# IReEn: Reverse-Engineering of Black-Box Functions via Iterative Neural Program Synthesis

**Hossein Hajipour**[1], Mateusz Malinowski[2], and Mario Fritz[1]

[1]CISPA Helmholtz Center for Information Security
[2]DeepMind

# Introduction

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

## Program synthesis

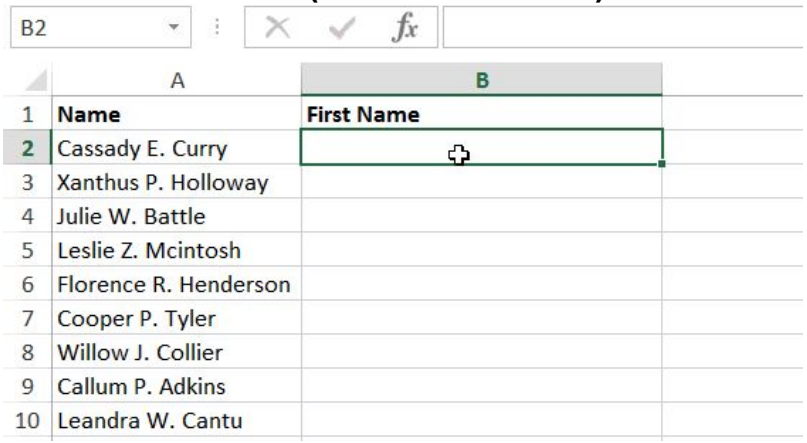
- Automatically generate a computer program that satisfies some specifications, such as a set of input-output examples

# Introduction

## Program synthesis

- Automatically generate a computer program that satisfies some specifications, such as a set of input-output examples

**Flash Fill (Microsoft Excel)**

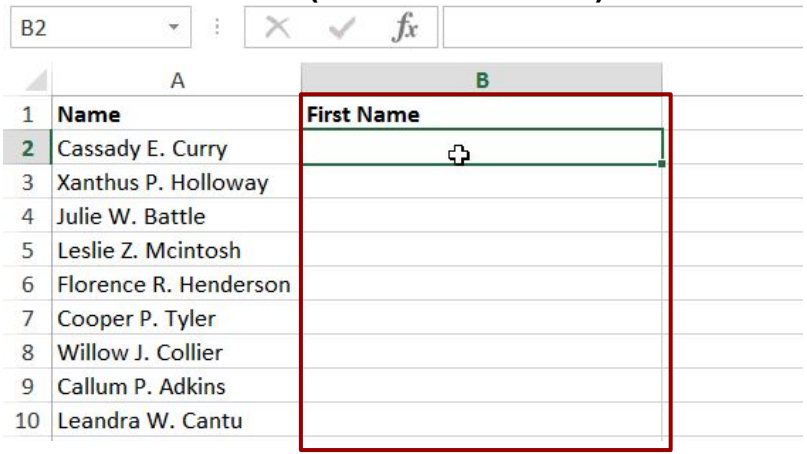


# Introduction

## Program synthesis

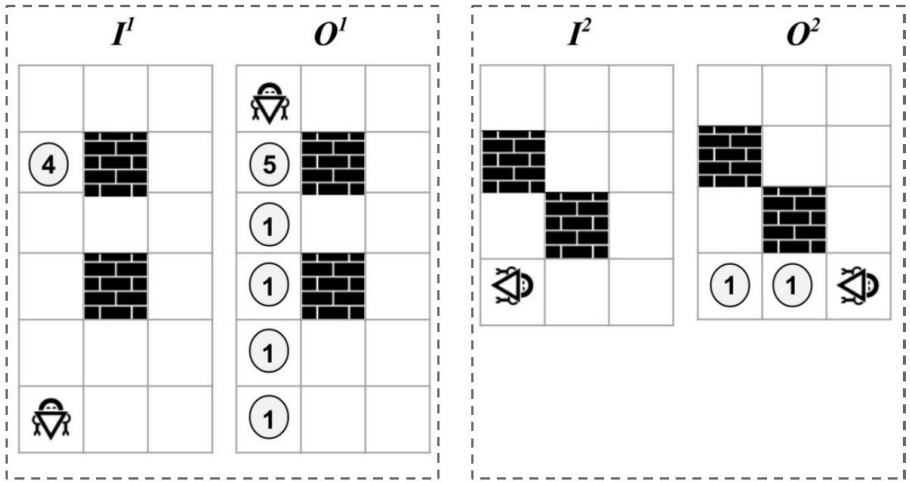
- Automatically generate a computer program that satisfies some specifications, such as a set of input-output examples

**Flash Fill (Microsoft Excel)**



# Introduction

## Program synthesis - Example

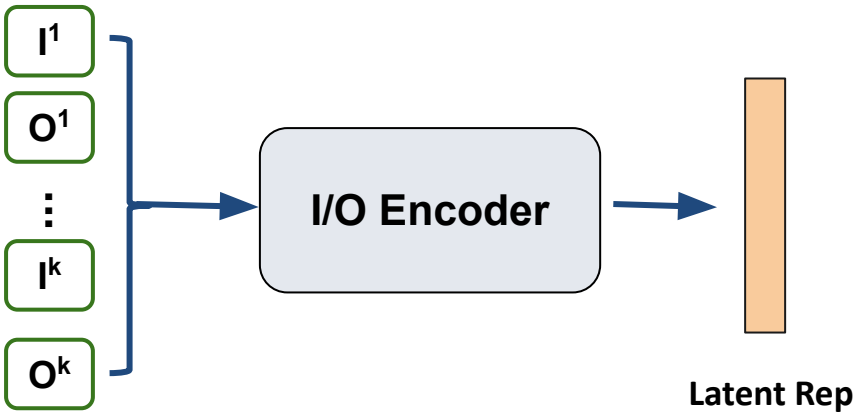



**Underlying program**

```
def run():
  putMarker()
  while(frontIsClear()):
     move()
     putMarker()
```

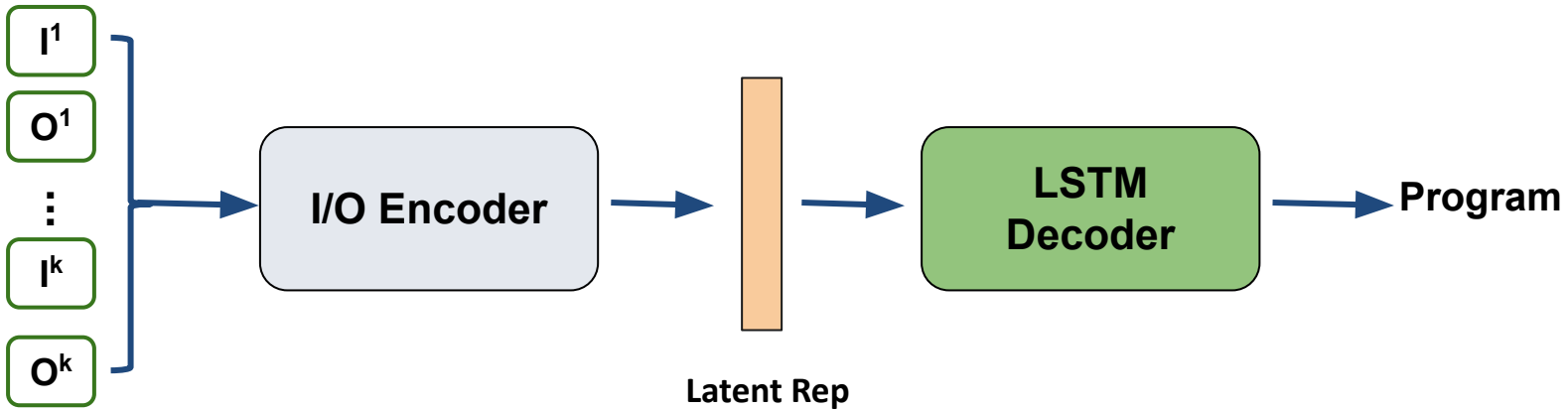
# Introduction

## Neural Program Synthesis (NPS)



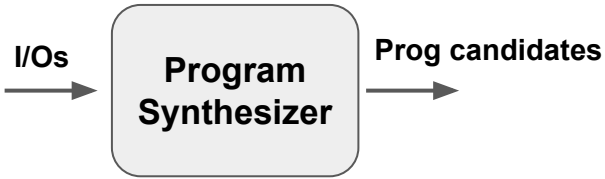
# Introduction

## Neural Program Synthesis (NPS)



# Introduction

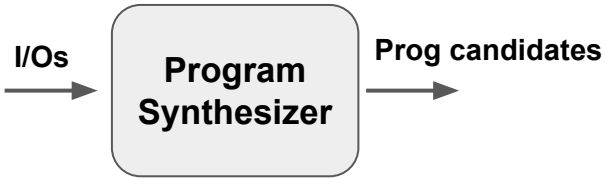
## Program Synthesis in black-box setting



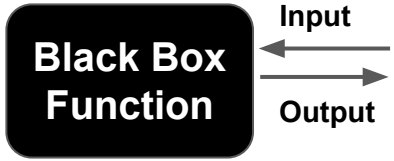

- Program synthesizers use privileged information via biased sampling scheme.
- They assume that the I/Os covers all branches of the desired program.
- We call these crafted specifications **crafted I/Os**

# Introduction

## Program Synthesis in black-box setting



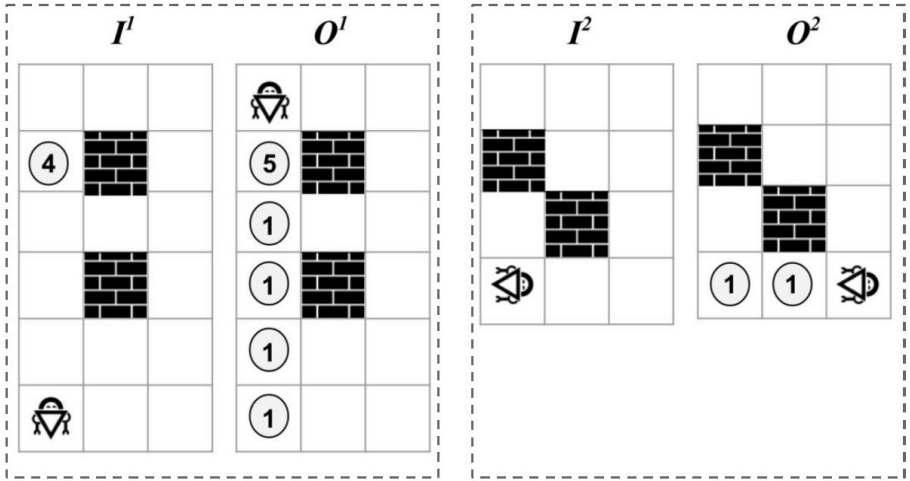

- Program synthesizers use privileged information via biased sampling scheme.
- They assume that the I/Os covers all branches of the desired program.
- We call these crafted specifications **crafted I/Os**




- In the black-box setting we do not have access to the privileged information
- We can only query the black-box function with random **input** to get the corresponding **output**
- We call the obtained I/Os **random I/Os**

# Background

## Karel dataset




**Underlying program**

```
def run():
  putMarker()
  while(frontIsClear()):
     move()
     putMarker()
```

# Background

## Karel dataset

### Karel DSL

**Function:**

```
def run():
  block
```

**Conditional:**

```
if (condition):
  block
if (condition):
  block
else:
  block
```

**Loops:**

```
for i in range(count):
  body
while (condition):
  body
while (not condition):
  body
```

**Actions:**

```
move()
turnLeft()
turnRight()
putMarker()
pickMarker()
```

**Conditions:**

```
frontIsClear()
leftIsClear()
rightIsClear()
markerPresent()
```

# Background

**Karel dataset**

## Karel DSL

**Function:**

```
def run():
  block
```

**Conditional:**

```
if (condition):
  block
if (condition):
  block
else:
  block
```

**Loops:**

```
for i in range(count):
  body
while (condition):
  body
while (not condition):
  body
```

**Actions:**

```
move()
turnLeft()
turnRight()
putMarker()
pickMarker()
```

**Conditions:**

```
frontIsClear()
leftIsClear()
rightIsClear()
markerPresent()
```

$$
\begin{array}{rcl}
\text{Prog } p & := & \texttt{def run() : } s \\
\text{Stmt } s & := & \texttt{while}(b) : s \mid \texttt{repeat}(r) : s \mid s_1 ; s_2 \mid a \\
 & \mid & \texttt{if}(b) : s \mid \texttt{ifelse}(b) : s_1 \texttt{ else } : s_2 \\
\text{Cond } b & := & \texttt{frontIsClear()} \mid \texttt{leftIsClear()} \mid \texttt{rightIsClear()} \\
 & \mid & \texttt{markersPresent()} \mid \texttt{noMarkersPresent()} \mid \texttt{not } b \\
\text{Action a} & := & \texttt{move()} \mid \texttt{turnRight()} \mid \texttt{turnLeft()} \\
 & \mid & \texttt{pickMarker()} \mid \texttt{putMarker()} \\
\text{Cste } r & := & 0 \mid 1 \mid \cdots \mid 19
\end{array}
$$

# Background

## Karel dataset

### Karel DSL

**Function:**

```
def run():
  block
```

**Conditional:**

```
if (condition):
  block
if (condition):
  block
else:
  block
```

**Loops:**

```
for i in range(count):
  body
while (condition):
  body
while (not condition):
  body
```

**Actions:**

```
move()
turnLeft()
turnRight()
putMarker()
pickMarker()
```

**Conditions:**

```
frontIsClear()
leftIsClear()
rightIsClear()
markerPresent()
```

| | | |
|---|---|---|
| Prog $p$ | := | def run() : $s$ |
| Stmt $s$ | := | while($b$) : $s$ \| repeat($r$) : $s$ \| $s_1$; $s_2$ \| $a$ |
| | \| | if($b$) : $s$ \| ifelse($b$) : $s_1$ else : $s_2$ |
| Cond $b$ | := | frontIsClear() \| leftIsClear() \| rightIsClear() |
| | \| | markersPresent() \| noMarkersPresent() \| not $b$ |
| Action a | := | move() \| turnRight() \| turnLeft() |
| | \| | pickMarker() \| putMarker() |
| Cste $r$ | := | 0 \| 1 \| $\cdots$ \| 19 |

# Background

## Karel dataset

### Karel DSL

**Function:**

```
def run():
  block
```

**Conditional:**

```
if (condition):
  block
if (condition):
  block
else:
  block
```

**Loops:**

```
for i in range(count):
  body
while (condition):
  body
while (not condition):
  body
```

**Actions:**

```
move()
turnLeft()
turnRight()
putMarker()
pickMarker()
```

**Conditions:**

```
frontIsClear()
leftIsClear()
rightIsClear()
markerPresent()
```

| | | |
|---|---|---|
| Prog $p$ | := | def run() : $s$ |
| Stmt $s$ | := | while($b$) : $s$ \| repeat($r$) : $s$ \| $s_1$; $s_2$ \| $a$ |
| | \| | if($b$) : $s$ \| ifelse($b$) : $s_1$ else : $s_2$ |
| Cond $b$ | := | frontIsClear() \| leftIsClear() \| rightIsClear() |
| | \| | markersPresent() \| noMarkersPresent() \| not $b$ |
| Action a | := | move() \| turnRight() \| turnLeft() |
| | \| | pickMarker() \| putMarker() |
| Cste $r$ | := | 0 \| 1 \| $\cdots$ \| 19 |

# Background

## Karel dataset

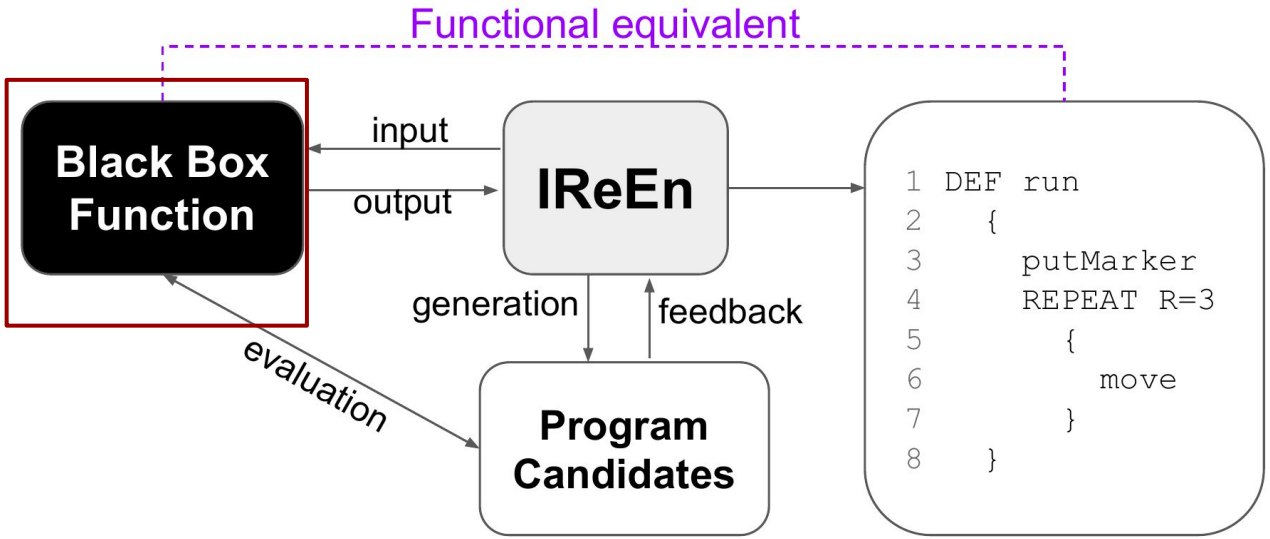
**Input/Output representation:**

- HxWx16
- 2x2x16 to 16x16x16

| |
|---|
| Hero facing North |
| Hero facing South |
| Hero facing West |
| Hero facing East |
| Obstacle |
| Grid boundary |
| 1 marker |
| 2 marker |
| 3 marker |
| 4 marker |
| 5 marker |
| 6 marker |
| 7 marker |
| 8 marker |
| 9 marker |
| 10 marker |

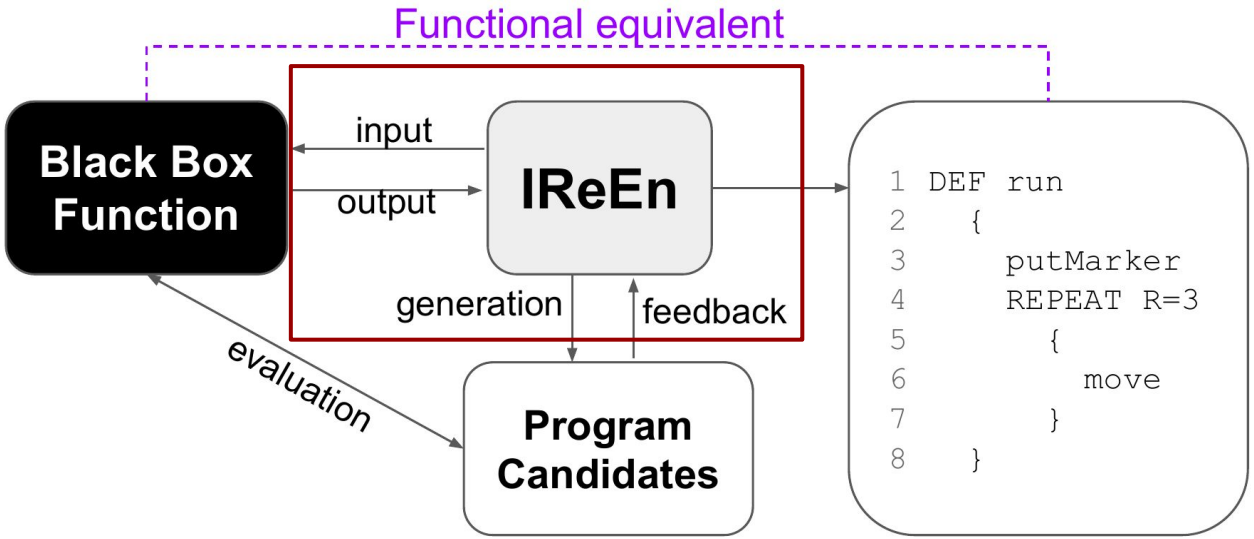
# IReEn

- **Overview**



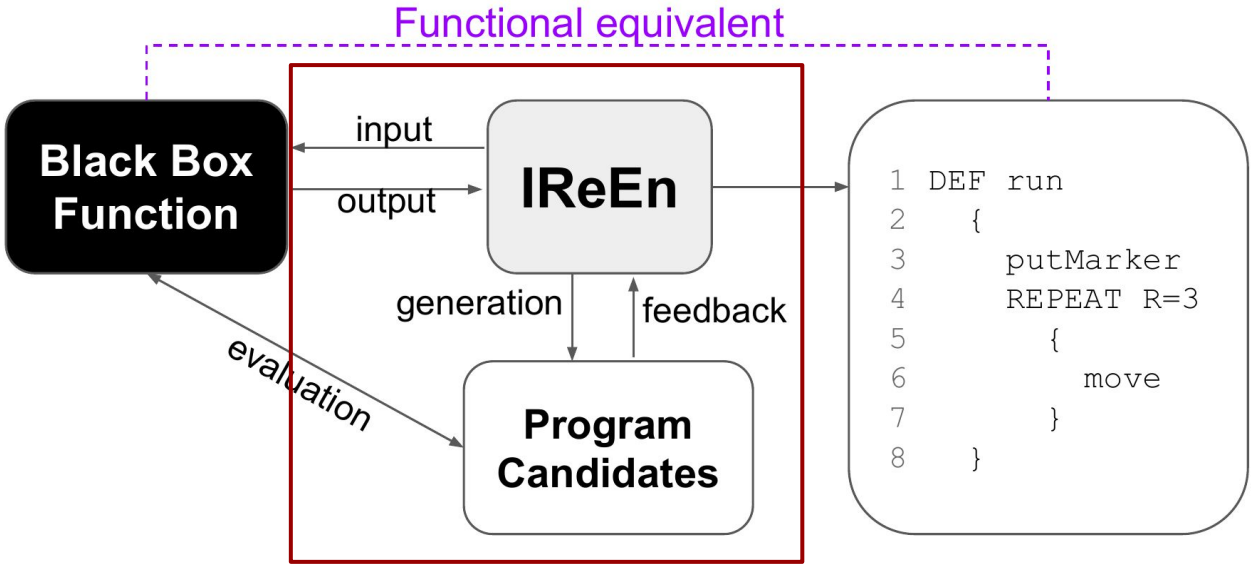
# IReEn

- **Overview**



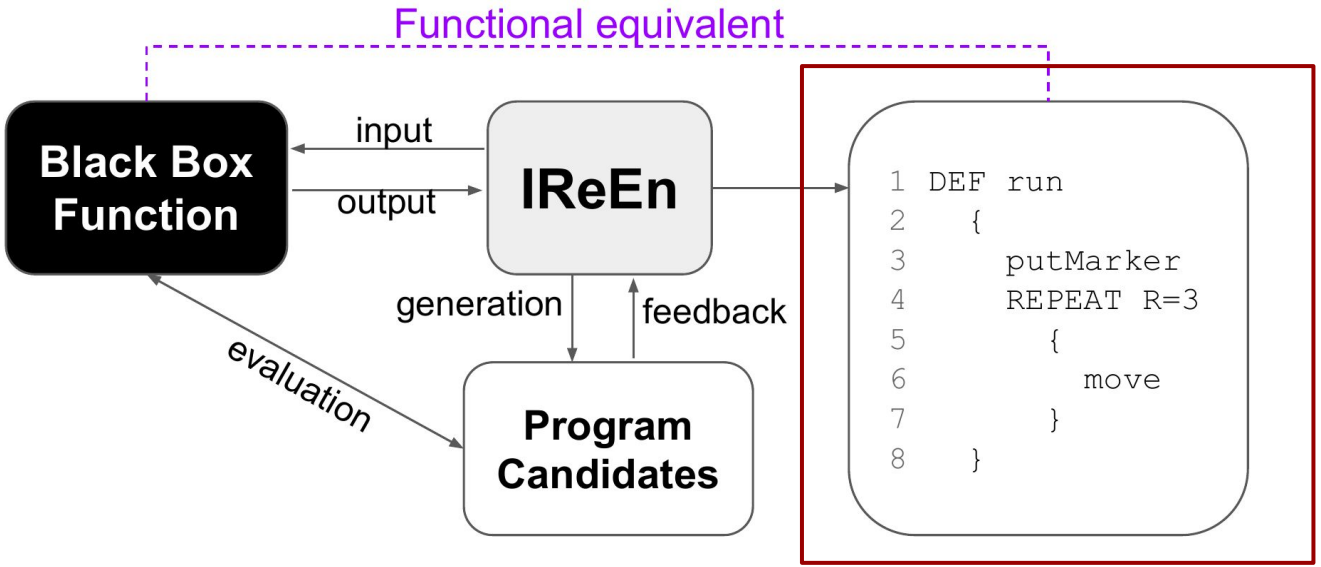
# IReEn

- **Overview**



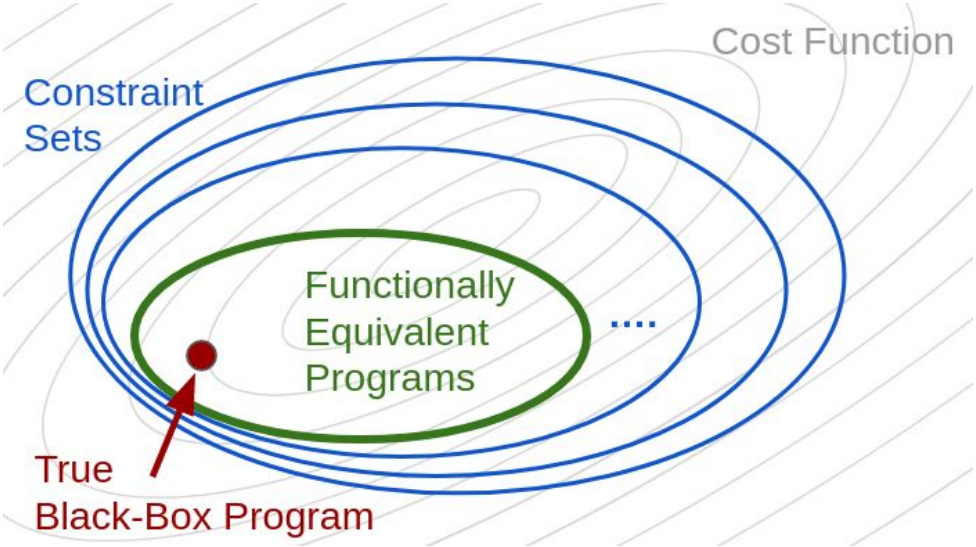
# IReEn

- **Overview**



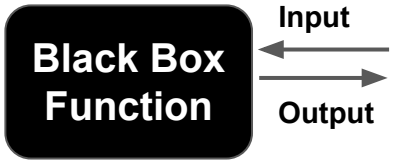
# IReEn

- **Overview**
  - **Iterative program synthesis**



# IReEn

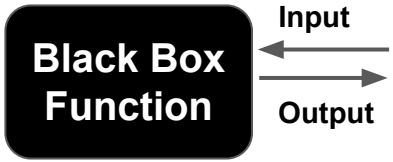
- **Iterative Neural Program Synthesis**



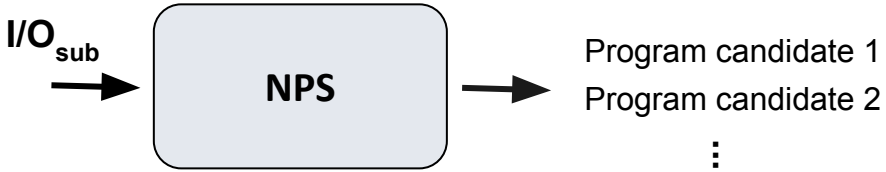

- **I/O**
- **$I/O_{sub} \subseteq I/O$**

# IReEn

- **Iterative Neural Program Synthesis**

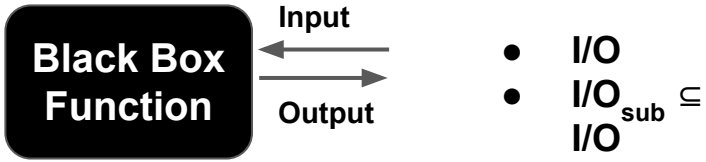



- **I/O**
- **$I/O_{sub} \subseteq I/O$**

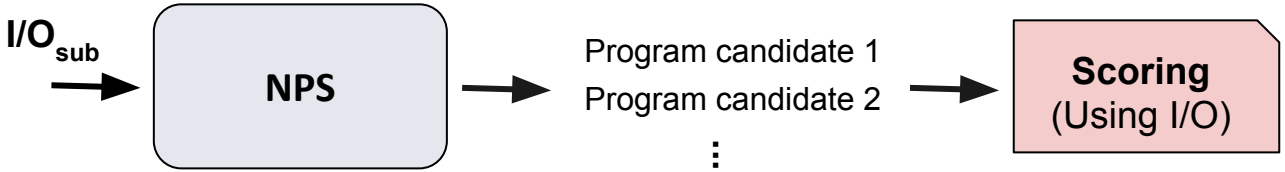


# IReEn

- **Iterative Neural Program Synthesis**

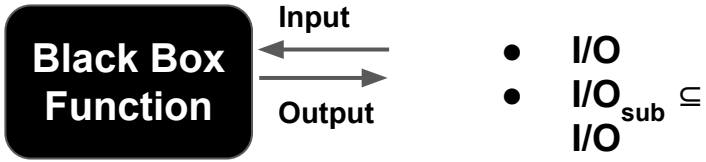



- **I/O**
- **$I/O_{sub} \subseteq I/O$**

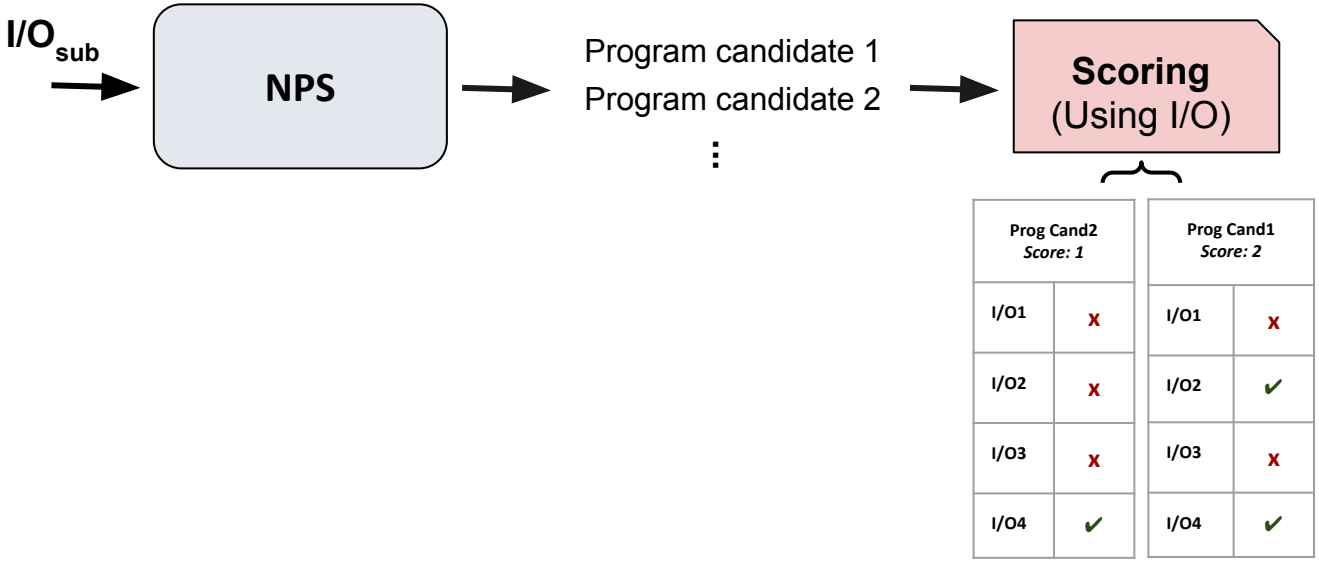


# IReEn

- **Iterative Neural Program Synthesis**

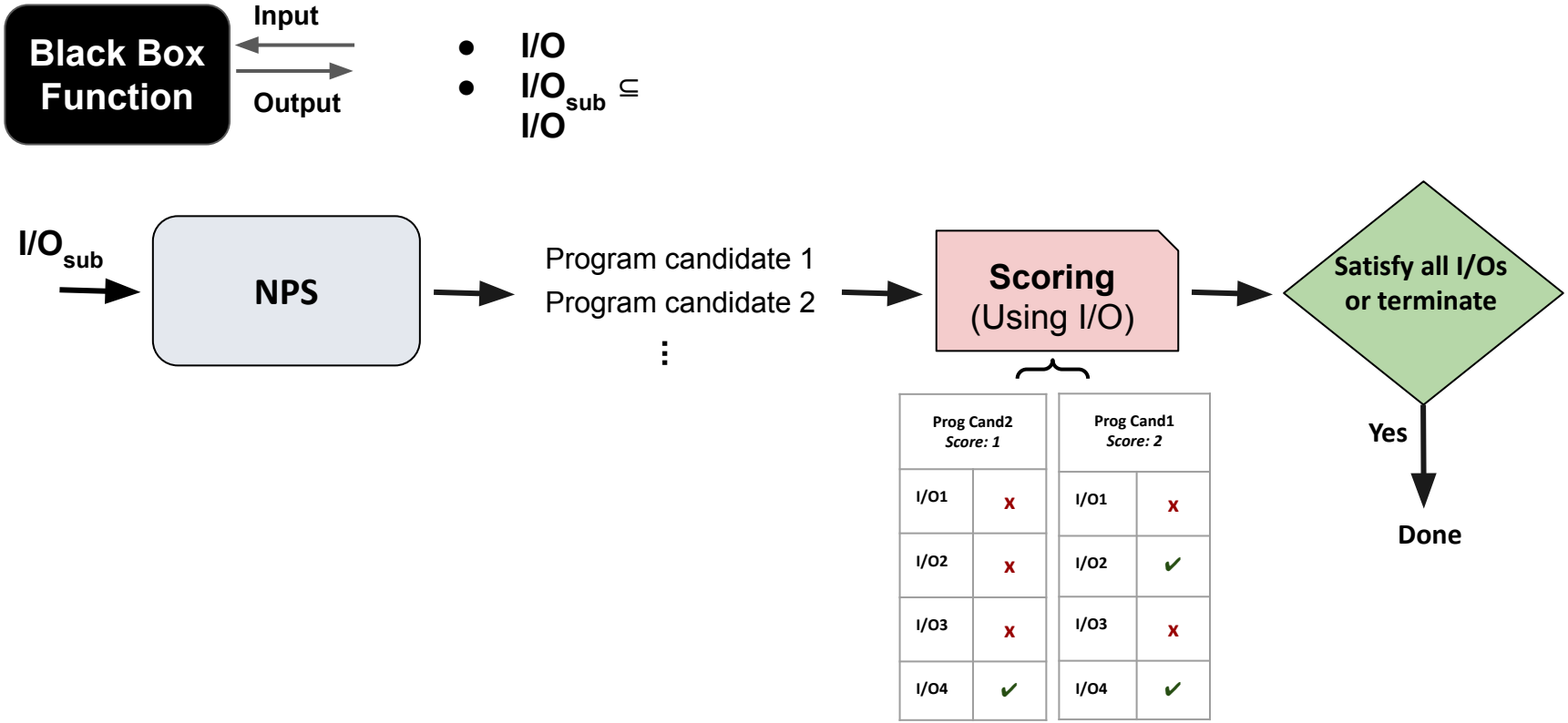



- I/O
- $I/O_{sub} \subseteq I/O$



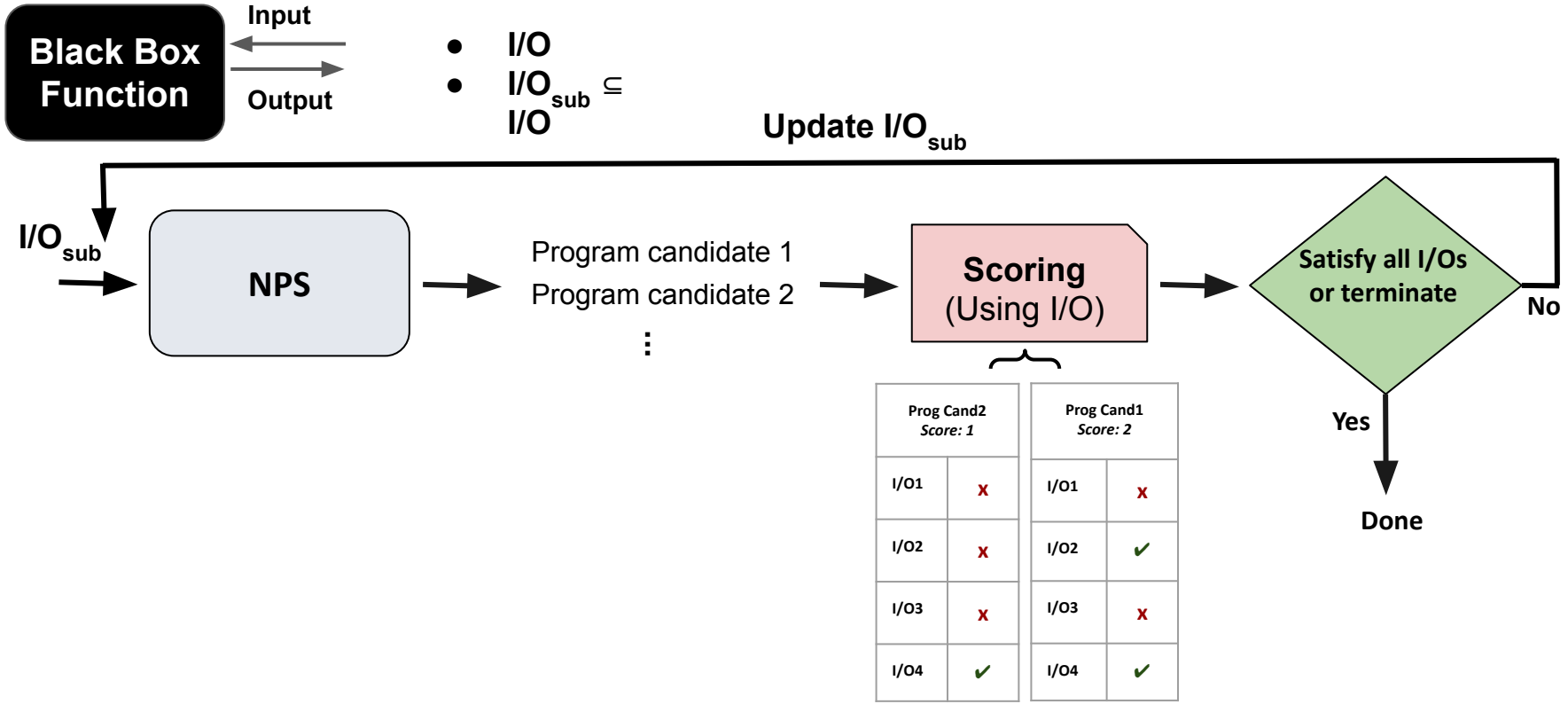
# IReEn

- **Iterative Neural Program Synthesis**



# IReEn

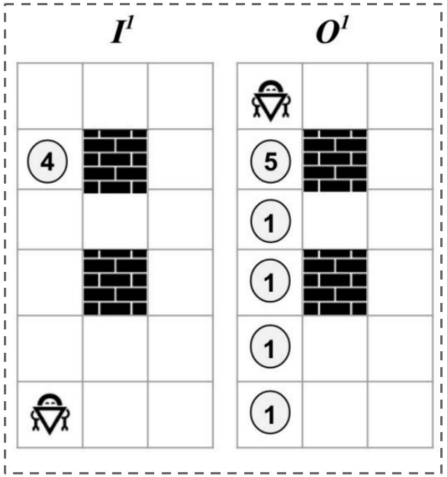
- **Iterative Neural Program Synthesis**



# Experiments

## Evaluation Setup

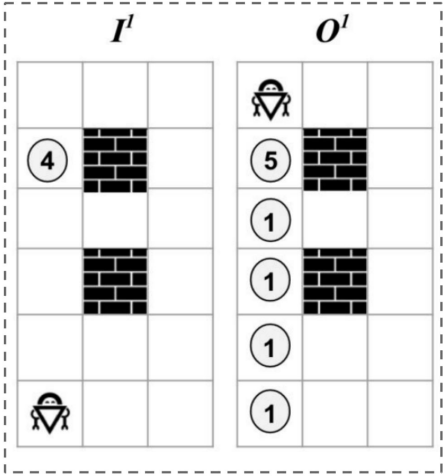
- **Karel dataset**
  - The Karel benchmark is one of the largest publicly available program synthesis dataset.
  - 1,116,854 samples for training, 2,500 validation set, and 2,500 test examples.



# Experiments

## Evaluation Setup

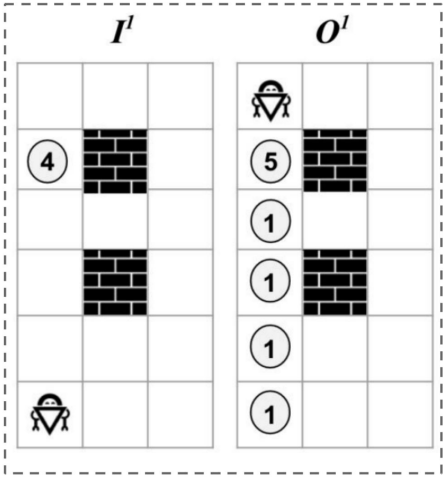
- **Karel dataset**
  - The Karel benchmark is one of the largest publicly available program synthesis dataset.
  - 1,116,854 samples for training, 2,500 validation set, and 2,500 test examples.
  - Each sample is provided with a ground truth program.
  - 5 input-output pairs as the specification, and an additional one as the held-out test example (Crafted I/Os).



# Experiments

## Evaluation Setup

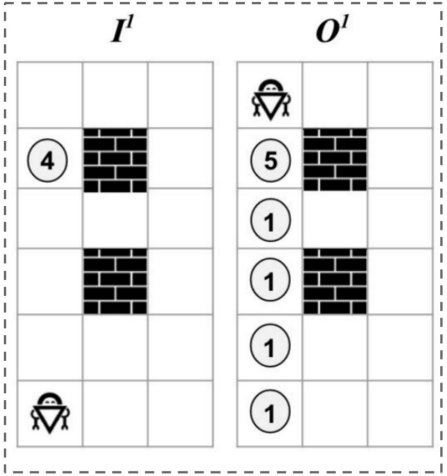
- **Karel dataset**
  - The Karel benchmark is one of the largest publicly available program synthesis dataset.
  - 1,116,854 samples for training, 2,500 validation set, and 2,500 test examples.
  - Each sample is provided with a ground truth program.
  - 5 input-output pairs as the specification, and an additional one as the held-out test example (Crafted I/Os).
  - In the black-box setting we can query the black-box function using random input to get the corresponding output (50 I/Os)



# Experiments

## Evaluation Setup

- **Karel dataset**
  - The Karel benchmark is one of the largest publicly available program synthesis dataset.
  - 1,116,854 samples for training, 2,500 validation set, and 2,500 test examples.
  - Each sample is provided with a ground truth program.
  - 5 input-output pairs as the specification, and an additional one as the held-out test example (Crafted I/Os).
  - In the black-box setting we can query the black-box function using random input to get the corresponding output (50 I/Os)
- **During inference**
  - We use beam search with beam width 64



# Experiments

## Functional Equivalence Metric

- **Generalization Metric**
  - A predicted program is a generalization of the target if it satisfies the given I/Os and one held-out example.
- **Exact Match Metric**
  - A predicted program is an exact match of the target if it is the same as the target program in terms of tokens.
- **Functional Equivalence Metric**
  - We consider a predicted program equivalent to the target program if it covers a large set of unseen I/Os (We use 100 I/Os).

# Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

## Results

| Models | Generalization | | Functional | | Exact Match | |
|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 57.12% | 71.48% | 49.36% | 63.72% | 34.96% | 40.92% |
| Random I/Os + FT | 64.72% | 77.64% | 55.64% | 70.12% | 39.44% | 45.4% |
| Random I/Os + IReEn | 76.20% | 85.28% | 61.64% | 73.24% | 40.95% | 44.99% |
| Random I/Os + FT + IReEn | **78.96%** | **88.39%** | **65.55%** | **78.08%** | **44.51%** | **48.11%** |
| Crafted I/Os ([4]) | 73.12% | 86.28% | 55.04% | 68.72% | 40.08% | 43.08% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

## Results

| Models | Generalization | | Functional | | Exact Match | |
|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 57.12% | 71.48% | 49.36% | 63.72% | 34.96% | 40.92% |
| Random I/Os + FT | 64.72% | 77.64% | 55.64% | 70.12% | 39.44% | 45.4% |
| Random I/Os + IReEn | 76.20% | 85.28% | 61.64% | 73.24% | 40.95% | 44.99% |
| Random I/Os + FT + IReEn | **78.96%** | **88.39%** | **65.55%** | **78.08%** | **44.51%** | **48.11%** |
| Crafted I/Os ([4]) | 73.12% | 86.28% | 55.04% | 68.72% | 40.08% | 43.08% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

## Results

| Models | Generalization | | Functional | | Exact Match | |
|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 57.12% | 71.48% | 49.36% | 63.72% | 34.96% | 40.92% |
| Random I/Os + FT | 64.72% | 77.64% | 55.64% | 70.12% | 39.44% | 45.4% |
| Random I/Os + IReEn | 76.20% | 85.28% | 61.64% | 73.24% | 40.95% | 44.99% |
| Random I/Os + FT + IReEn | **78.96%** | **88.39%** | **65.55%** | **78.08%** | **44.51%** | **48.11%** |
| Crafted I/Os ([4]) | 73.12% | 86.28% | 55.04% | 68.72% | 40.08% | 43.08% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

## Results

- **Functional equivalence** results of our approaches in synthesizing programs with different complexity

| Models | Action | | Repeat | | While | | If | | Mix | |
|---|---|---|---|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 95.59% | 99.69% | 85.52% | 91.44% | 26.98% | 61.58% | 48.88% | 72.69% | 10.69% | 27.12% |
| Random I/Os + FT | 99.39% | 99.76% | 90.72% | 96.38% | 56.50% | 82.22% | 52.06% | 77.46% | 14.33% | 32.19% |
| Random I/Os + IReEn | **99.84%** | 99.84% | **96.38%** | 97.36% | 60.95% | 84.76% | 81.26% | 89.84% | 27.67% | 49.94% |
| Random I/Os + FT + IReEn | **99.84%** | **100%** | 95.39% | **99.64%** | **81.58%** | **93.33%** | **81.52%** | **92.06%** | **32.08%** | **56.22%** |
| Crafted I/Os | 99.08% | 100.0% | 91.11% | 96.71% | 54.28% | 84.12% | 49.20% | 79.68% | 14.88% | 33.84% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

## Results

- **Functional equivalence** results of our approaches in synthesizing programs with different complexity

| Models | Action | | Repeat | | While | | If | | Mix | |
|---|---|---|---|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 95.59% | 99.69% | 85.52% | 91.44% | 26.98% | 61.58% | 48.88% | 72.69% | 10.69% | 27.12% |
| Random I/Os + FT | 99.39% | 99.76% | 90.72% | 96.38% | 56.50% | 82.22% | 52.06% | 77.46% | 14.33% | 32.19% |
| Random I/Os + IReEn | **99.84%** | 99.84% | **96.38%** | 97.36% | 60.95% | 84.76% | 81.26% | 89.84% | 27.67% | 49.94% |
| Random I/Os + FT + IReEn | **99.84%** | **100%** | 95.39% | **99.64%** | **81.58%** | **93.33%** | **81.52%** | **92.06%** | **32.08%** | **56.22%** |
| Crafted I/Os | 99.08% | 100.0% | 91.11% | 96.71% | 54.28% | 84.12% | 49.20% | 79.68% | 14.88% | 33.84% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

## Results

- **Functional equivalence** results of our approaches in synthesizing programs with different complexity

| Models | Action | | Repeat | | While | | If | | Mix | |
|---|---|---|---|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 95.59% | 99.69% | 85.52% | 91.44% | 26.98% | 61.58% | 48.88% | 72.69% | 10.69% | 27.12% |
| Random I/Os + FT | 99.39% | 99.76% | 90.72% | 96.38% | 56.50% | 82.22% | 52.06% | 77.46% | 14.33% | 32.19% |
| Random I/Os + IReEn | **99.84%** | 99.84% | **96.38%** | 97.36% | 60.95% | 84.76% | 81.26% | 89.84% | 27.67% | 49.94% |
| Random I/Os + FT + IReEn | **99.84%** | **100%** | 95.39% | **99.64%** | **81.58%** | **93.33%** | **81.52%** | **92.06%** | **32.08%** | **56.22%** |
| Crafted I/Os | 99.08% | 100.0% | 91.11% | 96.71% | 54.28% | 84.12% | 49.20% | 79.68% | 14.88% | 33.84% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

## Results

- **Functional equivalence** results of our approaches in synthesizing programs with different complexity

| Models | Action | | Repeat | | While | | If | | Mix | |
|---|---|---|---|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 95.59% | 99.69% | 85.52% | 91.44% | 26.98% | 61.58% | 48.88% | 72.69% | 10.69% | 27.12% |
| Random I/Os + FT | 99.39% | 99.76% | 90.72% | 96.38% | 56.50% | 82.22% | 52.06% | 77.46% | 14.33% | 32.19% |
| Random I/Os + IReEn | **99.84%** | 99.84% | **96.38%** | 97.36% | 60.95% | 84.76% | 81.26% | 89.84% | 27.67% | 49.94% |
| Random I/Os + FT + IReEn | **99.84%** | **100%** | 95.39% | **99.64%** | **81.58%** | **93.33%** | **81.52%** | **92.06%** | **32.08%** | **56.22%** |
| Crafted I/Os | 99.08% | 100.0% | 91.11% | 96.71% | 54.28% | 84.12% | 49.20% | 79.68% | 14.88% | 33.84% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

## Results

- **Functional equivalence** results of our approaches in synthesizing programs with different complexity

| Models | Action | | Repeat | | While | | If | | Mix | |
|---|---|---|---|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 95.59% | 99.69% | 85.52% | 91.44% | 26.98% | 61.58% | 48.88% | 72.69% | 10.69% | 27.12% |
| Random I/Os + FT | 99.39% | 99.76% | 90.72% | 96.38% | 56.50% | 82.22% | 52.06% | 77.46% | 14.33% | 32.19% |
| Random I/Os + IReEn | **99.84%** | 99.84% | **96.38%** | 97.36% | 60.95% | 84.76% | 81.26% | 89.84% | 27.67% | 49.94% |
| Random I/Os + FT + IReEn | **99.84%** | **100%** | 95.39% | **99.64%** | **81.58%** | **93.33%** | **81.52%** | **92.06%** | **32.08%** | **56.22%** |
| Crafted I/Os | 99.08% | 100.0% | 91.11% | 96.71% | 54.28% | 84.12% | 49.20% | 79.68% | 14.88% | 33.84% |

- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

# Experiments

## Results

- **Functional equivalence** results of our approaches in synthesizing programs with different complexity

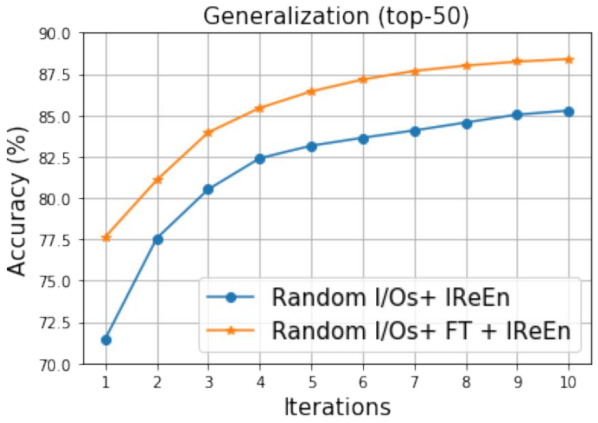
| Models | Action | | Repeat | | While | | If | | Mix | |
|---|---|---|---|---|---|---|---|---|---|---|
| | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 | top-1 | top-50 |
| Random I/Os | 95.59% | 99.69% | 85.52% | 91.44% | 26.98% | 61.58% | 48.88% | 72.69% | 10.69% | 27.12% |
| Random I/Os + FT | 99.39% | 99.76% | 90.72% | 96.38% | 56.50% | 82.22% | 52.06% | 77.46% | 14.33% | 32.19% |
| Random I/Os + IReEn | **99.84%** | 99.84% | **96.38%** | 97.36% | 60.95% | 84.76% | 81.26% | 89.84% | 27.67% | 49.94% |
| Random I/Os + FT + IReEn | **99.84%** | **100%** | 95.39% | **99.64%** | **81.58%** | **93.33%** | **81.52%** | **92.06%** | **32.08%** | **56.22%** |
| Crafted I/Os | 99.08% | 100.0% | 91.11% | 96.71% | 54.28% | 84.12% | 49.20% | 79.68% | 14.88% | 33.84% |

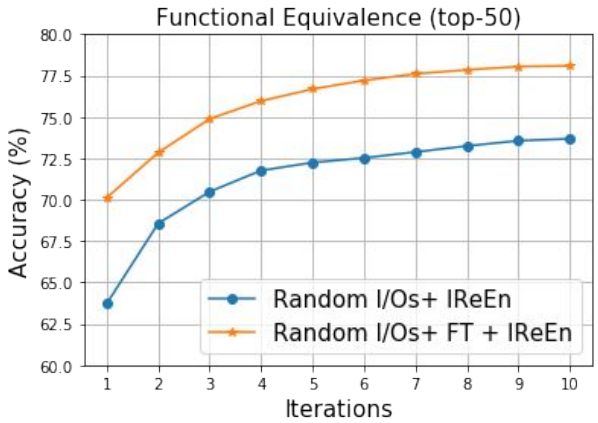
- **Random I/Os** mean that we use randomly obtained I/Os in the black-box setting.
- **FT** refers to fine-tuned model using random I/Os.
- **IReEn** denote to our iterative approach.
- **Crafted I/Os** refers to the results of the model which uses privileged information.

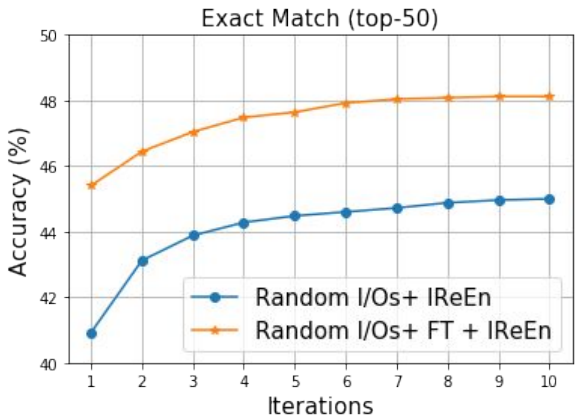
# Experiments

## Results

- **Effectiveness of iterative refinement**



# Experiments

## Results

- **Examples of Functional equivalent programs**

**Target Black-Box Program**

```
1 def run():
2   ifelse(not(rightIsClear())):
3       turnLeft()
4       move()
5       pickMarker()
6   else:
7       turnRight()
8   move()
9   move()
10  pickMarker()
11  pickMarker()
```

# Experiments

## Results

- **Examples of Functional equivalent programs**

**Target Black-Box Program**

```
1  def run():
2    ifelse(not(rightIsClear())):
3       turnLeft()
4       move()
5       pickMarker()
6    else:
7       turnRight()
8    move()
9    move()
10   pickMarker()
11   pickMarker()
```

**Output Program**

```
1  def run():
2    ifelse(rightIsClear()):
3       turnRight()
4    else:
5       turnLeft()
6       move()
7       pickMarker()
8    move()
9    move()
10   pickMarker()
11   pickMarker()
```

# Experiments

## Results

- **Examples of Functional equivalent programs**

**Target Black-Box Program**

```
1  def run():
2     ifelse(not(rightIsClear())):
3        turnLeft()
4        move()
5        pickMarker()
6     else:
7        turnRight()
8     move()
9     move()
10    pickMarker()
11    pickMarker()
```

**Output Program**

```
1  def run():
2     ifelse(rightIsClear()):
3        turnRight()
4     else:
5        turnLeft()
6        move()
7        pickMarker()
8     move()
9     move()
10    pickMarker()
11    pickMarker()
```

# Experiments

## Results

- **Examples of Functional equivalent programs**

**Target Black-Box Program**

```
1 def run():
2   repeat(5):
3     turnRight()
4   putMarker()
5   putMarker()
6   move()
```

# Experiments

## Results

- **Examples of Functional equivalent programs**

**Target Black-Box Program**

```
1 def run():
2   repeat(5):
3     turnRight()
4   putMarker()
5   putMarker()
6   move()
```

**Output Program**

```
1 def run():
2   putMarker()
3   putMarker()
4   turnRight()
5   move()
```

# Experiments

## Results

- **Examples of Functional equivalent programs**

**Target Black-Box Program**

```
1 def run():
2   repeat(5):
3     turnRight()
4   putMarker()
5   putMarker()
6   move()
```

**Output Program**

```
1 def run():
2   putMarker()
3   putMarker()
4   turnRight()
5   move()
```

# Conclusion

- We propose an iterative neural program synthesis scheme to reverse-engineer the black-box functions, and represent them in a high-level program.
- We proposed a functional equivalence metric in order to quantify progress on this challenging task.
- We evaluate our approach on Karel dataset, where our approach successfully revealed the underlying programs of 78% of the black-box programs.

# Next steps...

Thank you!