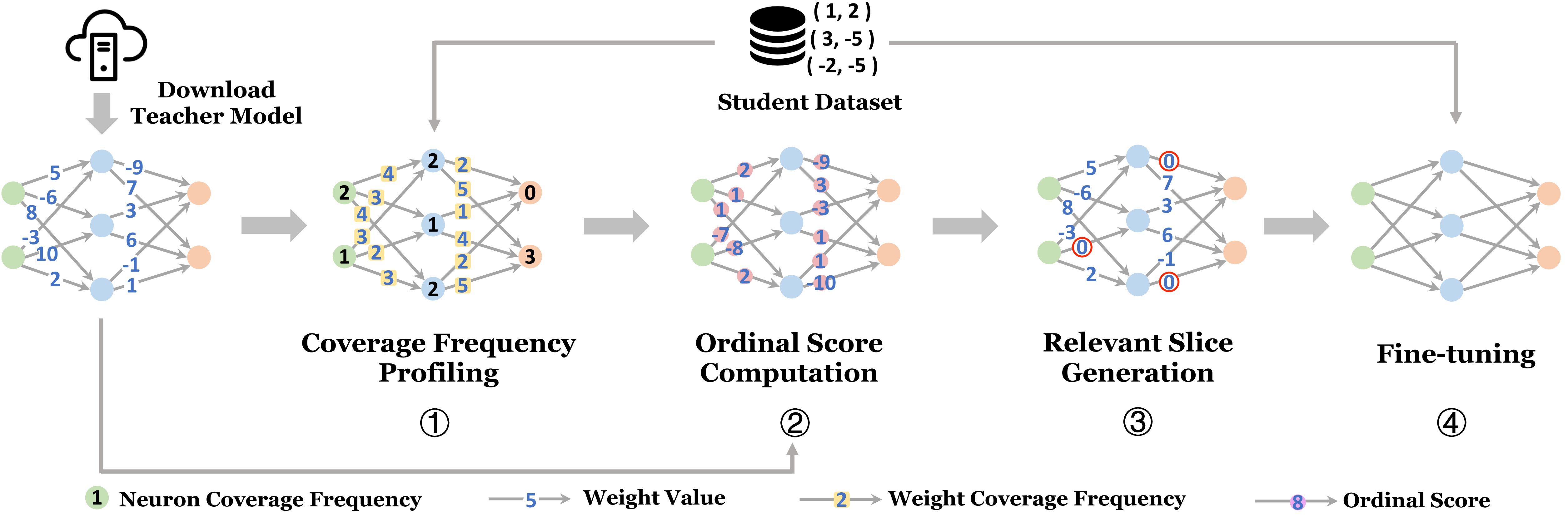
Criterion $< \{a = 0, b = 4\}, 14, w >$

ReMoS: Relevant Model Slicing

- Relevant Slicing for DNN Models
  - Given a DNN model M and a target domain dataset D, relevant model slicing is to compute a subset of model weights that are **more relevant** (bounded by a threshold) to the inference of samples in D and **less relevant** to the samples outside D.



ReMoS: Relevant Model Slicing



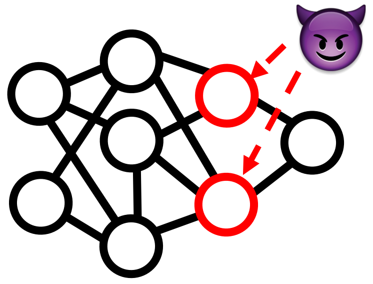
Evaluation

- Evaluation goals
  - Defect mitigation effectiveness
  - Generalizability
  - Efficiency
  - Interpretability

- Experiment setting
  - Four DNN models: two CV models and two NLP models
  - Seven DNN defects: adversarial vulnerability and backdoor
  - Eight datasets

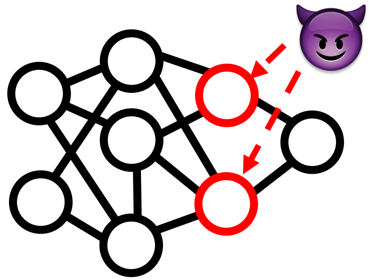
Evaluation

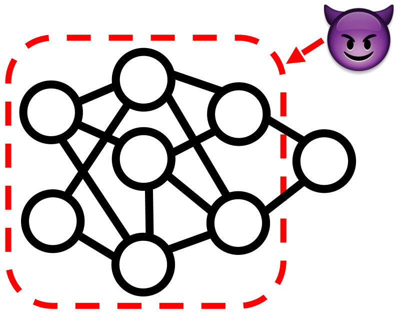
- Defect simulation
  - Penultimate-layer guided adversarial vulnerability
    - Tailored for transfer learning [58]
    - Targets the penultimate neurons



Evaluation

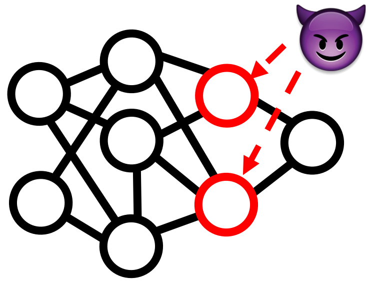
- Defect simulation
  - Penultimate-layer guided adversarial vulnerability
    - Tailored for transfer learning [58]
    - Targets the penultimate neurons
  - Neuron-coverage guided adversarial vulnerability
    - Adopted from DNN testing [33]
    - Targets all the internal neurons
    - Three coverages and three strategies

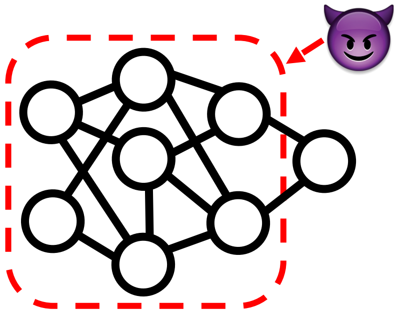


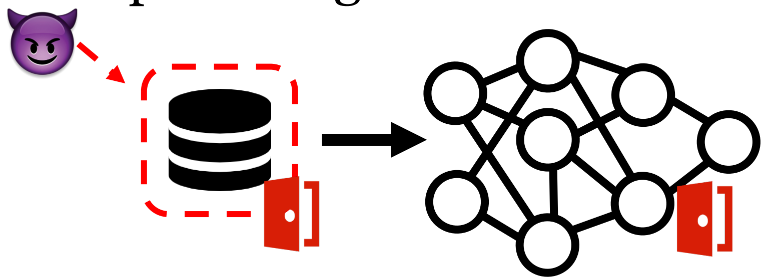


Evaluation

- Defect simulation
  - Penultimate-layer guided adversarial vulnerability
    - Tailored for transfer learning [58]
    - Targets the penultimate neurons
  - Neuron-coverage guided adversarial vulnerability
    - Adopted from DNN testing [33]
    - Targets all the internal neurons
    - Three coverages and three strategies
  - Backdoor
    - Data poisoning

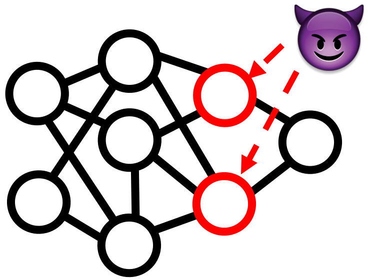


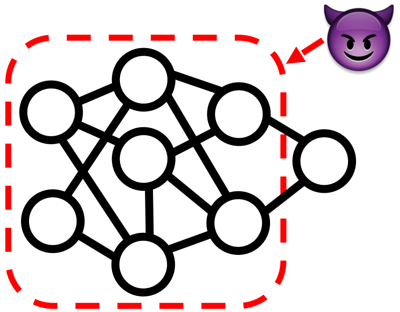


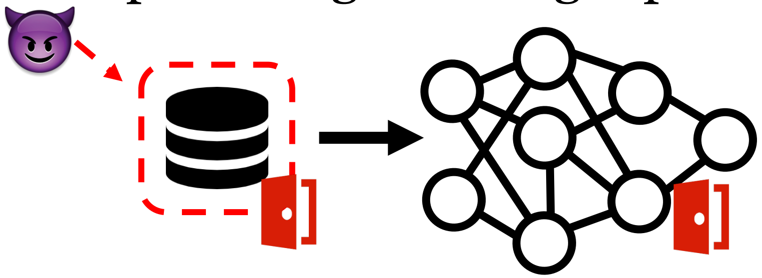


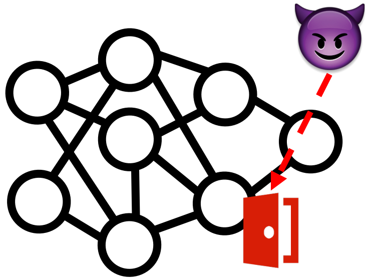
Evaluation

- Defect simulation
  - Penultimate-layer guided adversarial vulnerability
    - Tailored for transfer learning [58]
    - Targets the penultimate neurons
  - Neuron-coverage guided adversarial vulnerability
    - Adopted from DNN testing [33]
    - Targets all the internal neurons
    - Three coverages and three strategies
  - Backdoor
    - Data poisoning and weight poisoning









Evaluation

- Defect mitigation for Neuron-coverage guided adversarial vulnerability
  - ReMoS eliminates averagely 63% of the inherited defects

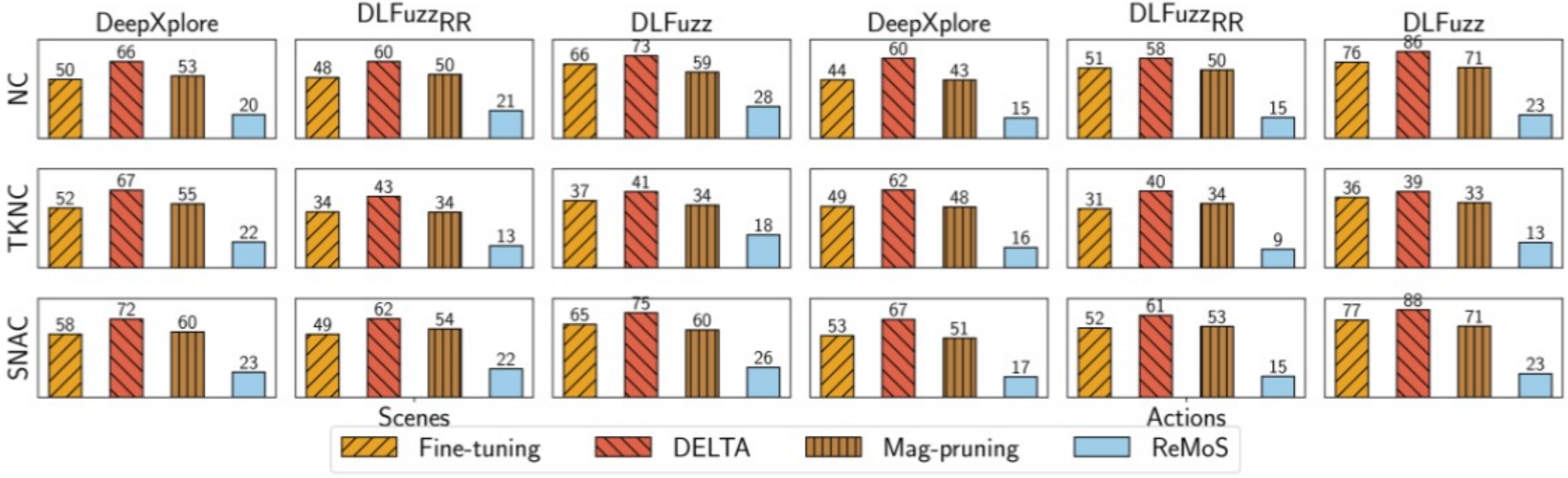


Figure 6: The inheritance rate of adversarial inputs generated by different neuron-coverage-guided test generators on ResNet18.

Evaluation

- Defect mitigation for NLP backdoor
  - ReMoS eliminate 50% data poisoning and 61% weight poisoning defects

Table 2: The defect reduction effectiveness of ReMoS against two backdoor attacks on NLP tasks. For each model, we include four situations where the attacker's dataset may be the same or different as the student dataset.

Model		Dataset	Data Poisoning						Weight Poisoning					
			Fine-tune		Mag-prune		ReMoS		Fine-tune		Mag-prune		ReMoS	
			ACC	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*	*ACC*	*DIR*
BERT	FDK	SST-2 to SST-2	92.70	100.00	92.35	100.00	91.27	39.09	92.29	100.00	92.44	100.00	90.92	29.82
		IMDB to IMDB	87.96	96.11	88.24	96.15	85.53	61.73	89.34	96.15	89.48	96.09	87.00	37.72
	DS	SST-2 to IMDB	90.53	100.00	91.26	100.00	90.04	74.67	91.67	100.00	91.16	100.00	87.42	61.48
		IMDB to SST-2	93.21	96.17	92.46	96.17	91.15	27.71	92.80	96.22	92.58	96.02	91.94	21.55
RoBERTa	FDK	SST-2 to SST-2	94.19	100.00	93.70	100.00	91.17	29.82	93.37	100.00	93.19	98.93	90.70	24.94
		IMDB to IMDB	90.60	93.52	89.54	95.24	85.74	70.19	89.05	96.53	88.76	92.05	86.34	85.91
	DS	SST-2 to IMDB	92.11	99.88	92.27	100.00	90.32	24.14	91.85	100.00	90.82	99.53	88.71	30.83
		IMDB to SST-2	93.52	88.15	92.65	85.26	92.17	61.26	93.85	93.93	93.57	91.21	89.95	18.07
Average Relative Value			-	-	0.99	0.99	0.97	0.50	-	-	0.99	0.98	0.97	0.39

Conclusion

- DNN model reuse (transfer learning), like traditional software reuse, faces defect inheritance problem
- Two possible types of inheritable defects are adversarial vulnerability and DNN backdoor
- The defect inheritance problem can be mitigated by only reuse the relevant model slice instead of the whole DNN model
- The proposed approach, ReMoS (Relevant Model Slicing), can mitigate over 60% of the CV inherited defects and 40% of the NLP inherited defects