ICML 2020

Adversarial Robustness for Code

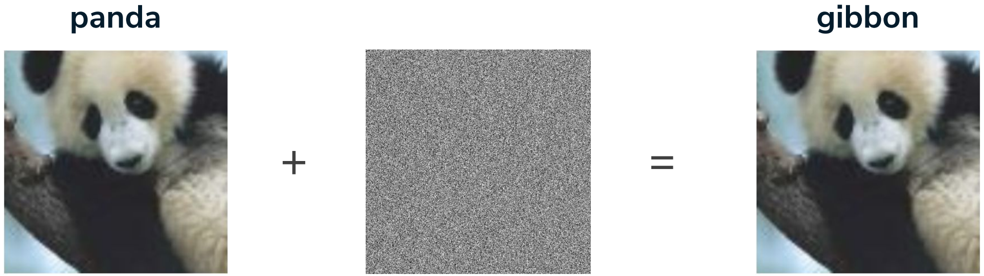
Pavol Bielik, Martin Vechev
pavol.bielik@inf.ethz.ch, martin.vechev@inf.ethz.ch

Department of Computer Science
ETH zürich

SRILAB

Adversarial Robustness

Vision



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

Sound

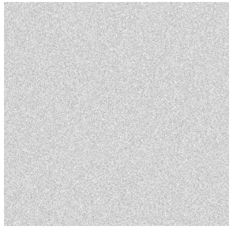
Audio Adversarial Examples: Targeted Attacks on Speech-to-Text. Carlini et. al. ICML’18 workshop

Adversarial Robustness for Code

Vision

panda

+

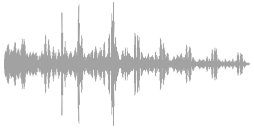


=

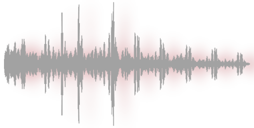
gibbon

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

Sound



+ noise =



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

Code

+ **code refactoring** =

Deep Learning + Code

2016	2017	2018	2019
	Bug Detection	Loop Invariants	Bug Repair
Code Classification	Code Search	Type Inference	Neural Decompilation
Code Captioning	Code Completion	Variable Naming	Program Translation

Prior Works

90%

Accuracy

Adversarial Robustness for Code

2016	2017	2018	2019
	Bug Detection	Loop Invariants	Bug Repair
Code Classification	Code Search	Type Inference	Neural Decompilation
Code Captioning	Code Completion	Variable Naming	Program Translation

Prior Works

90%

Accuracy

?

Robustness

Adversarial Robustness for Code

2016	2017	2018	2019
	Bug Detection	Loop Invariants	Bug Repair
Code Classification	Code Search	Type Inference	Neural Decompilation
Code Captioning	Code Completion	Variable Naming	Program Translation

Prior Works

90% Accuracy

4%-50% Robustness

This Work

88% Accuracy

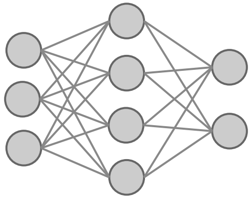
84% Robustness

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]
)
...
```

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

Our Work: Three Key Techniques

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

1 **Abstain**

Allows model
***not* to make**
a prediction
if uncertain

Our Work: Three Key Techniques

54% robustness	δ = `hex` $\longrightarrow$ `color`	54% robustness

1 **Abstain**

2 **Adversarial Training**

Our Work: Three Key Techniques

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

δ = hex $\longrightarrow$ color

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

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

```
parseInt^num(
  _,
  _
)
```

1. Abstain
2. Adversarial Training
3. Representation Learning

Our Work: Three Key Techniques

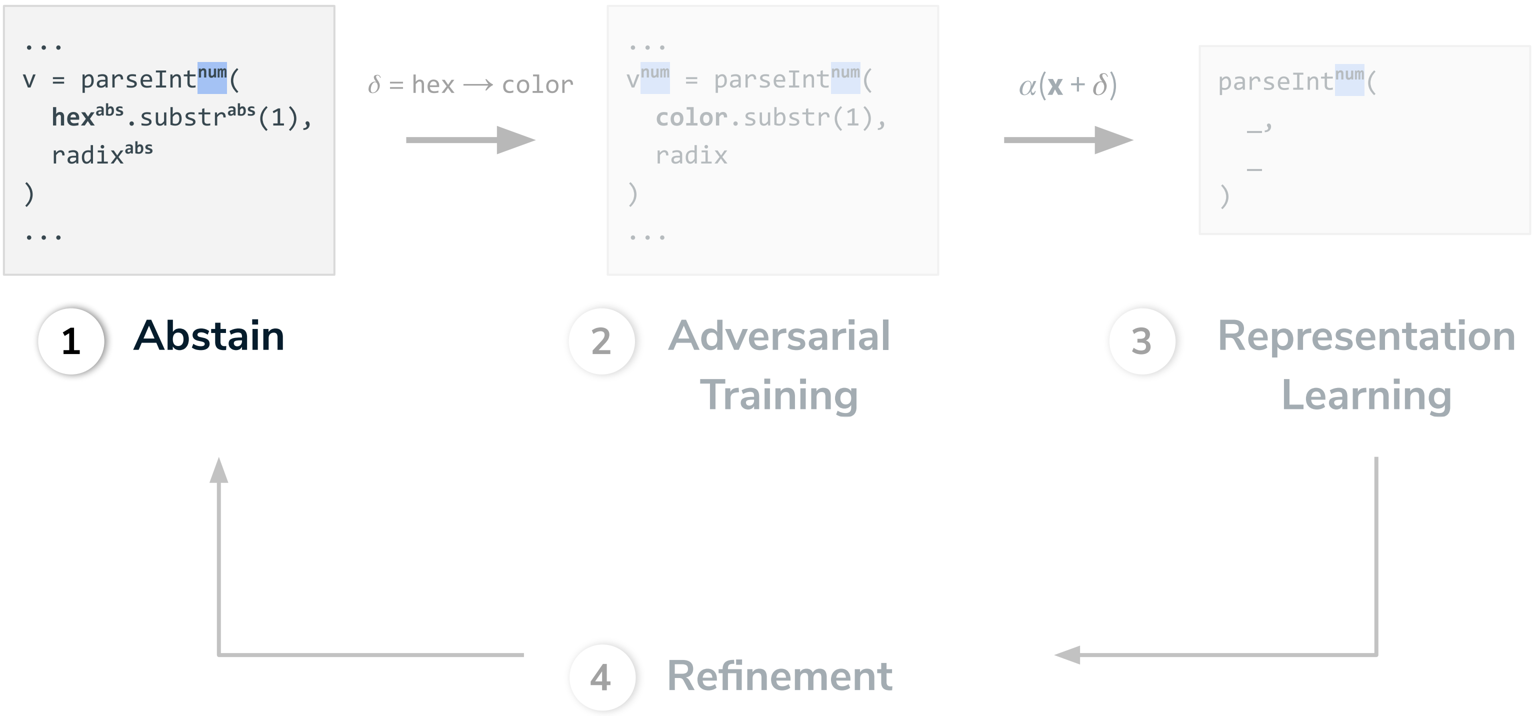
$$
\begin{array}{l}
\texttt{...} \\
\texttt{v = parseInt(} \\
\quad \textbf{hex}^{\text{abs}}\texttt{.substr}^{\text{abs}}\texttt{(1),} \\
\quad \texttt{radix}^{\text{abs}} \\
\texttt{)} \\
\texttt{...}
\end{array}
\xrightarrow{\delta = \text{hex} \longrightarrow \texttt{color}}
\begin{array}{l}
\texttt{...} \\
\texttt{v}^{\text{num}}\texttt{ = parseInt}^{\text{num}}\texttt{(} \\
\quad \textbf{color}\texttt{.substr(1),} \\
\quad \texttt{radix} \\
\texttt{)} \\
\texttt{...}
\end{array}
\xrightarrow{\alpha(\mathbf{x} + \delta)}
\boxed{\begin{array}{c} \textbf{84\%} \\ \text{robustness} \end{array}}
$$

1 **Abstain**

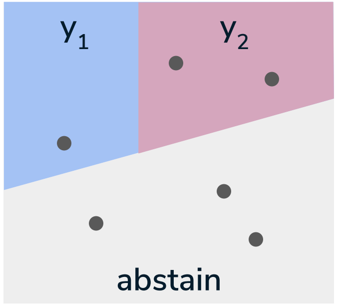
2 **Adversarial Training**

3 **Representation Learning**

Our Work: Three Key Techniques

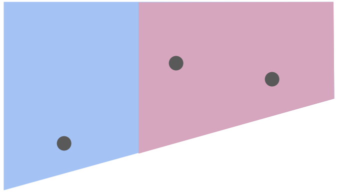


Learning to Abstain



● input x_i

=



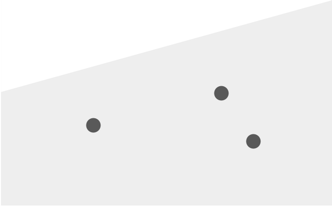
Model should be both Robust and Accurate

Predict Class

+

Abstains

Model should be only Robust



Leads to a simpler optimization problem

Property prediction problem is undecidable

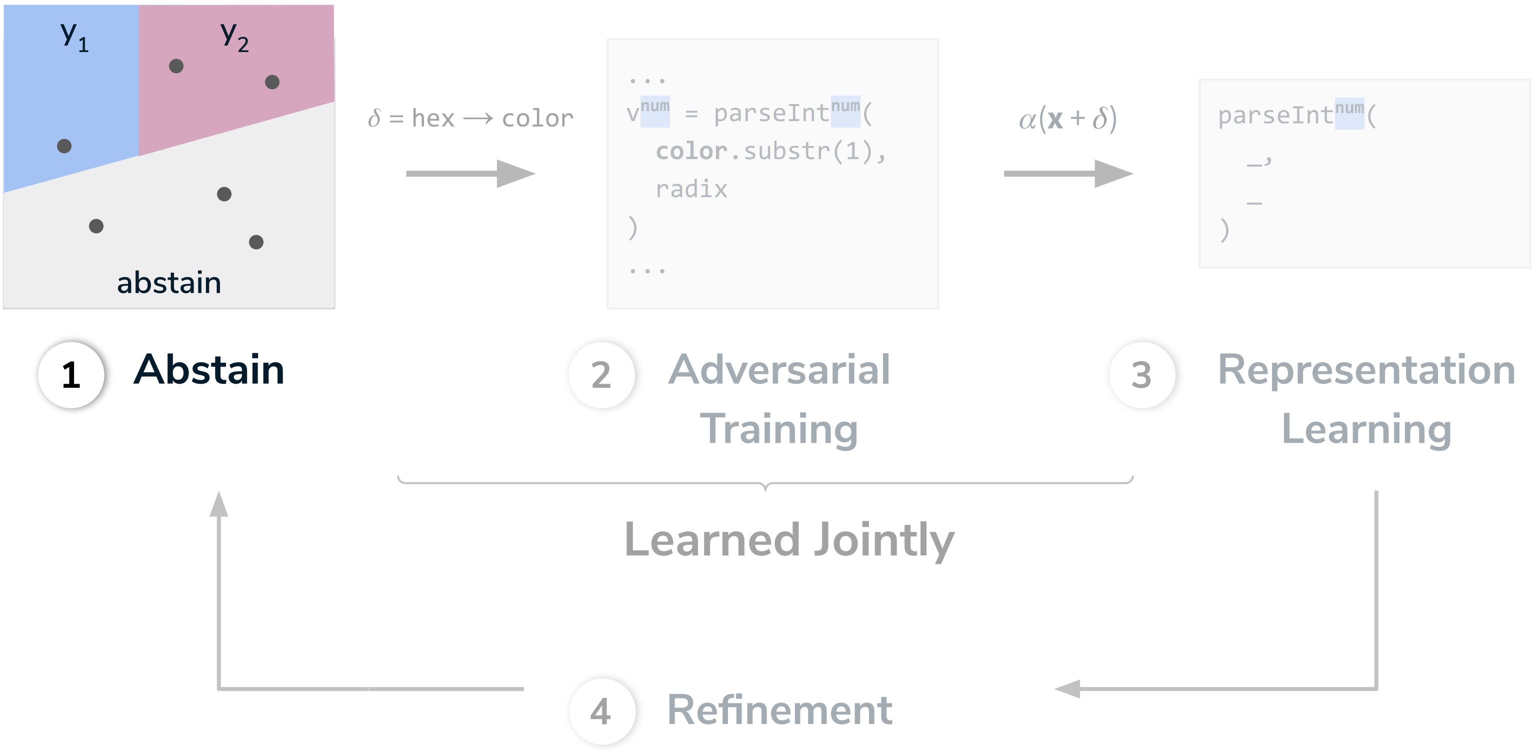
Learning to Abstain

Main Insight	Combine Robustness + Learning to Abstain
How to Abstain?	Deep Gamblers: Learning to Abstain with Portfolio Theory. Liu et. al. NeurIPS'19

Leads to a simpler optimization problem

Property prediction problem is undecidable

Our Work: Three Key Techniques



Adversarial Training

measures the model performance   ground-truth label

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

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

Label preserving program transformations

1 **Define the space S of program transformations**

2 **Solve the inner max *loss* efficiently**

Label Preserving Program Transformations

Word Substitution

tensors + δ

very fast

Constants, Binary Operators, ...

```
7
radix + offset
```

$\mathbf{x} + \delta$

```
42
radix - offset
```

Word Renaming

tensors + δ + analysis

fast

Rename Variables, Parameters, Fields, Method Names, ...

```
def getID() {...}
client.Name
```

$\mathbf{x} + \delta$

```
def get_id() {...}
client.name
```

Sequence Substitution

tensors→code + δ + analysis→tensors

slow

Adding Dead Code, Reordering Statements, ...

```
a = get_id()
b = 42
```

$\mathbf{x} + \delta$

```
b = 42
a = get_id()
```

Adversarial Training

measures the model performance

ground-truth label

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

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

Label preserving program transformations

1 **Define the space S of program transformations**

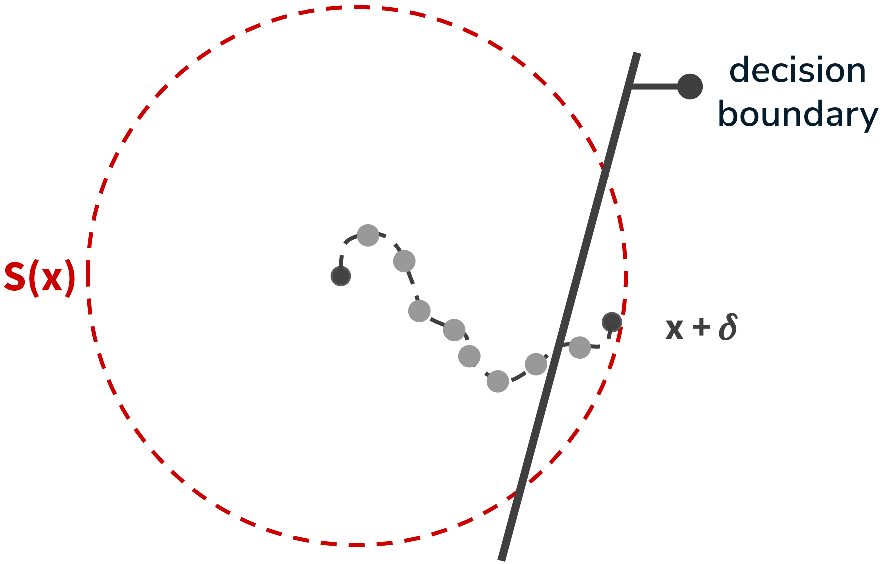
2 **Solve the inner max *loss* efficiently**

Solving the Inner max *loss* Efficiently

Gradient Based Optimization

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

$\delta \in$ **S(x)**



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

Limitations

54% → **54%**

standard | adversarial

same or worse robustness

Discrete and disruptive changes

Highly structured and large programs

hard optimization problem

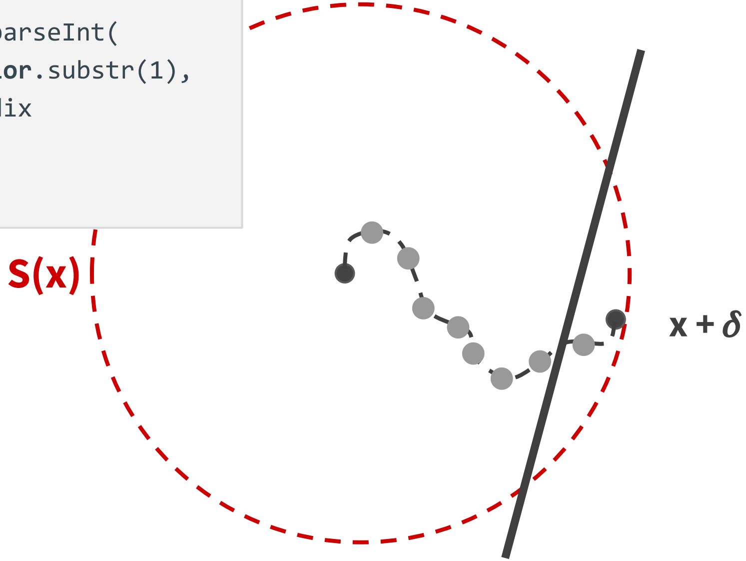
no structural transformations

Solving the Inner max *loss* Efficiently

Gradient Based Optimization

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

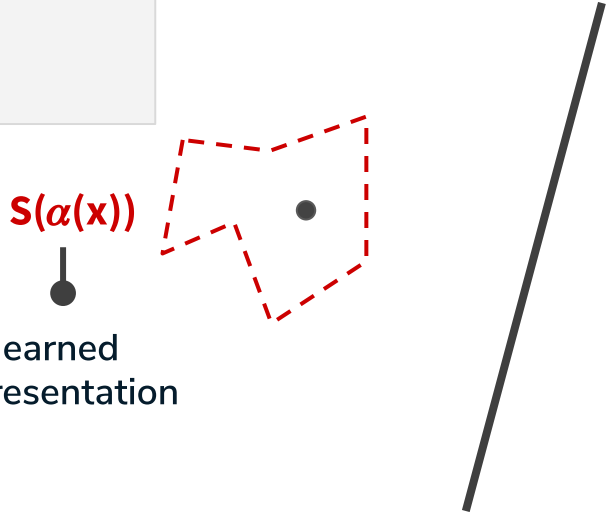
```
...
v = parseInt(
  color.substr(1),
  radix
)
...
```



Refine **S**

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

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



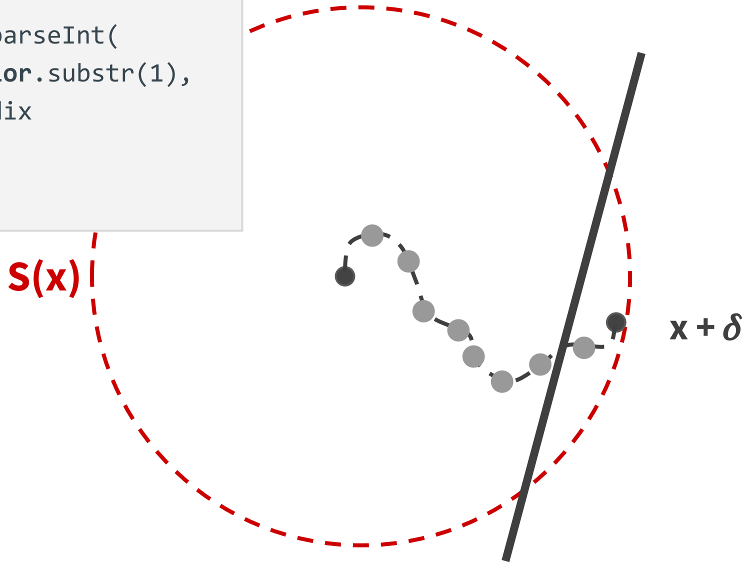
Solving the Inner max *loss* Efficiently

Gradient Based Optimization

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

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

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

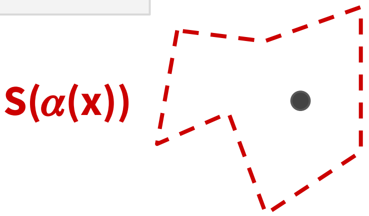


Refine S

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

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

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



reduces the search space

leads to an easier optimization

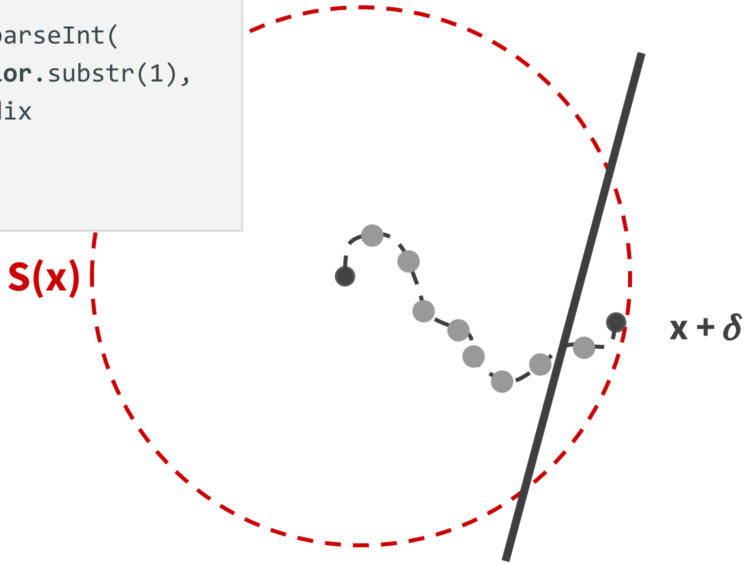
Solving the Inner max *loss* **Efficiently**

Gradient Based Optimization

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

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

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

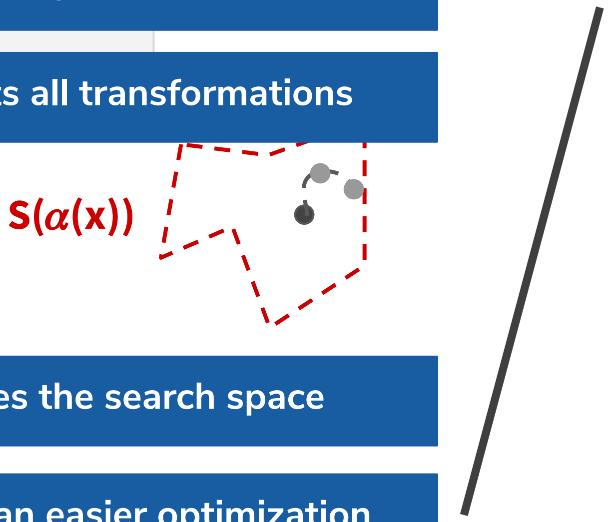


Refine S

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

orthogonal to gradient optimization

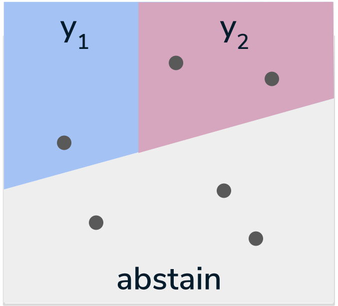
supports all transformations



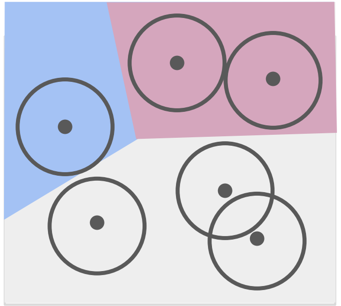
reduces the search space

leads to an easier optimization

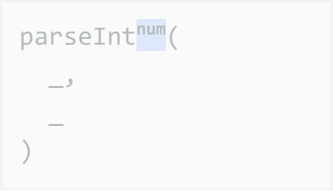
Our Work: Three Key Techniques



δ = hex $\longrightarrow$ color



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



1 **Abstain**

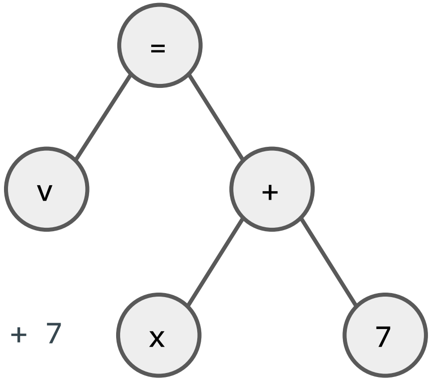
2 **Adversarial Training**

3 Representation Learning

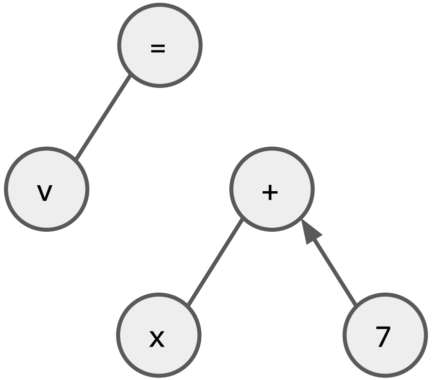
Learned Jointly

4 Refinement

Representation Learning



`v = x + 7`



nodes attributes

$$G = \langle V,\ E,\ \xi \rangle$$

edges

$$\alpha : \langle V, E, \xi \rangle \rightarrow \langle V, E' \subseteq E, \xi \rangle$$

1 **Programs as Graphs**

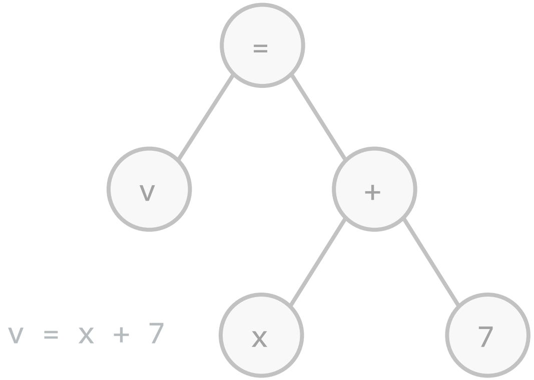
Learning to Represent Programs with Graphs. Allamanis et. al. ICLR'18

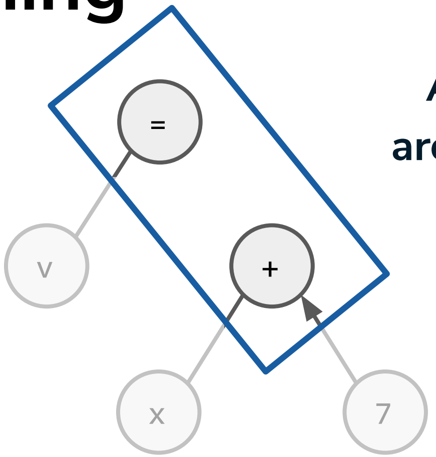
Generative Code Modeling with Graphs. Brockschmidt et. al. ICLR'19

2 **Define Refinement**

Remove Graph Edges

Representation Learning





All decisions are made locally

nodes attributes

$G = \langle V, E, \xi \rangle$

edges

$\alpha:$

$\langle V, E, \xi \rangle \rightarrow \langle V, E' \subseteq E, \xi \rangle$

1 Programs as Graphs

Learning to Represent Programs with Graphs. Allamanis et. al. ICLR'18

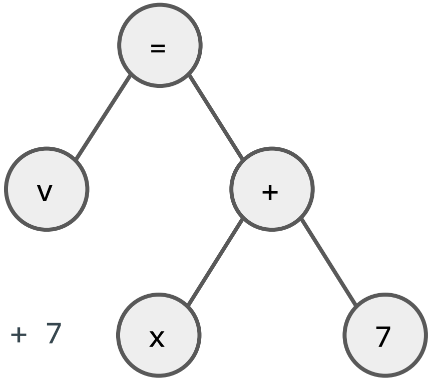
Generative Code Modeling with Graphs. Brockschmidt et. al. ICLR'19

2 Define Refinement

Remove Graph Edges

Representation Learning

`v = x + 7`



nodes attributes

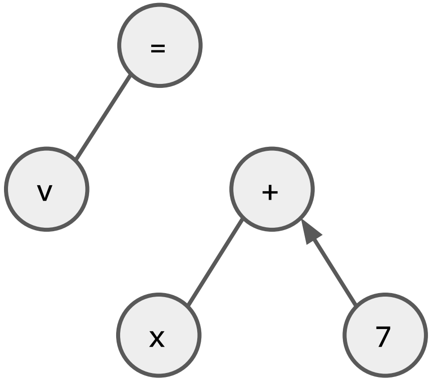
$$G = \langle V, E, \xi \rangle$$

edges

1 **Programs as Graphs**

Learning to Represent Programs with Graphs. Allamanis et. al. ICLR'18

Generative Code Modeling with Graphs. Brockschmidt et. al. ICLR'19



$$\alpha: \langle V, E, \xi \rangle \rightarrow \langle V, E' \subseteq E, \xi \rangle$$

2 **Define Refinement**

Remove Graph Edges

$$\arg\min_{\alpha} \sum_{(\mathbf{x}, \mathbf{y}) \in \mathcal{D}} |\alpha(\mathbf{x})|$$

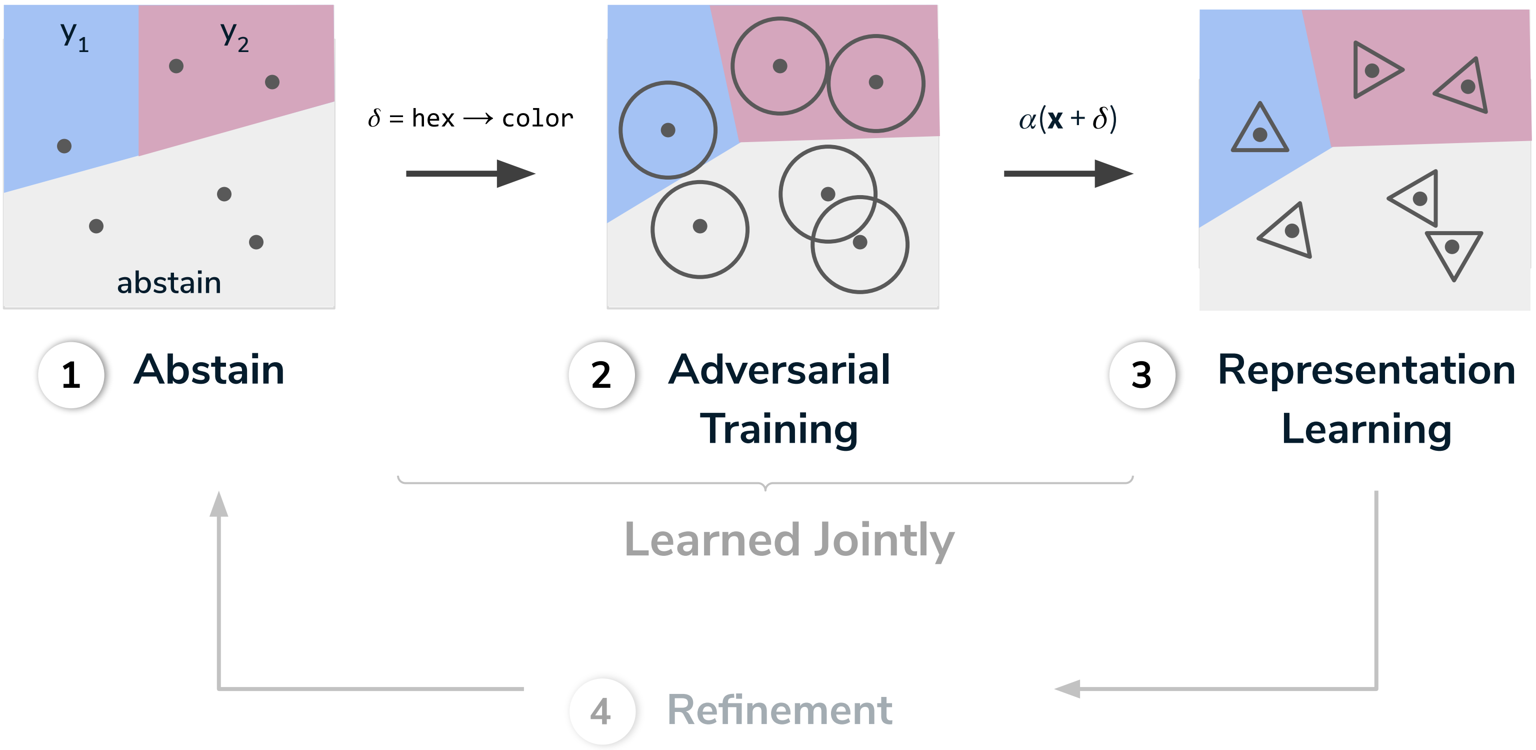
subject to

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

3 **Optimize α**

Minimize Graph Size

Our Work: Three Key Techniques



Evaluation

Task

Type Inference

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

target classes (y)

`string, number, boolean, void`

`()⇒string, ()⇒number, ()⇒boolean, ()⇒void`

`any`

TS TypeScript

JS JavaScript

Models

LSTM

DeepTyper

LSTM + 1 layer GNN + LSTM

DeepTyper: Deep Learning Type Inference. Hellendoorn et. al., FSE'18

Graph Neural Networks

$\text{GNN}_{\text{Transformer}}$ GNN_{GCN} GNN_{GGNN}

more complex type inference

Typilus: Neural Type Hints. Allamanis et. al. PLDI'20

LambdaNet: Probabilistic Type Inference using Graph Neural Networks. Wei et. al. ICLR'20

Our Work: Three Key Techniques

1st Model

2nd Model

3rd Model

4 **Refinement**

Evaluation

$GNN_{Transformer}$

	Accuracy	Robustness
Standard Training	89.3%	54.9%
Adversarial Training	90.3%	54.3%
All Components	88.4%	83.8%

-1%
Accuracy

+29%
Robustness

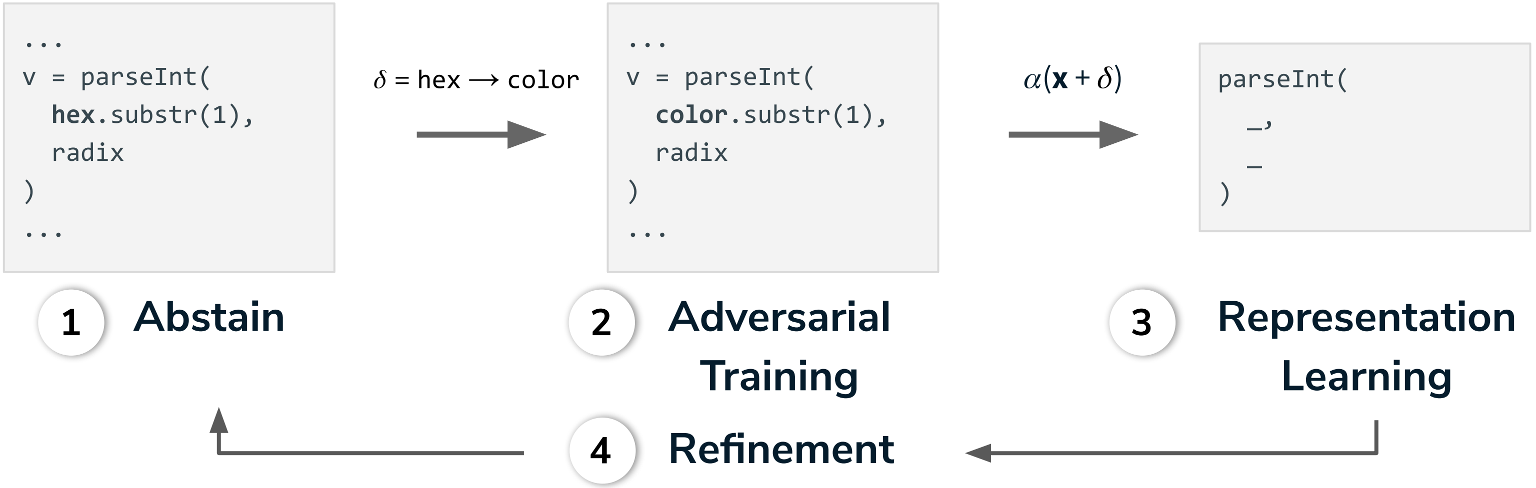
Evaluation

GNN$_{\text{Transformer}}$

Target Accuracy		Accuracy	Robustness	Abstain
	Standard Training	89.3%	54.9%	-
	Adversarial Training	90.3%	54.3%	-
0%	All Components	88.4%	83.8%	-
99%	All Components	99.0%	99.6%	61.3%
100%	All Components	99.9%	99.9%	75.9%

Allows training highly accurate & robust models

Adversarial Robustness for Code



For more experiments and results, please refer to the extended version of our paper

We only scratched the surface, more work in domain of code is needed and is being done, e.g.:

Adversarial Examples for Models of Code. Yefet et. al. ArXiv
Optimization-guided binary diversification to mislead neural networks for malware detection. Sharif et. al. ArXiv
Semantic Robustness of Models of Source Code. Ramakrishnan et. al., ArXiv