FDL 2020

Robust Models for Source Code: Techniques and Applications

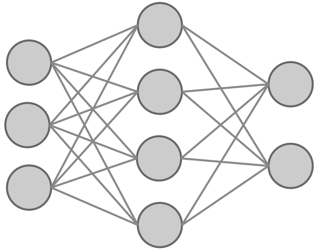
Pavol Bielik
pavol.bielik@inf.ethz.ch

Department of Computer Science
ETH zürich

SRILAB

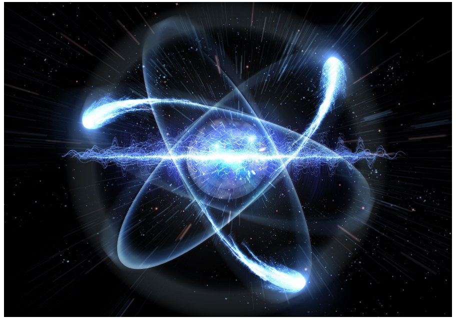
Secure, Reliable, and Intelligent Systems Lab @ ETH

Robust ML



Neural network verification

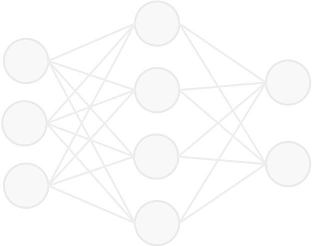
ML for programming



Probabilistic programming

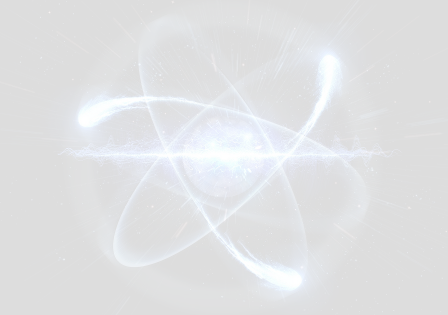
Secure, Reliable, and Intelligent Systems Lab @ ETH

Robust ML



Neural network verification

ML for programming



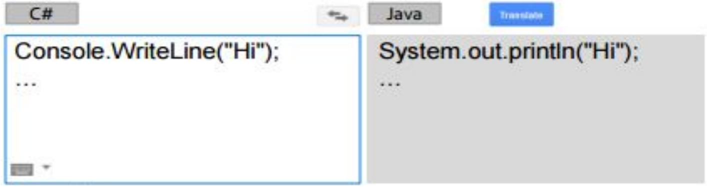
Probabilistic programming

Statistical Programming Tools

Code Completion

```
Camera camera = Camera.open();
camera.SetDisplayOrientation(90);
                  ?
```

Port Code

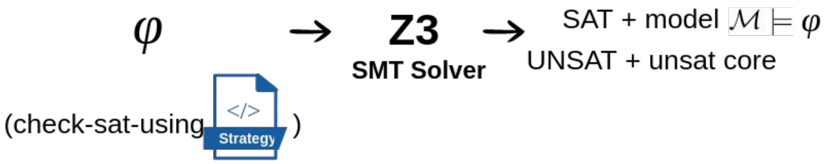


```
Console.WriteLine("Hi");
...
```

```
System.out.println("Hi");
...
```

Program Synthesis

Learning to Solve Formulas



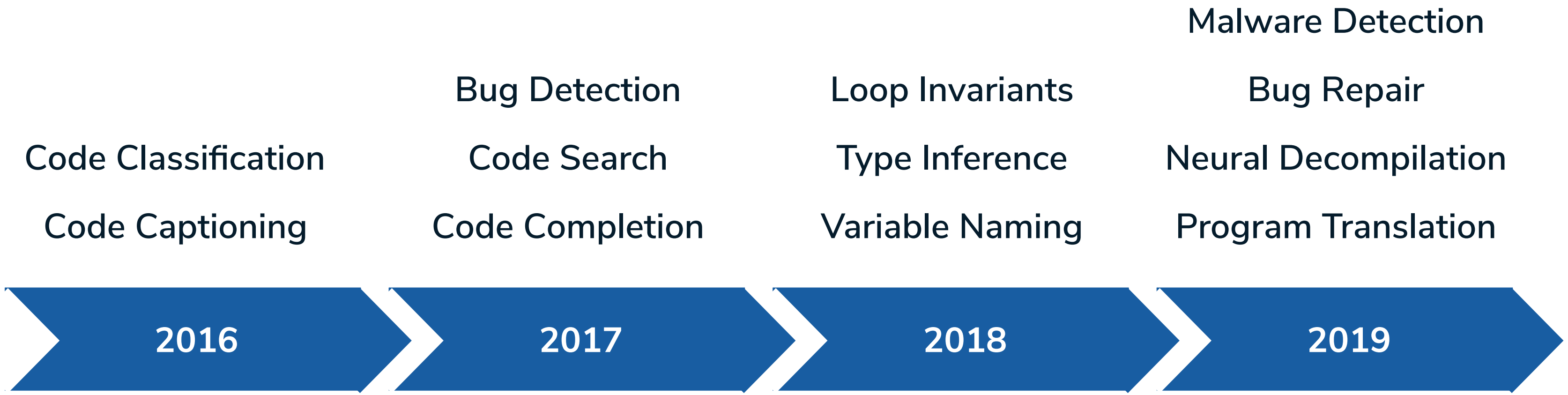
Up to 100x speed-up over Z3

Bug Detection

likely error

```
...
for x in range(a):
     print a[x]
```

Deep Learning + Code



2016

- Code Classification
- Code Captioning

2017

- Bug Detection
- Code Search
- Code Completion

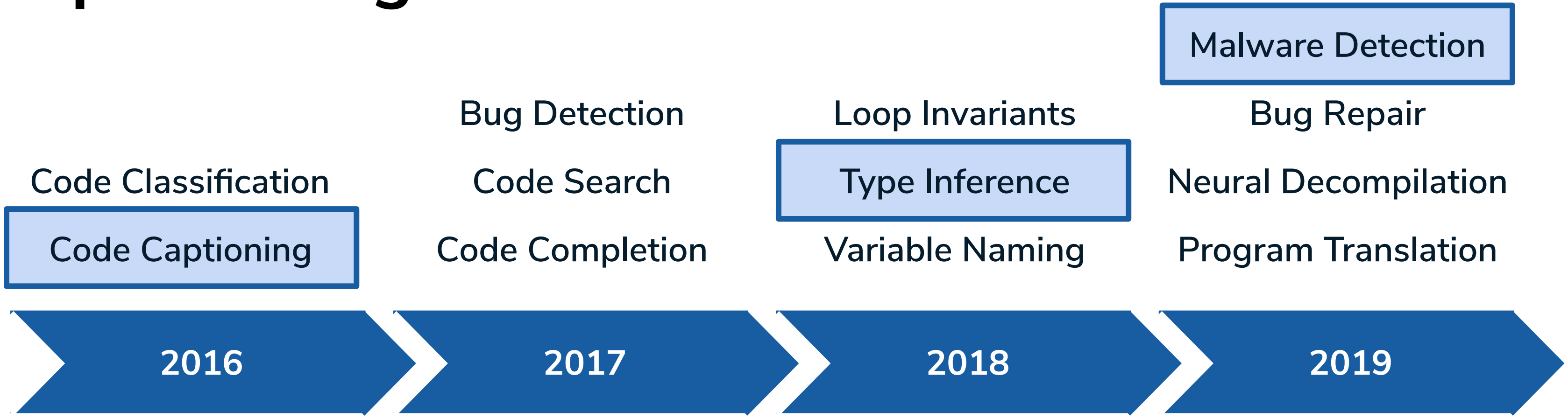
2018

- Loop Invariants
- Type Inference
- Variable Naming

2019

- Malware Detection
- Bug Repair
- Neural Decompilation
- Program Translation

Deep Learning + Code



Code Classification

Code Captioning

2016

Bug Detection

Code Search

Code Completion

2017

Loop Invariants

Type Inference

Variable Naming

2018

Malware Detection

Bug Repair

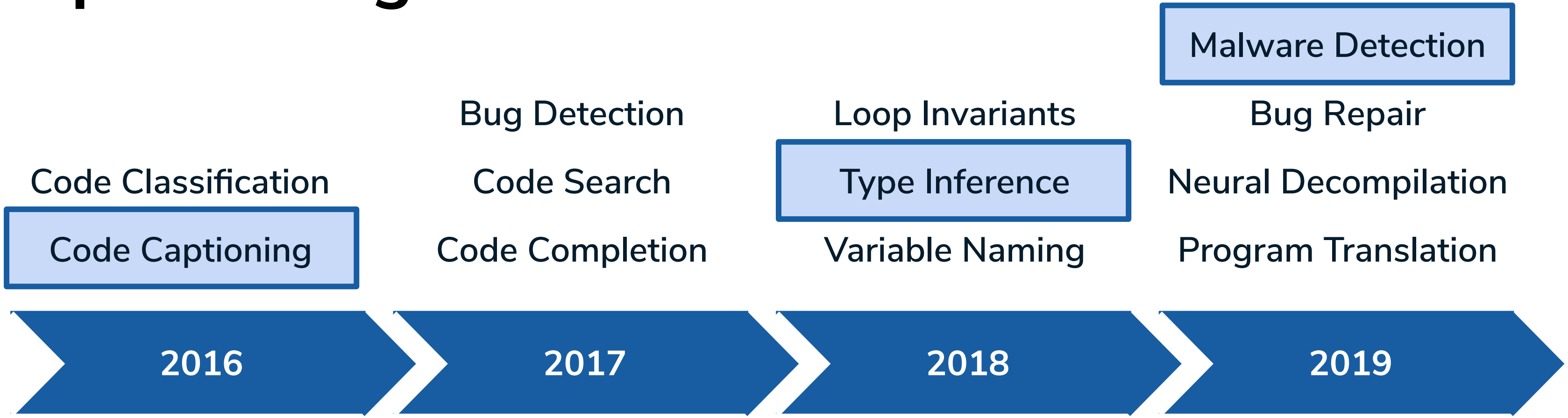
Neural Decompilation

Program Translation

2019

Techniques are general and apply to other tasks

Deep Learning + Code



2016

Code Classification

Code Captioning

2017

Bug Detection

Code Search

Code Completion

2018

Loop Invariants

Type Inference

Variable Naming

2019

Malware Detection

Bug Repair

Neural Decompilation

Program Translation

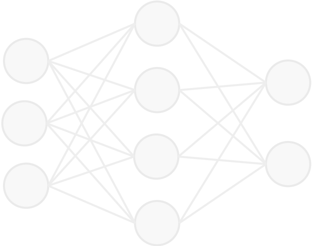
↑

Majority is based on deep learning models

Techniques are general and apply to other tasks

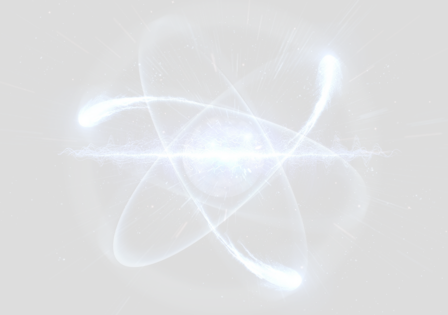
Secure, Reliable, and Intelligent Systems Lab @ ETH

Robust ML



Neural Network Verification

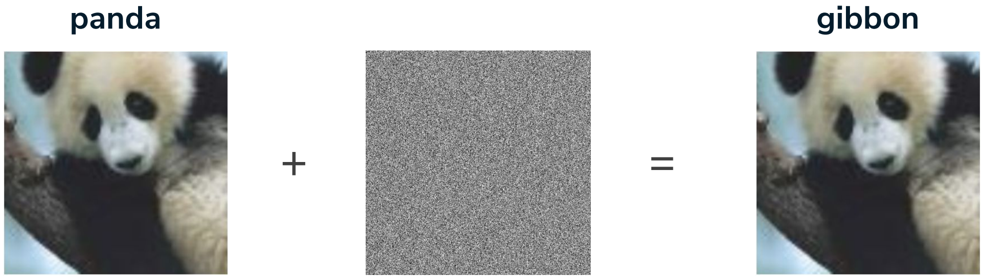
ML for programming



Probabilistic programming

Adversarial Robustness

Vision

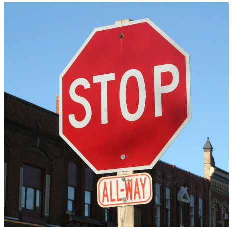


Explaining and Harnessing Adversarial Examples. Goodfellow et. al. ICLR’15

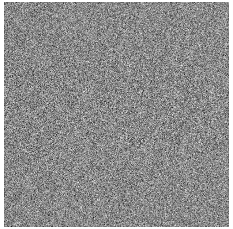
Adversarial Robustness

Vision

stop sign

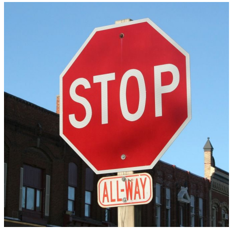


+



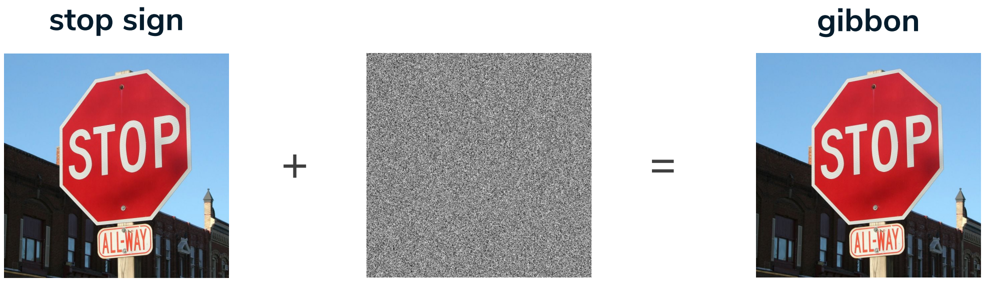
=

gibbon

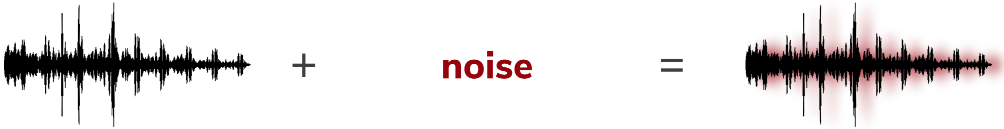


Adversarial Robustness

Vision



Sound



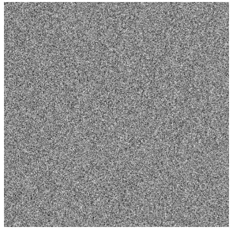
Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. Carlini et. al. ICML'18 workshop

Adversarial Robustness

Vision

stop sign

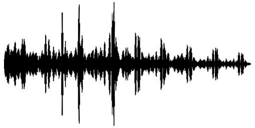
+



=

gibbon

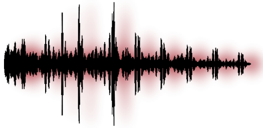
Sound



+

noise

=



Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. Carlini et. al. ICML'18 workshop

Code

+

code refactoring

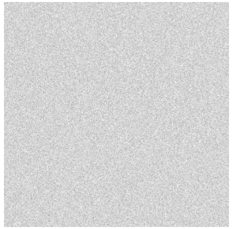
=

Adversarial Robustness

Vision

stop sign

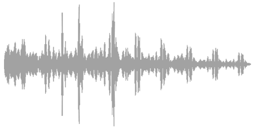
+



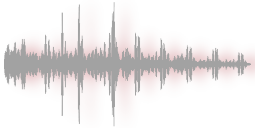
=

gibbon

Sound



+ noise =



Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. Carlini et. al. ICML’18 workshop

Code

+ **code refactoring** =

Adversarial Robustness for Code

1 **How robust are existing models?**

</> Code

90% accuracy + **code refactoring** = **??** robustness

Adversarial Robustness for Code

1 **How robust are existing models?**

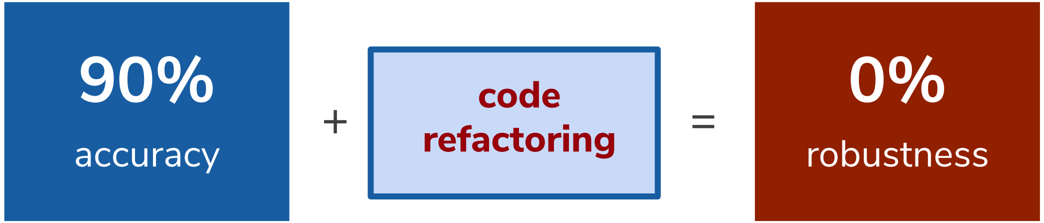
</> Code

90% accuracy + **code refactoring** = **0%** robustness

Adversarial Robustness for Code

1. How robust are existing models?
2. How to find adversarial examples?

</> Code



90% accuracy + code refactoring = 0% robustness

Adversarial Robustness for Code

1. How robust are existing models?
2. How to find adversarial examples?
3. How to improve robustness?

</> Code

90% accuracy + **code refactoring** = **0%** robustness ⇧

Adversarial Robustness Example

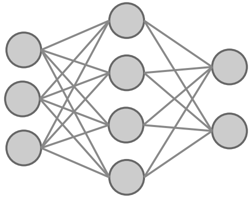
Input Program

x

```
...
v = parseInt(
  hex.substr(1),
  radix
)
...
```

Model

f(**x**) → **y**



(Type Inference)

Program Properties

y

```
...
v^num = parseInt^num(
  hex^str.substr^str(1),
  radix^num
)
...
```

Adversarial Robustness Example

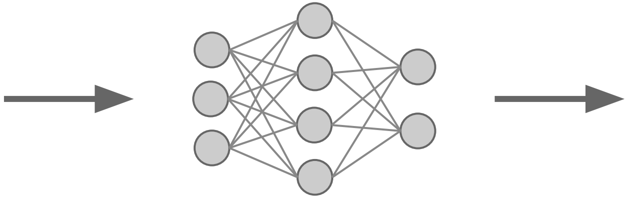
Input Program

x

```
...
v = parseInt(
  hex.substr(1),
  radix
)
...
```

Model

f(**x**) → **y**



(Type Inference)

Program Properties

y

$$\begin{array}{l}
\texttt{...} \\
\texttt{v}^{\texttt{num}}\texttt{ = parseInt}^{\texttt{num}}\texttt{(} \\
\quad\texttt{hex}^{\texttt{str}}\texttt{.substr}^{\texttt{str}}\texttt{(1),} \\
\quad\texttt{radix}^{\texttt{num}} \\
\texttt{)} \\
\texttt{...}
\end{array}$$

Goal (Adversarially Robustness):

Model is correct for *all* label preserving program transformations

```
...
v = parseInt(
  color.substr(1),
  radix
)
...
```

variable renaming

```
...
v = parseInt(
  hex.substr(42),
  radix
)
...
```

constant replacement

```
...
v = parseInt(
  hex.substr(1),
  radix + 0
)
...
```

semantic equivalence

```
...
parseInt(
  hex.substr(1),
  radix
)
...
```

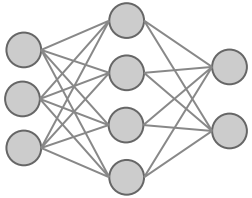
remove assignment

Adversarial Robustness Example

Input Program
x

```
...
v = parseInt(
  hex.substr(1),
  radix
)
...
```

Model
f(**x**) → **y**



(Type Inference)
Program Properties
y

```
...
v^num = parseInt^num(
  hex^str.substr^str(1),
  radix^num
)
...
```

Goal (Adversarially Robustness):
Model is correct for *all* label preserving program transformations

```
...
v = parseInt(
  color.substr(1),
  radix
)
...
```

variable renaming

```
...
v = parseInt(
  hex.substr(42),
  radix
)
...
```

constant replacement

```
...
v = parseInt(
  hex.substr(1),
  radix + 0
)
...
```

semantic equivalence

```
...
parseInt(
  hex.substr(1),
  radix
)
...
```

remove assignment

S(x)

Set of valid program transformations for **x**

How robust are existing models?

Type Inference [1]

```
...
v^num = parseInt^num(
  hex^str.substr^str(1),
  radix^num
)
...
```

[1] Adversarial Robustness for Code. ICML’20

How robust are existing models?

Type Inference [1]

```
...
vnum = parseIntnum(
  hexstr.substrstr(1),
  radixnum
)
...
```

Code Captioning [2,3]

```
int indexOfTarget(Object target) {
  int i = 0;
  for (Object elem: this.elements) {
    if (elem.equals(target))
      return i
    i++;
  return -1;
}
```

[1] Adversarial Robustness for Code. ICML'20

[2] Adversarial Examples for Models of Code. ArXiv'19

[3] Semantic Robustness of Models of Source Code. ArXiv'20

How robust are existing models?

Type Inference [1]

```
...
vnum = parseIntnum(
  hexstr.substrstr(1),
  radixnum
)
...
```

Code Captioning [2,3]

```
int indexOfTarget(Object target) {
  int i = 0;
  for (Object elem: this.elements) {
    if (elem.equals(target))
      return i
    i++;
  return -1;
}
```

Malware Detection [4]

```
push ebp
mov ebp, esp
push ebx
push edx                  yes
mov ebx, [ebp+4]          no
add ebx, 0x10
mov edx, [ebp+8]
mov [edx], ebx
```

[1] Adversarial Robustness for Code. ICML’20

[2] Adversarial Examples for Models of Code. ArXiv’19

[3] Semantic Robustness of Models of Source Code. ArXiv’20

[4] Optimization-Guided Binary Diversification to Mislead Neural Networks for Malware Detection. ArXiv’19

How robust are existing models?

Type Inference [1]

```
...
vnum = parseIntnum(
  hexstr.substrstr(1),
  radixnum
)
...
```

89% accuracy

48% robustness

Code Captioning [2,3]

```
int indexOfTarget(Object target) {
  int i = 0;
  for (Object elem: this.elements) {
    if (elem.equals(target))
      return i
    i++;
  return -1;
}
```

39.2 F1

19.6 robust F1

Malware Detection [4]

```
push ebp
mov ebp, esp
push ebx
push edx                      yes
mov ebx, [ebp+4]              no
add ebx, 0x10
mov edx, [ebp+8]
mov [edx], ebx
```

99% accuracy

~2% robustness

[1] Adversarial Robustness for Code. ICML’20

[2] Adversarial Examples for Models of Code. ArXiv’19

[3] Semantic Robustness of Models of Source Code. ArXiv’20

[4] Optimization-Guided Binary Diversification to Mislead Neural Networks for Malware Detection. ArXiv’19

How robust are existing models?

Not trained with robustness in mind

Type Inference [1]

```
}
...
```

89% accuracy

48% robustness

Code Captioning [2,3]

```
int indexOfTarget(Object target) {
    i++;
  return -1;
}
```

39.2 F1

19.6 robust F1

Malware Detection [4]

```
push ebp
add ebx, 0x10
mov edx, [ebp+8]
mov [edx], ebx
```

99% accuracy

~2% robustness

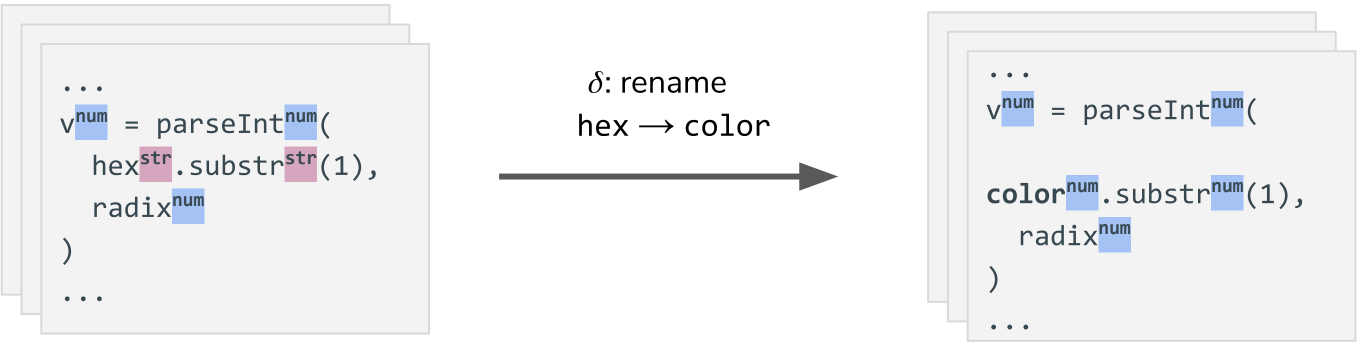
[1] Adversarial Robustness for Code. ICML'20

[2] Adversarial Examples for Models of Code. ArXiv'19

[3] Semantic Robustness of Models of Source Code. ArXiv'20

[4] Optimization-Guided Binary Diversification to Mislead Neural Networks for Malware Detection. ArXiv'19

So Far...



original programs

```
...
v^num = parseInt^num(
  hex^str.substr^str(1),
  radix^num
)
...
```

δ: rename

hex ⟶ color

transformed programs

```
...
v^num = parseInt^num(

color^num.substr^num(1),
  radix^num
)
...
```

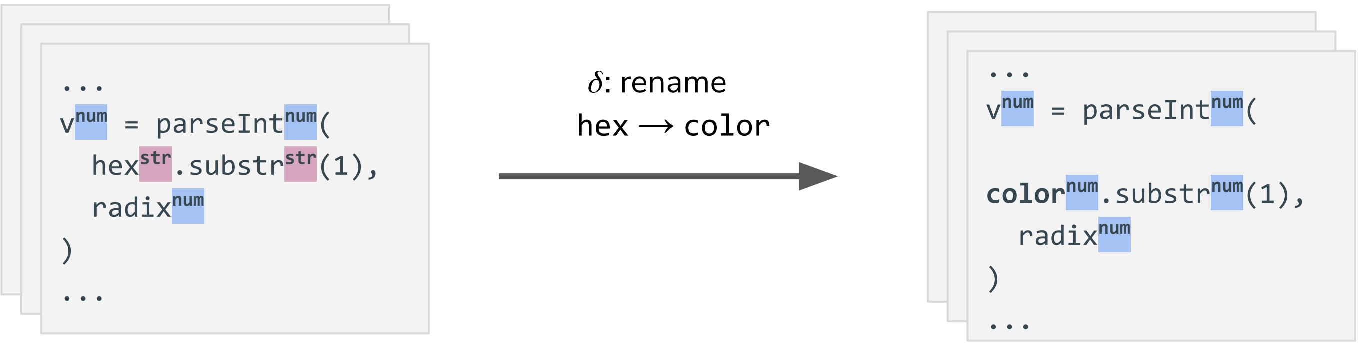
89%

accuracy

48%

robustness

Adversarial Training



original programs

```
...
v = parseInt(
  hex.substr(1),
  radix
)
...
```

δ: rename

`hex` → `color`

transformed programs

```
...
v = parseInt(

color.substr(1),
  radix
)
...
```

Standard training

$$\min \, loss(\theta, \mathbf{x}, \mathbf{y})$$

measures the model performance

Adversarial training

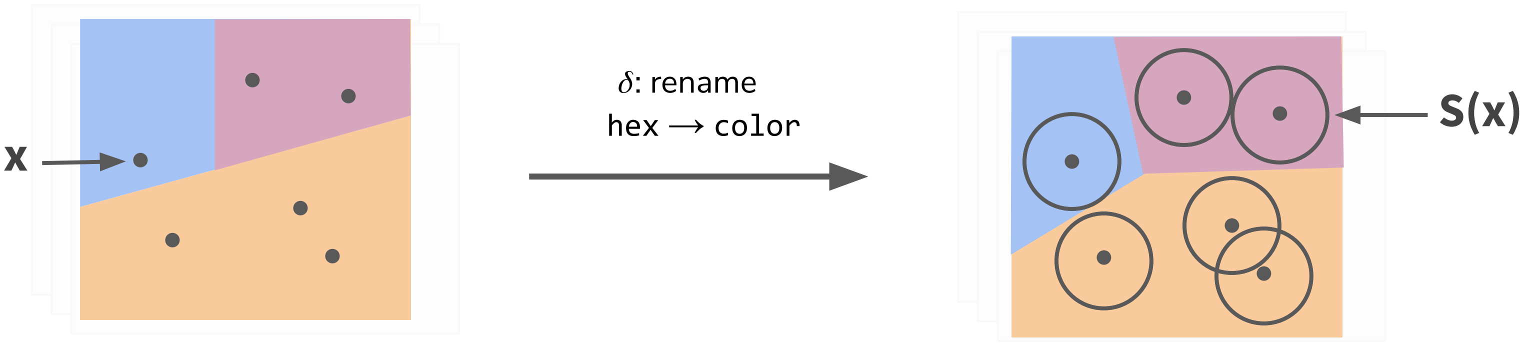
$$\min \left[\max_{\boldsymbol{\delta} \in \mathbf{S}(\mathbf{x})} loss(\theta, \mathbf{x} + \boldsymbol{\delta}, \mathbf{y}) \right]$$

program transformations

Adversarial Training

original programs

generated programs



Standard training

$$\min\ loss(\theta, \mathbf{x}, \mathbf{y})$$

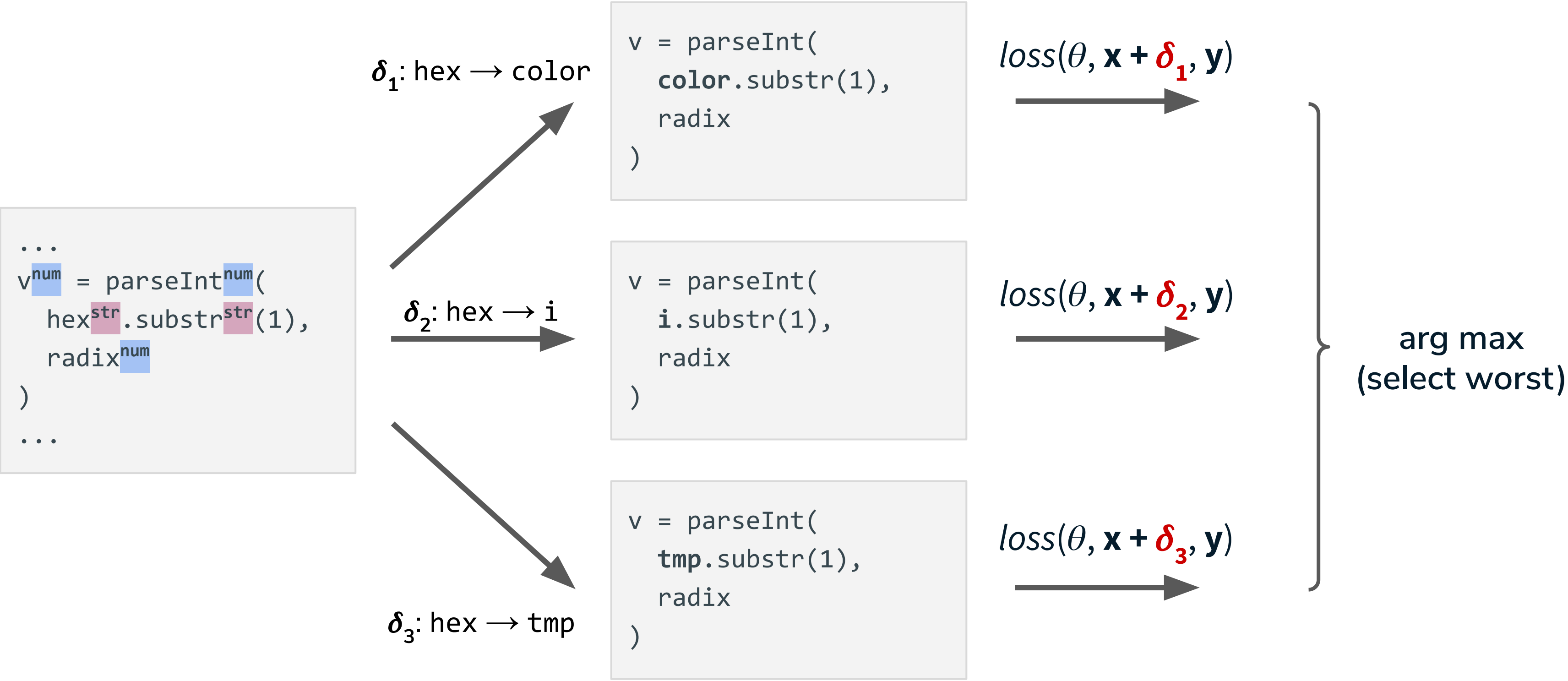
Adversarial training

$$\min\ [\max_{\boldsymbol{\delta} \in \mathbf{S}(\mathbf{x})} loss(\theta, \mathbf{x} + \boldsymbol{\delta}, \mathbf{y})]$$

trains on individual programs

trains on worst case generated programs

Finding Adversarial Examples



...

$\texttt{v}^{\texttt{num}}$ = $\texttt{parseInt}^{\texttt{num}}$(

  $\texttt{hex}^{\texttt{str}}$.$\texttt{substr}^{\texttt{str}}$(1),

  $\texttt{radix}^{\texttt{num}}$

)

...

$\boldsymbol{\delta}_1$: hex → color

```
v = parseInt(
  color.substr(1),
  radix
)
```

$loss(\theta, \mathbf{x} + \boldsymbol{\delta}_1, \mathbf{y})$

$\boldsymbol{\delta}_2$: hex → i

```
v = parseInt(
  i.substr(1),
  radix
)
```

$loss(\theta, \mathbf{x} + \boldsymbol{\delta}_2, \mathbf{y})$

$\boldsymbol{\delta}_3$: hex → tmp

```
v = parseInt(
  tmp.substr(1),
  radix
)
```

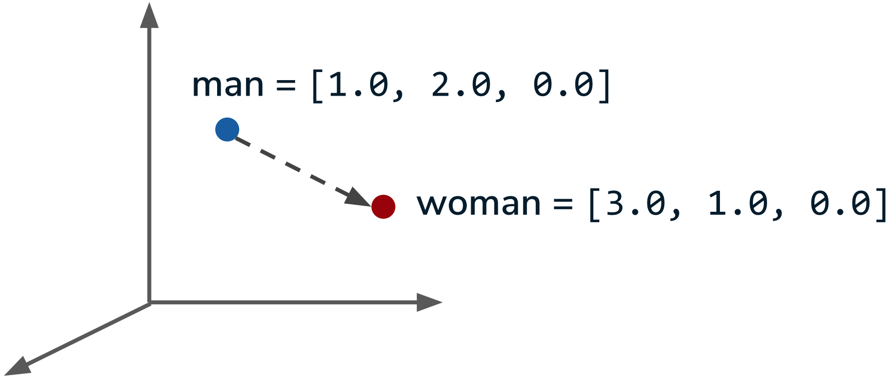
$loss(\theta, \mathbf{x} + \boldsymbol{\delta}_3, \mathbf{y})$

arg max
(select worst)

Basic approach: Try all valid modifications

Word Embeddings

maps each **discrete word** to a **continuous vector**



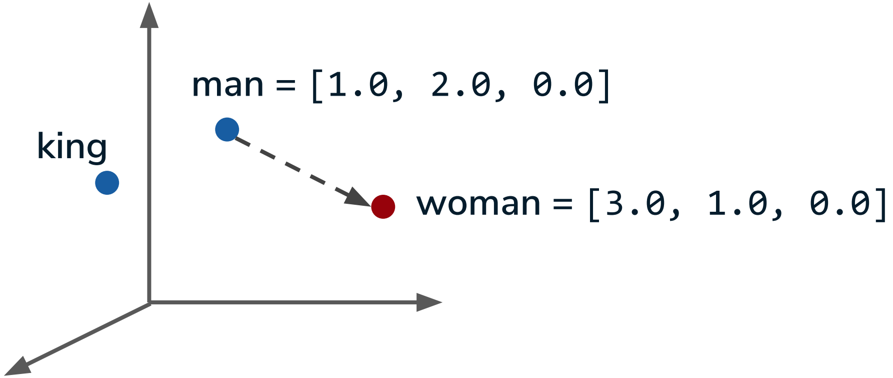
man = [1.0, 2.0, 0.0]

woman = [3.0, 1.0, 0.0]

Distributed Representations of Words and Phrases and their Compositionality. Mikolov et. al. NeurIPS’13

Word Embeddings

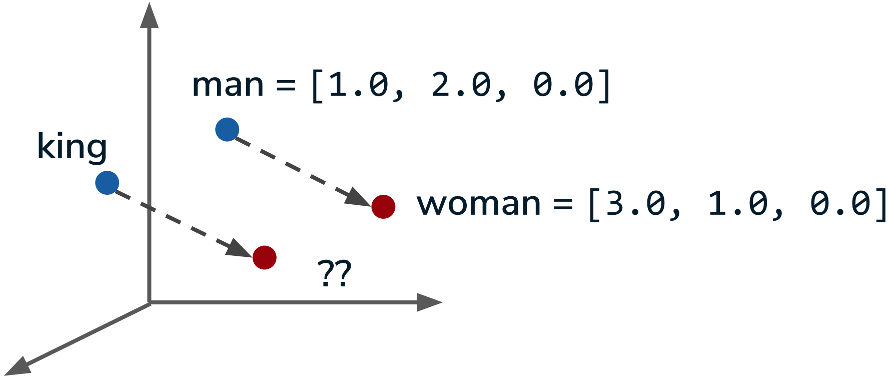
maps each **discrete word** to a **continuous vector**



Distributed Representations of Words and Phrases and their Compositionality. Mikolov et. al. NeurIPS’13

Word Embeddings

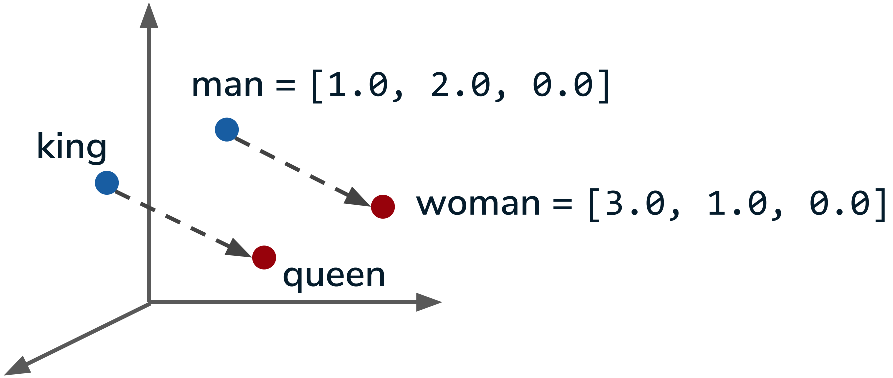
maps each **discrete word** to a **continuous vector**



Distributed Representations of Words and Phrases and their Compositionality. Mikolov et. al. NeurIPS'13

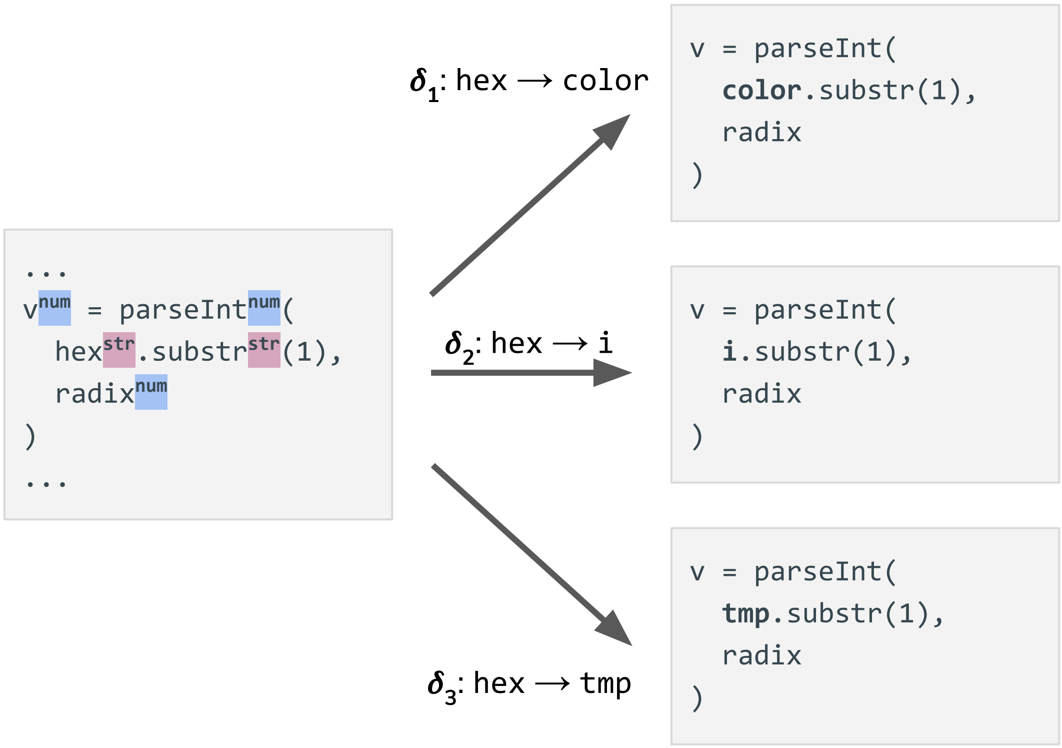
Word Embeddings

maps each **discrete word** to a **continuous vector**



Distributed Representations of Words and Phrases and their Compositionality. Mikolov et. al. NeurIPS'13

Finding Adversarial Examples



...
v^{num} = parseInt^{num}(
  hex^{str}.substr^{str}(1),
  radix^{num}
)
...

$\boldsymbol{\delta}_1$: hex ⟶ color

```
v = parseInt(
  color.substr(1),
  radix
)
```

$\boldsymbol{\delta}_2$: hex ⟶ i

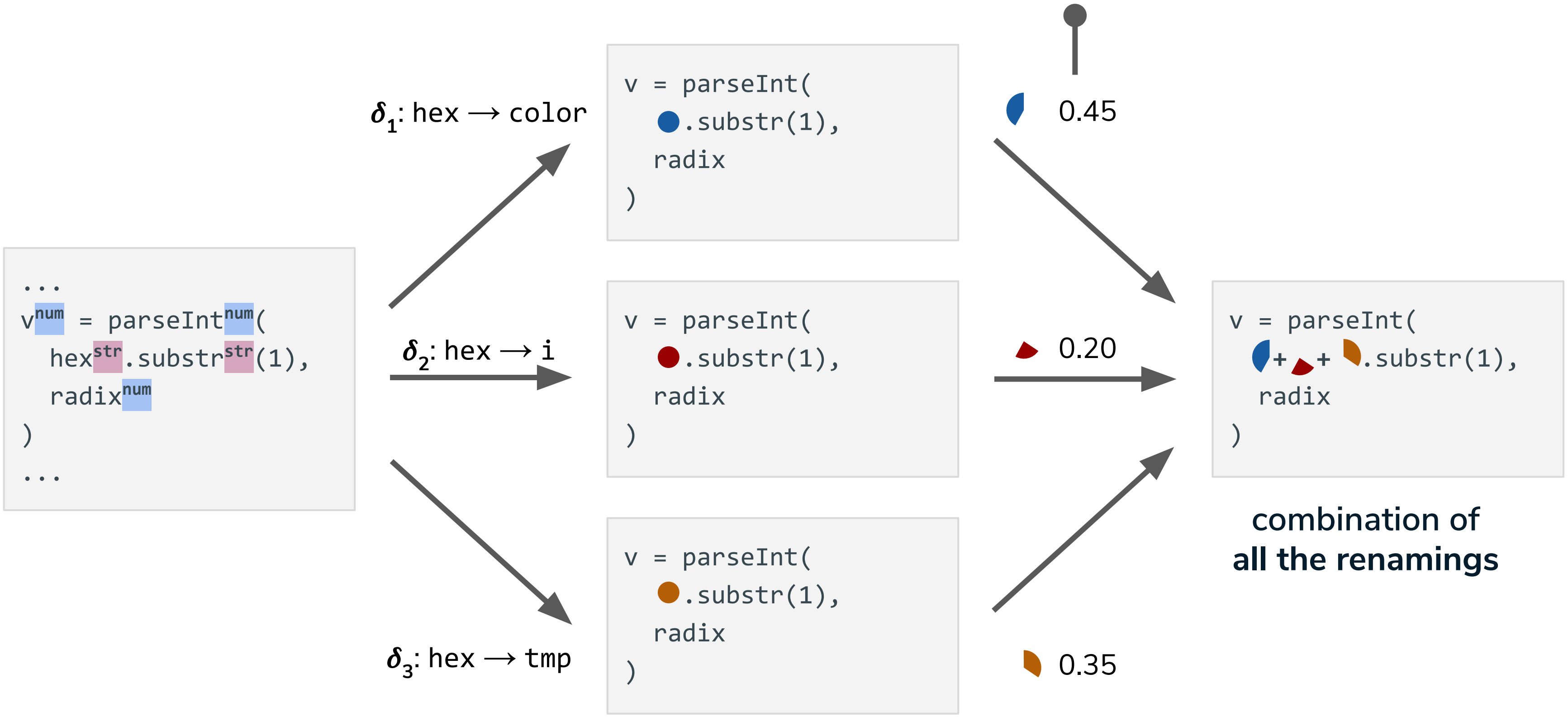
```
v = parseInt(
  i.substr(1),
  radix
)
```

$\boldsymbol{\delta}_3$: hex ⟶ tmp

```
v = parseInt(
  tmp.substr(1),
  radix
)
```

Finding Adversarial Examples

renaming weight



```
...
v = parseInt(
  hex.substr(1),
  radix
)
...
```

δ_1: hex → color

```
v = parseInt(
  ●.substr(1),
  radix
)
```

0.45

δ_2: hex → i

```
v = parseInt(
  ●.substr(1),
  radix
)
```

0.20

δ_3: hex → tmp

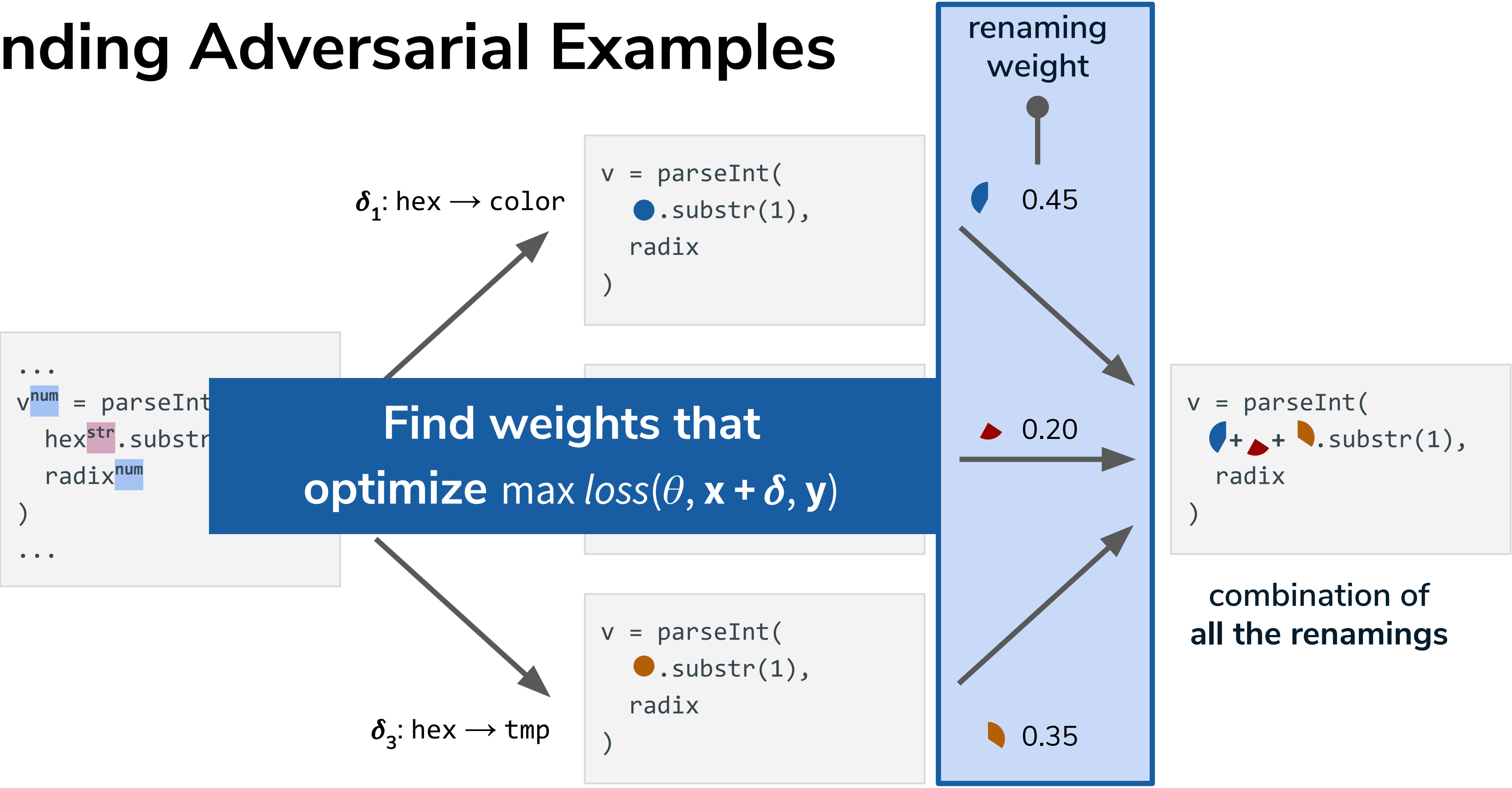
```
v = parseInt(
  ●.substr(1),
  radix
)
```

0.35

```
v = parseInt(
  ◖+◣+◗.substr(1),
  radix
)
```

combination of
all the renamings

Finding Adversarial Examples



```
...
v = parseInt
  hex.substr
  radix
)
...
```

δ_1: hex → color

```
v = parseInt(
  ●.substr(1),
  radix
)
```

δ_3: hex → tmp

```
v = parseInt(
  ●.substr(1),
  radix
)
```

Find weights that optimize $\max loss(\theta, \mathbf{x} + \delta, \mathbf{y})$

```
v = parseInt(
  ● + ● + ●.substr(1),
  radix
)
```

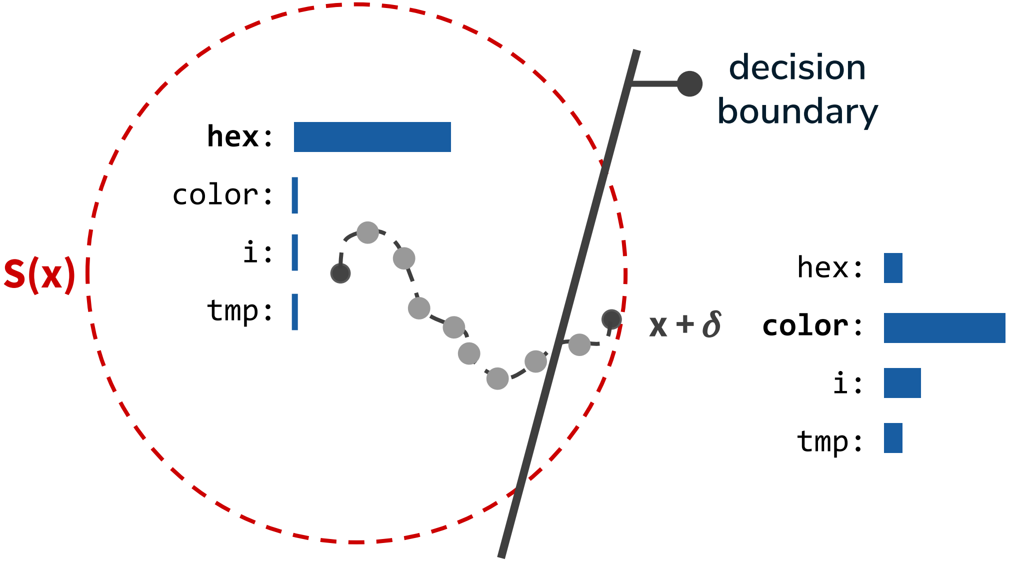
combination of
all the renamings

Solving the Inner max *loss* Efficiently

Gradient Based Optimization

$$\theta \leftarrow \theta - \nabla_{\boldsymbol{\delta}}\, loss(\theta, \mathbf{x} + \boldsymbol{\delta}, \mathbf{y})$$

$\boldsymbol{\delta} \in \mathbf{S(x)}$



Adversarial Examples for Models of Code.
Yefet et. al. ArXiv'20

Robustness of Existing Models

Type Inference [1]

```
...
v^num = parseInt^num(
  hex^str.substr^str(1),
  radix^num
)
...
```

89% accuracy

45% robustness

Adversarial training

57% robustness

Code Captioning [2,3]

```
int indexOfTarget(Object target) {
  int i = 0;
  for (Object elem: this.elements) {
    if (elem.equals(target))
      return i
    i++;
  return -1;
}
```

39.2 F1

19.6 robust F1

Malware Detection [4]

```
push ebp
mov ebp, esp
push ebx
push edx                yes
mov ebx, [ebp+4]        no
add ebx, 0x10
mov edx, [ebp+8]
mov [edx], ebx
```

99% accuracy

~2% robustness

Robustness of Existing Models

Type Inference [1]

```
...
vnum = parseIntnum(
  hexstr.substrstr(1),
  radixnum
)
...
```

89%	45%
accuracy	robustness

Adversarial training

57%
robustness

Code Captioning [2,3]

```
int indexOfTarget(Object target) {
  int i = 0;
  for (Object elem: this.elements) {
    if (elem.equals(target))
      return i
    i++;
  return -1;
}
```

39.2	19.6
F1	robust F1
40.6	27.5
F1	robust F1

Malware Detection [4]

```
push ebp
mov ebp, esp
push ebx
push edx                   yes
mov ebx, [ebp+4]           no
add ebx, 0x10
mov edx, [ebp+8]
mov [edx], ebx
```

99%	~2%
accuracy	robustness

Robustness of Existing Models

Type Inference [1]

```
...
v^num = parseInt^num(
  hex^str.substr^str(1),
  radix^num
)
...
```

89% accuracy

45% robustness

Adversarial training

57% robustness

Code Captioning [2,3]

```
int indexOfTarget(Object target) {
  int i = 0;
  for (Object elem: this.elements) {
    if (elem.equals(target))
      return i
    i++;
  return -1;
}
```

39.2 F1

19.6 robust F1

40.6 F1

27.5 robust F1

Malware Detection [4]

```
push ebp
mov ebp, esp
push ebx
push edx                yes
mov ebx, [ebp+4]        no
add ebx, 0x10
mov edx, [ebp+8]
mov [edx], ebx
```

99% accuracy

~2% robustness

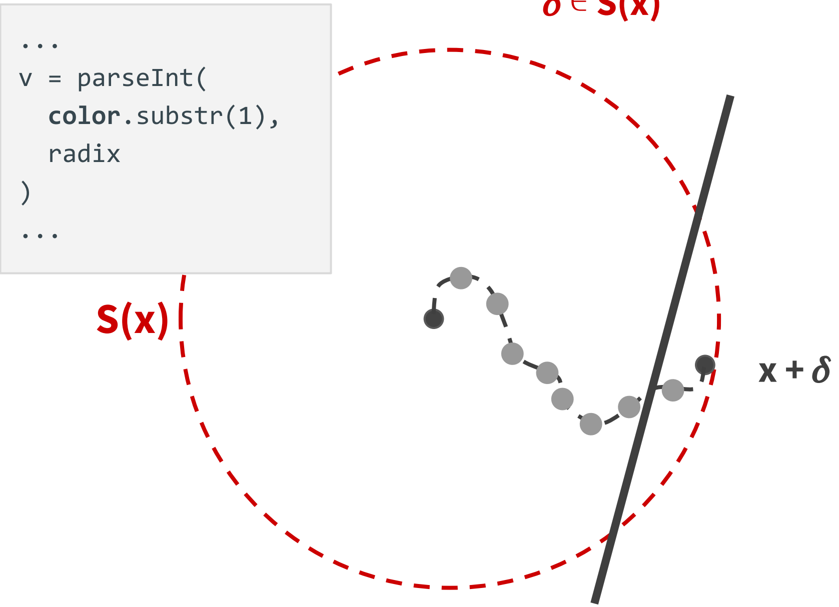
too expensive

Solving the Inner max *loss* Efficiently

Gradient Based Optimization

$$\theta \leftarrow \theta - \nabla_{\boldsymbol{\delta}}\, loss(\theta, \mathbf{x} + \boldsymbol{\delta}, \mathbf{y})$$

$$\boldsymbol{\delta} \in \mathbf{S}(\mathbf{x})$$

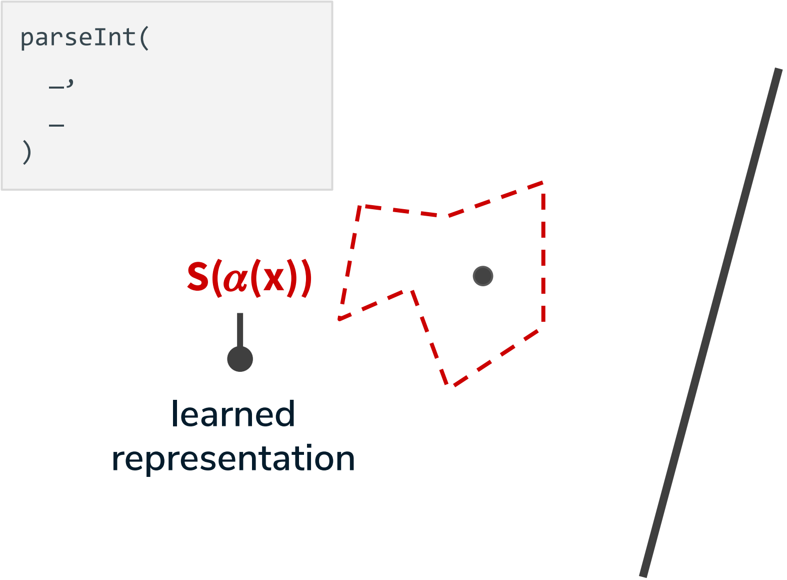


```
...
v = parseInt(
  color.substr(1),
  radix
)
...
```

Refine S

$$\min\,[\max\, loss(\theta, \mathbf{x} + \boldsymbol{\delta}, \mathbf{y})]$$

$$\boldsymbol{\delta} \in \mathbf{S}(\alpha(\mathbf{x}))$$



```
parseInt(
  _,
  _
)
```

Adversarial Robustness for Code.
Bielik et. al. ICML’20

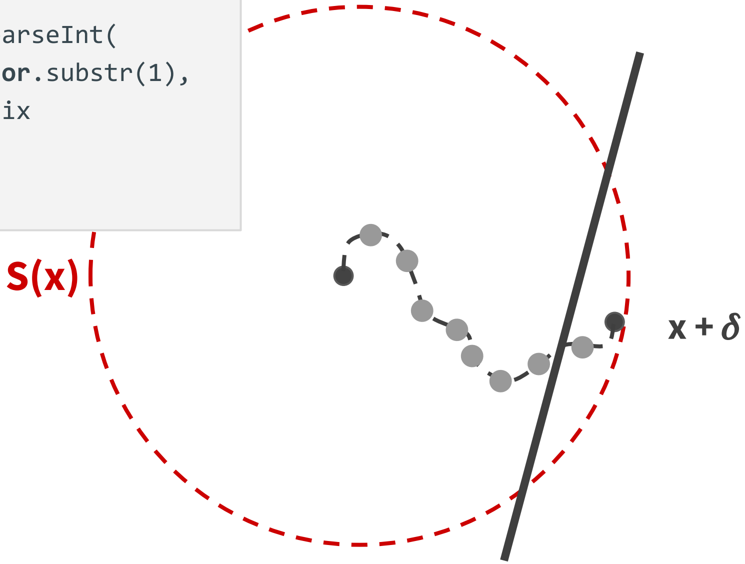
Solving the Inner max *loss* Efficiently

Gradient Based Optimization

$$\theta \leftarrow \theta - \nabla_{\delta}\, loss(\theta, \mathbf{x} + \delta, \mathbf{y})$$

$\delta \in \mathbf{S}(\mathbf{x})$

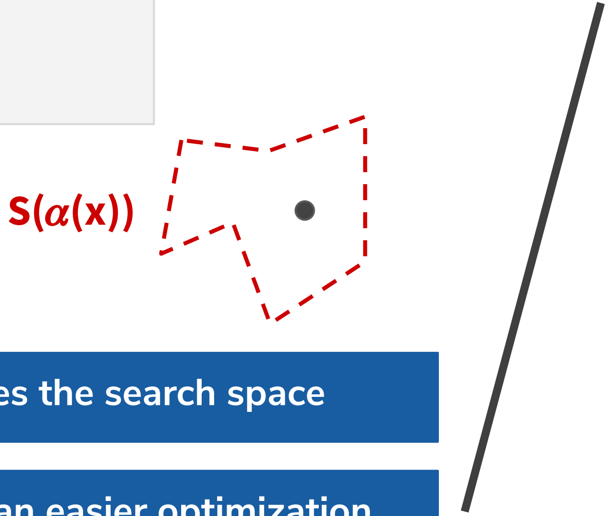
```
...
v = parseInt(
  color.substr(1),
  radix
)
...
```



Refine S

$$\min \left[\max_{\delta \in \mathbf{S}(\alpha(\mathbf{x}))} loss(\theta, \mathbf{x} + \delta, \mathbf{y}) \right]$$

```
parseInt(
  _,
  _
)
```



reduces the search space

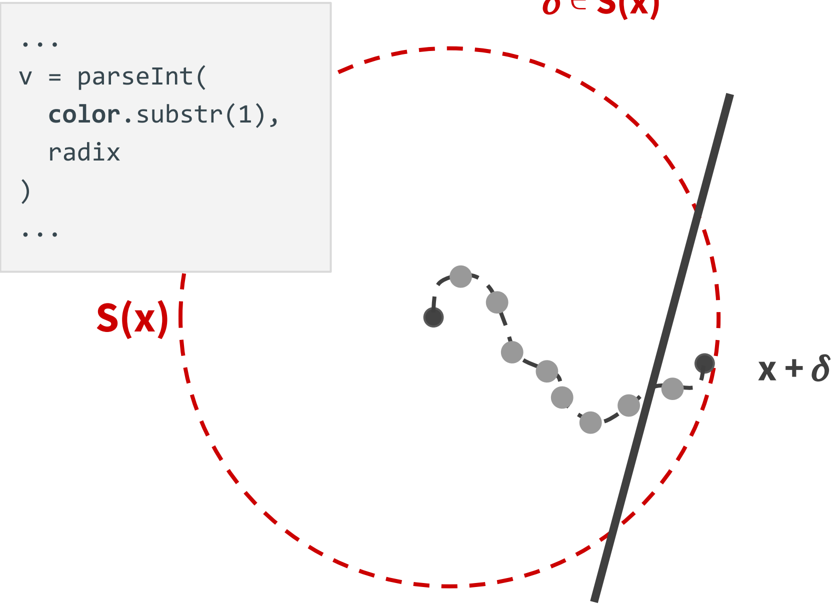
leads to an easier optimization

Solving the Inner max *loss* Efficiently

Gradient Based Optimization

$$\theta \leftarrow \theta - \nabla_{\delta}\, loss(\theta, \mathbf{x} + \delta, \mathbf{y})$$

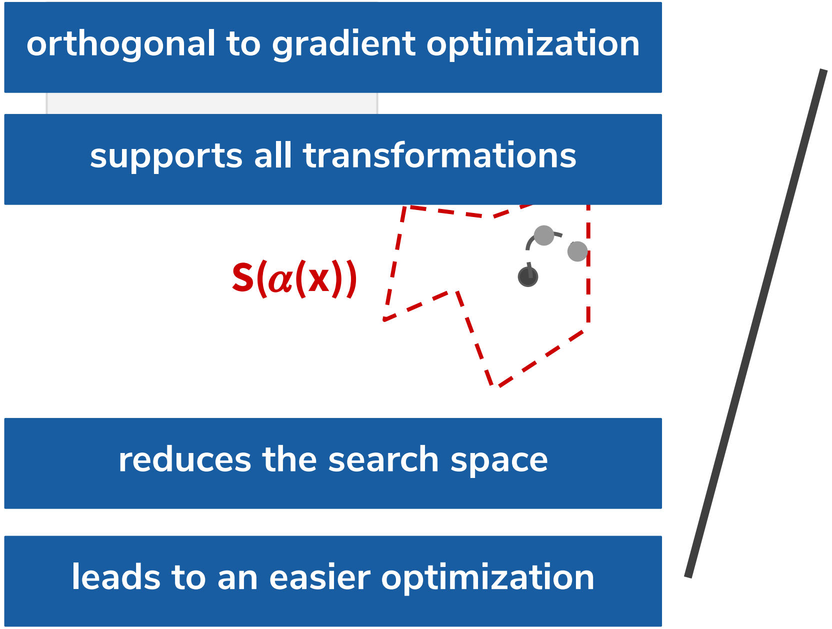
$$\delta \in \mathbf{S}(\mathbf{x})$$



```
...
v = parseInt(
  color.substr(1),
  radix
)
...
```

Refine S

$$\min\,[\max\limits_{\delta \in \mathbf{S}(\alpha(\mathbf{x}))} loss(\theta, \mathbf{x} + \delta, \mathbf{y})]$$



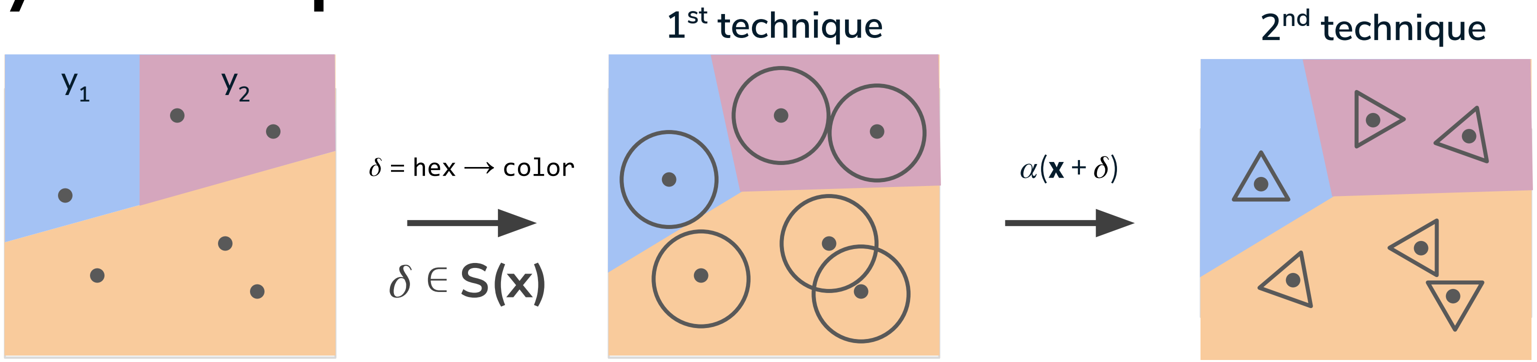
orthogonal to gradient optimization

supports all transformations

reduces the search space

leads to an easier optimization

Key Techniques



1st technique

2nd technique

δ = hex ⟶ color

$\delta \in \mathbf{S}(\mathbf{x})$

$\alpha(\mathbf{x} + \delta)$

Standard Training

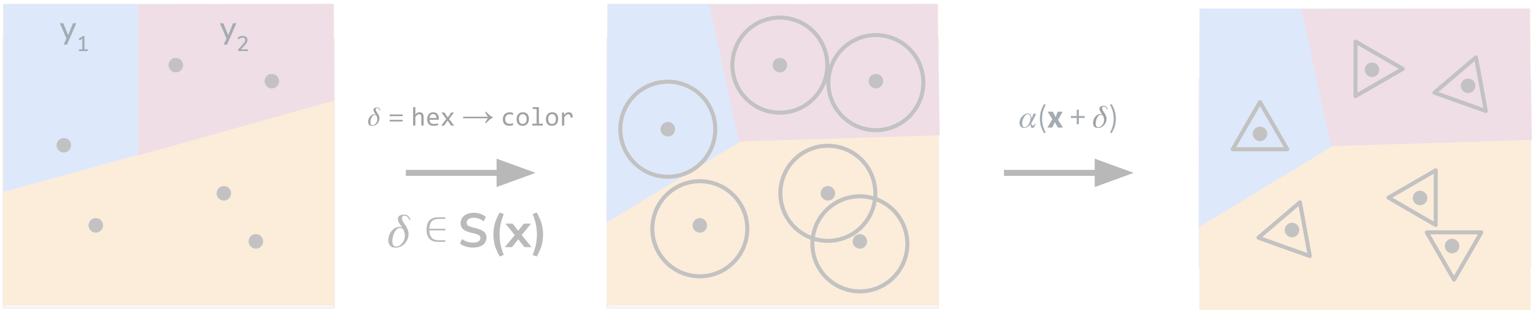
Adversarial Training

Representation Learning

Key Techniques

1st technique

2nd technique

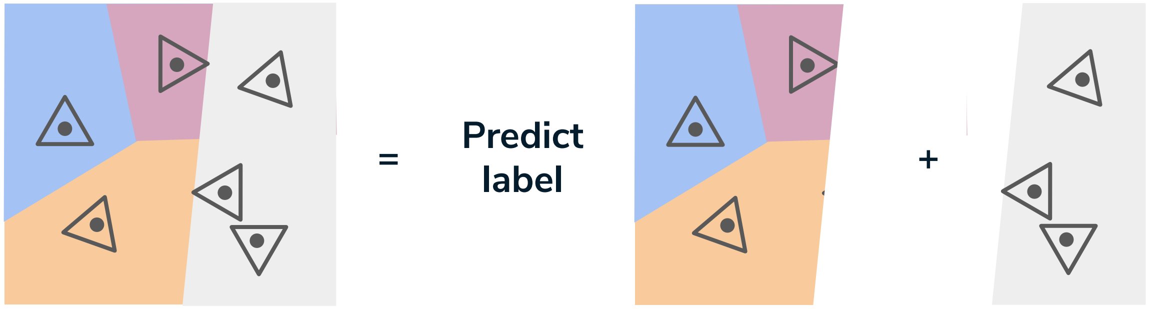

y_1
y_2
δ = hex ⟶ color
$\delta \in \mathbf{S}(\mathbf{x})$
$\alpha(\mathbf{x} + \delta)$

Standard Training

Adversarial Training

Representation Learning

3rd technique



Abstain

Robust & Accurate

Robust

Robustness of Existing Models

Type Inference [1]

89%
accuracy

45%
robustness

Adversarial training

57%
robustness

+
Refinement & Abstain

67%
robustness

Code Captioning [2,3]

39.2
F1

19.6
robust F1

27.5
robust F1

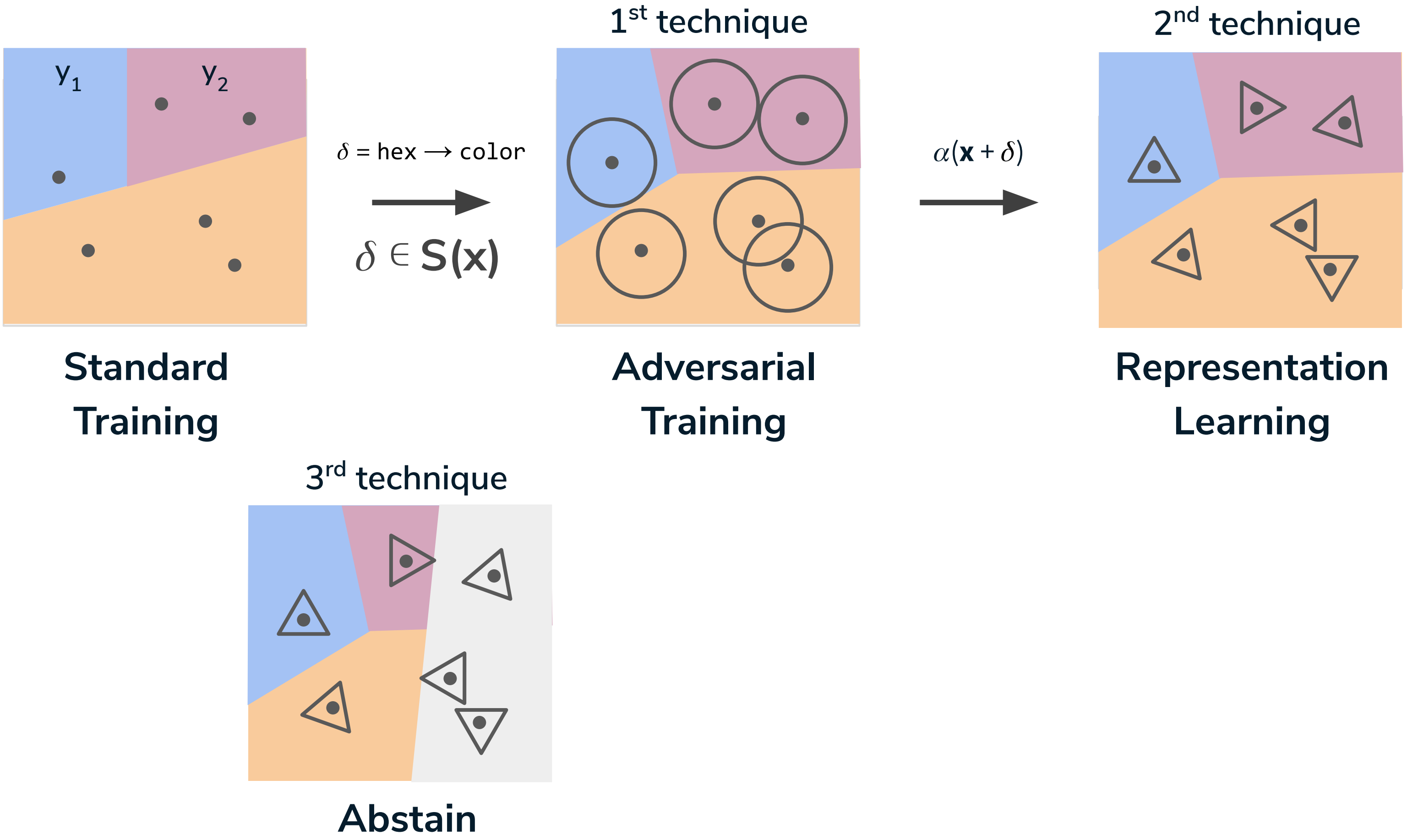
Malware Detection [4]

99%
accuracy

~2%
robustness

too expensive

Robust Models for Source Code



Standard Training

δ = hex ⟶ color

$\delta \in \mathbf{S(x)}$

1st technique

Adversarial Training

$\alpha(\mathbf{x} + \delta)$

2nd technique

Representation Learning

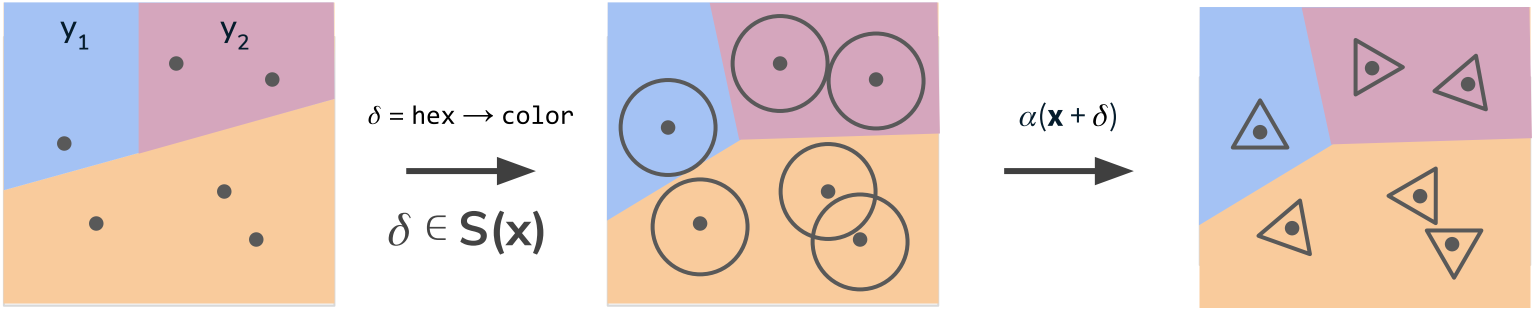
3rd technique

Abstain

Robust Models for Source Code

1st technique

2nd technique

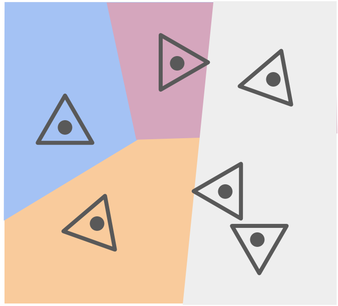


Standard Training

Adversarial Training

Representation Learning

3rd technique



Abstain

Next technique?

pre-processing

compositionality

normalization