ReCode: Robustness Evaluation of Code Generation Models

Shiqi Wang*, *Zheng Li*, Haifeng Qian, Chenghao Yang, Zijian Wang, Mingyue Shang, Varun Kumar, Samson Tan, Baishakhi Ray, Parminder Bhatia, Ramesh Nallapati, Murali Krishna Ramanathan, Dan Roth, Bing Xiang*

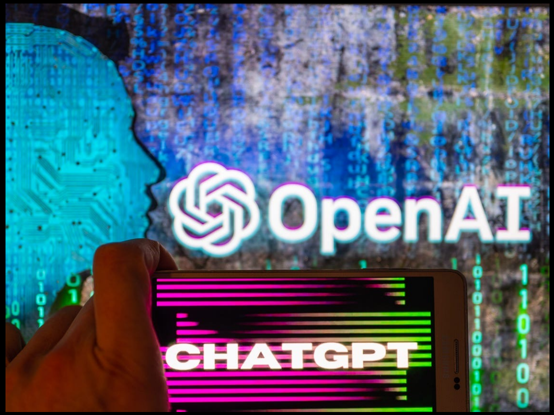
AWS AI Lab*, AWS AI Research & Education,*

Cornell university, University of Chicago

Code Generation Model

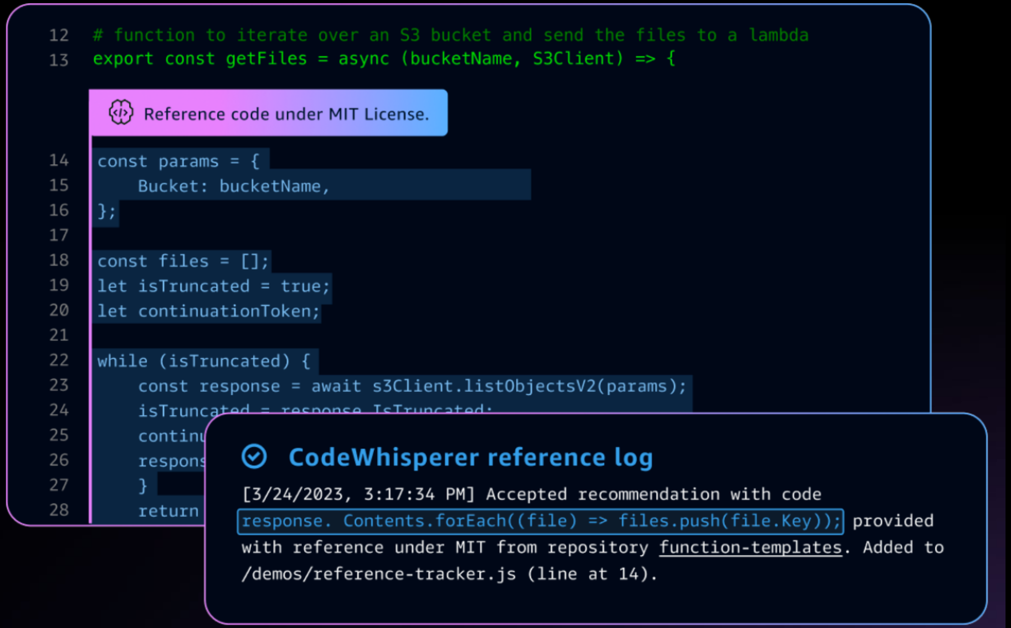
Code generation has emerged as an important AI application

- Offer real-life help to software engineers and enhance their productivity
- Popular public models: CodeGen, InCoder, GPT-J
- Popular tools: CodeWhisperer, Copilot, ChatGPT



Code Generation Demo (CodeWhisperer)

Efficient Code Snippet Generation



…Don't worry about copyright issues!

Robustness?

Why robustness for code?

- Robustness of the code generation model is commonly overlooked
- Simple perturbations will cause mistakes by code generation models
- Significantly affect user experience

Robustness?

Why robustness for code?

- Robustness of the code generation model is commonly overlooked
- Simple perturbations will cause mistakes by code generation models
- Significantly affect user experience

Original docstring

Original completion

```python
def test_distinct(data):
    """
    Write a python function to determine whether all the
    numbers are different from each other are not.
    >>> test_distinct([1,5,7,9])
    True
    >>> test_distinct([2,4,5,5,7,9])
    False
    >>> test_distinct([1,2,3])
    True
    """
    return len(set(data)) == len(data)
```

Robustness?

Why robustness for code?

- Robustness of the code generation model is commonly overlooked
- Simple perturbations will cause mistakes by code generation models
- Significantly affect user experience

Original docstring / **Original completion**

```python
def test_distinct(data):
    """
    Write a python function to determine whether all the
    numbers are different from each other are not.
    >>> test_distinct([1,5,7,9])
    True
    >>> test_distinct([2,4,5,5,7,9])
    False
    >>> test_distinct([1,2,3])
    True
    """
    return len(set(data)) == len(data)
```

Perturbed docstring / **New completion**

```python
def test_distinct(data):
    """
    Write a Python function to see if all
    numbers differ from each other.
    >>> test_distinct([1,5,7,9])
    True
    >>> test_distinct([2,4,5,5,7,9])
    False
    >>> test_distinct([1,2,3])
    True
    """
    return len(set(data)) != len(data)
```

Paraphrasing docstring will cause mistakes by InCoder-6B

Robustness?

Why robustness for code?

- Robustness of the code generation model is commonly overlooked
- Simple perturbations will cause mistakes by code generation models
- Significantly affect user experience

Original Function name

```python
def remove_lowercase(str1):
    """
    Write a function to remove lowercase
    substrings from a given string.
    >>> remove_lowercase("PYTHon")
    ('PYTH')
    >>> remove_lowercase("FInD")
    ('FID')
    >>> remove_lowercase("STRinG")
    ('STRG')
    """
    return "".join([i for i in str1 if i.isupper()])
```

Original completion

Perturbed function name

```python
def removeLowercase(str1):
    """
    Write a function to remove lowercase
    substrings from a given string.
    >>> removeLowercase("PYTHon")
    ('PYTH')
    >>> removeLowercase("FInD")
    ('FID')
    >>> removeLowercase("STRinG")
    ('STRG')
    """
    str2 = str1.lower()
    return str2
```

New completion

Changing function name style cause mistakes by CodeGen-16B-mono

ReCode

ReCode: the first comprehensive **R**obustness **E**valuation framework for **Code**.

4 categories, 30 customized perturbations

- Docstrings
- Function names
- Code syntax
- Code format

Semantic Preserving!

ReCode: Transformations

ReCode: the first comprehensive **R**obustness **E**valuation framework for **Code**.

4 categories, 30 customized perturbations

- **Docstrings**
- Function names
- Code syntax
- Code format

Semantic Preserving!

Perturbations	MBPP Docstrings
Nominal	Write a function to find all words which are at least 4 characters long in a string by using regex.
BackTranslation	Write a function to find all words **in a string at least 4 characters long** using regex.
ButterFingers	Wri**h**e a function to find all words which are a**r** leas**v** 4 characters long in a string by using regex.
ChangeCharCase	Wri**T**e a f**U**ncti**O**n to find All wo**R**ds whic**H** are at le**A**st 4 Cha**R**acter**S** Lon**G** in a string by u**SI**ng re**G**ex.
EnglishInflectionalVariation	**Writes** a **functions** to **found** all **word** which **was** at least 4 **character** long in a string by use regex.
SwapCharacters	**rW**ite a function to find all words which are at **el**ast 4 ch**ra**acters long in a string by **su**ing regex.
SynonymInsertion	Write a function to find **discover** all words which are at least 4 characters long in a string by using regex.
SynonymSubstitution	Write a function to find all words which **equal** at least 4 character long in a **chain** by using regex.
TenseTransformationPast	Write a function to find all words which **was** at least 4 characters long in a string by using regex.
TenseTransformationFuture	Write a function to find all words which **will be** at least 4 characters long in a string by using regex.
Whitespace	Write a function to find all words **w** **hichar**e at least 4 characters long in a string by using regex.

Table 1: Illustrations for docstring perturbations on a MBPP sample.

ReCode: Transformations

ReCode: the first comprehensive **R**obustness **E**valuation framework for **Code**.

4 categories, 30 customized perturbations

- Docstrings
- **Function names**
- Code syntax
- Code format

Semantic Preserving!

Perturbations on Function Names	MBPP
Nominal	find_char_long
CamelCase	**findCharLong**
ButterFingers	fin**f**_char_long
SwapCharacters	find_**cahr**_long
ChangeCharCase	fin**D**_cha**R**_long
InflectionalVariation	**found_chars_long**
SynonymSubstition	**discover**_char_long

ReCode: Transformations

ReCode: the first comprehensive **R**obustness **E**valuation framework for **Code**.

4 categories, 30 customized perturbations

- Docstrings
- Function names
- **Code syntax**
- Code format

Semantic Preserving!

```python
def remove_Occ(s, ch):
    """
    Write a python function to remove
     first and last occurrence of a
     given character from the string.
    >>> remove_Occ("hello","l")
    "heo"
    >>> remove_Occ("abcda","a")
    "bcd"
    >>> remove_Occ("PHP","P")
    "H"
    """
    for i in range(len(s)):
        if s[i] == ch:
            s = s[0:i] + s[i + 1 :]
            break
```

MBPP baseline partial code

```python
def remove_Occ(s, ch):
    # [same doc string]
    i = 0
    while i < len(s):
        if s[i] == ch:
            s = s[0:i] + s[i + 1 :]
            break
        i = i + 1
```

For-while switch

```python
def remove_Occ(lines, ch):
    # [same doc string]
    for i in range(len(lines)):
        if lines[i] == ch:
            lines = lines[0:i] + lines[i + 1 :]
            break
```

CodeBERT variable rename

aws

ReCode: Transformations

ReCode: the first comprehensive **R**obustness **E**valuation framework for **Code**.

4 categories, 30 customized perturbations

- Docstrings
- Function names
- Code syntax
- **Code format**

Semantic Preserving!

```python
def remove_Occ(s, ch):
    """
    Write a python function to remove
     first and last occurrence of a
     given character from the string.
    >>> remove_Occ("hello","l")
    "heo"
    >>> remove_Occ("abcda","a")
    "bcd"
    >>> remove_Occ("PHP","P")
    "H"
    """
    for i in range(len(s)):
        if s[i] == ch:
            s = s[0:i] + s[i + 1 :]
            break
```

MBPP baseline partial code

```python
def remove_Occ(s, ch):
    # Write a python function to remove
    #  first and last occurrence of a
    #  given character from the string.
    # >>> remove_Occ("hello","l")
    # "heo"
    # >>> remove_Occ("abcda","a")
    # "bcd"
    # >>> remove_Occ("PHP","P")
    # "H"
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
```

Docstring to comments

```python
def remove_Occ(s, ch):
    """
    Write a python function to remove
     first and last occurrence of a
     given character from the string.
    >>> remove_Occ("hello","l")
    "heo"
    >>> remove_Occ("abcda","a")
    "bcd"
    >>> remove_Occ("PHP","P")
    "H"
    """
(new line)
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
(new line)
            break
```

Newline insertion

ReCode: Implementation

Code perturbations customize from Tree-sitter

https://tree-sitter.github.io/tree-sitter/playground

GitHub repository

Introduction

Language Bindings

Introduction

Tree-sitter is a parser generator tool and an incremental parsing library. It can build a concrete syntax tree for a source file and efficiently update the syntax tree as the source file is edited. Tree-sitter aims to be:

- **General** enough to parse any programming language
- **Fast** enough to parse on every keystroke in a text editor
- **Robust** enough to provide useful results even in the presence of syntax errors
- **Dependency-free** so that the runtime library (which is written in pure C) can be embedded in any application

ReCode: Implementation

Code perturbations customize from Tree-sitter

https://tree-sitter.github.io/tree-sitter/playground

Text perturbations customized from NL-Augmenter

https://github.com/GEM-benchmark/NL-Augmenter

GitHub repository

Introduction

Language Bindings

Introduction

Tree-sitter is a parser generator tool and an incremental parsing library. It can build a concrete syntax tree for a source file and efficiently update the syntax tree as the source file is edited. Tree-sitter aims to be:

- **General** enough to parse any programming language
- **Fast** enough to parse on every keystroke in a text editor
- **Robust** enough to provide useful results even in the presence of syntax errors
- **Dependency-free** so that the runtime library (which is written in pure C) can be embedded in any application

NL-Augmenter 🦎 → 🐍

The NL-Augmenter is a collaborative effort intended to add transformations of datasets dealing with natural language. Transformations augment text datasets in diverse ways, including: randomizing names and numbers, changing style/syntax, paraphrasing, KB-based paraphrasing ... and whatever creative augmentation you contribute. We invite submissions of transformations to this framework by way of GitHub pull request.

ReCode: Implementation

Question:

- How to do perturbations for docstrings?

Original docstring

```python
def test_distinct(data):
    """
    Write a python function to determine whether all the
    numbers are different from each other are not.
    >>> test_distinct([1,5,7,9])
    True
    >>> test_distinct([2,4,5,5,7,9])
    False
    >>> test_distinct([1,2,3])
    True
    """
    return len(set(data)) == len(data)
```

Original completion

Perturbed docstring

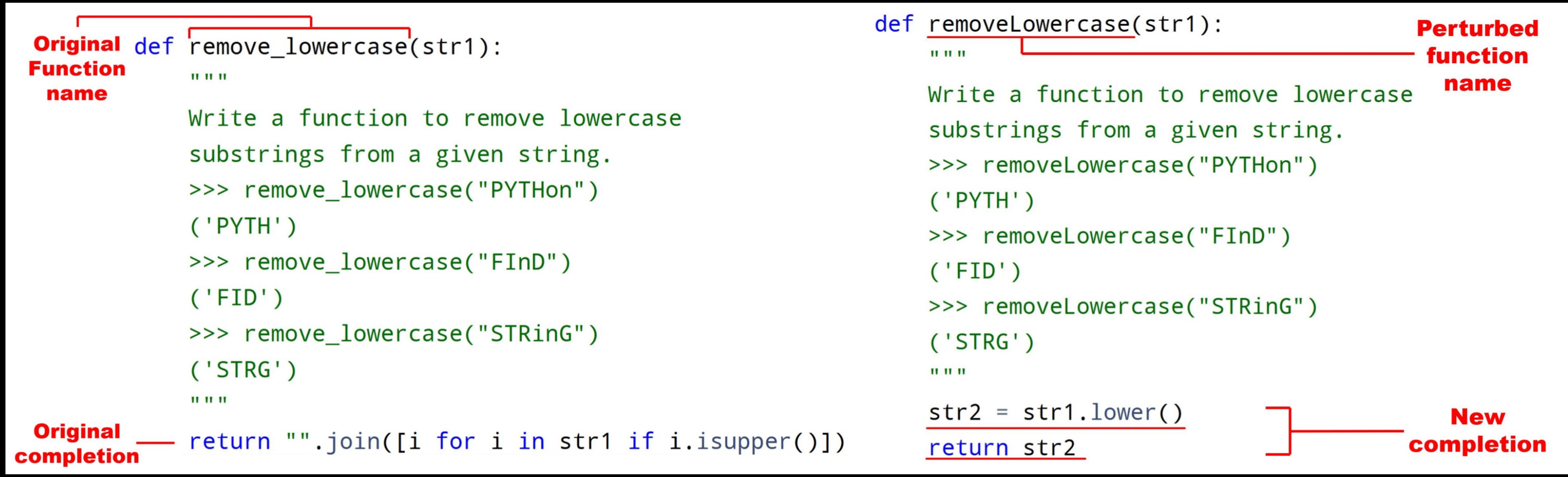
```python
def test_distinct(data):
    """
    Write a Python function to see if all
    numbers differ from each other.
    >>> test_distinct([1,5,7,9])
    True
    >>> test_distinct([2,4,5,5,7,9])
    False
    >>> test_distinct([1,2,3])
    True
    """
    return len(set(data)) != len(data)
```

New completion

ReCode: Implementation

Question:

- How to do perturbations for function rename?



ReCode: Implementation

Question:

- How to do perturbations for code syntax?

```
def remove_Occ(s, ch):
    """
    Write a python function to remove
     first and last occurrence of a
     given character from the string.
    >>> remove_Occ("hello","l")
    "heo"
    >>> remove_Occ("abcda","a")
    "bcd"
    >>> remove_Occ("PHP","P")
    "H"
    """
    for i in range(len(s)):
        if s[i] == ch:
            s = s[0:i] + s[i + 1 :]
            break
```

(a) Baseline Partial Code

```
def remove_Occ(s, ch):
    # [same doc string]
    i = 0
    while i < len(s):
        if s[i] == ch:
            s = s[0:i] + s[i + 1 :]
            break
        i = i + 1
```

(b) For-While Switch

```
def remove_Occ(lines, ch):
    # [same doc string]
    for i in range(len(lines)):
        if lines[i] == ch:
            lines = lines[0:i] + lines[i + 1 :]
            break
```

(c) Variable Renaming with CodeBERT

ReCode: Implementation

Question:

- How to do perturbations for code format?

```python
def remove_Occ(s, ch):
    """
    Write a python function to remove
     first and last occurrence of a
     given character from the string.
    >>> remove_Occ("hello","l")
    "heo"
    >>> remove_Occ("abcda","a")
    "bcd"
    >>> remove_Occ("PHP","P")
    "H"
    """
    for i in range(len(s)):
        if s[i] == ch:
            s = s[0:i] + s[i + 1 :]
            break
```

MBPP baseline partial code

```python
def remove_Occ(s, ch):
    # Write a python function to remove
    #  first and last occurrence of a
    #  given character from the string.
    # >>> remove_Occ("hello","l")
    # "heo"
    # >>> remove_Occ("abcda","a")
    # "bcd"
    # >>> remove_Occ("PHP","P")
    # "H"
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
            break
```

Docstring to comments

```python
def remove_Occ(s, ch):
    """
    Write a python function to remove
     first and last occurrence of a
     given character from the string.
    >>> remove_Occ("hello","l")
    "heo"
    >>> remove_Occ("abcda","a")
    "bcd"
    >>> remove_Occ("PHP","P")
    "H"
    """
(new line)
    for i in range(len(s)):
        if (s[i] == ch):
            s = s[0 : i] + s[i + 1:]
(new line)
            break
```

Newline insertion

aws

ReCode: Eval Metric for Code Models

“Functional Correct” – for each sampled code generation, if executing generated code passes the unit tests, we count it true.

First proposed in Codex paper, a code finetuned model based on GPT-3.

Greedy:

- Pass@1

Evaluating Large Language Models Trained on Code

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =⇒12
    solution([3, 3, 3, 3, 3]) =⇒9
    solution([30, 13, 24, 321]) =⇒0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

HumanEval Datasets

ReCode: Eval Metric for Code Models

“Functional Correct” – for each sampled code generation, if executing generated code passes the unit tests, we count it true.

First proposed in Codex paper, a code finetuned model based on GPT-3.

Sampling n = 1:

- Pass@1

Evaluating Large Language Models Trained on Code

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =⇒12
    solution([3, 3, 3, 3, 3]) =⇒9
    solution([30, 13, 24, 321]) =⇒0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

HumanEval Datasets

ReCode: Eval Metric for Code Models

"Functional Correct" – for each sampled code generation, if executing generated code passes the unit tests, we count it true.

First proposed in Codex paper, a code finetuned model based on GPT-3.

Sampling n = 100:

- Pass@100

Evaluating Large Language Models Trained on Code

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.

    Examples
    solution([5, 8, 7, 1]) =⇒12
    solution([3, 3, 3, 3, 3]) =⇒9
    solution([30, 13, 24, 321]) =⇒0
    """
    return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

HumanEval Datasets

ReCode: Eval Metric for Code Models

"Functional Correct" – for each sampled code generation, if executing generated code passes the unit tests, we count it true.

First proposed in Codex paper, a code finetuned model based on GPT-3.

Sampling n = 100:

- Pass@100

- Pass@1

ReCode: Eval Metric for Code Models

"Functional Correct" – for each sampled code generation, if executing generated code passes the unit tests, we count it true.

First proposed in Codex paper, a code finetuned model based on GPT-3.

Sampling n = 100:

- Pass@100
- Pass@1
- Pass@10
- Pass@k

aws

ReCode: Eval Metric for Code Models

"Functional Correct" – for each sampled code generation, if executing generated code passes the unit tests, we count it true.

First proposed in Codex paper, a code finetuned model based on GPT-3.

Sampling n = 100:

- Pass@1
- Pass@10
- Pass@100
- Pass@k

$$\text{pass@}k := \underset{\text{Problems}}{\mathbb{E}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]$$

c is the count of correct predictions out of n sampled generations for each problem

ReCode: Eval Metric for Code Models

"Functional Correct" – for each sampled code generation, if executing generated code passes the unit tests, we count it true.

First proposed in Codex paper, a code finetuned model based on GPT-3.

Sampling n = 100:

- Pass@1
- Pass@10
- Pass@100
- Pass@k

Model	pass@k [%]		
	$k = 1$	$k = 10$	$k = 100$
GPT-Neo 350M	0.85	2.55	5.95
GPT-Neo 2.7B	6.41	11.27	21.37
GPT-J 6B	11.62	15.74	27.74
Codex 300M	13.17	20.37	36.27
Codex 2.5B	21.36	35.42	59.50
Codex 12B	28.81	46.81	72.31
code-cushman-001*	33.5	54.3	77.4
code-davinci-001*	39.0	60.6	84.1
code-davinci-002*	47.0	74.9	92.1
CodeGen-NL 350M	2.12	4.10	7.38
CodeGen-NL 2.7B	6.70	14.15	22.84
CodeGen-NL 6.1B	10.43	18.36	29.85
CodeGen-NL 16.1B	14.24	23.46	38.33
CodeGen-Multi 350M	6.67	10.61	16.84
CodeGen-Multi 2.7B	14.51	24.67	38.56
CodeGen-Multi 6.1B	18.16	28.71	44.85
CodeGen-Multi 16.1B	18.32	32.07	50.80
CodeGen-Mono 350M	12.76	23.11	35.19
CodeGen-Mono 2.7B	23.70	36.64	57.01
CodeGen-Mono 6.1B	26.13	42.29	65.82
CodeGen-Mono 16.1B	**29.28**	**49.86**	**75.00**

Table 1: Evaluation results on the HumanEval benchmark. Each pass@k (where $k \in \{1, 10, 100\}$) for each model is computed with three sampling temperatures ($t \in \{0.2, 0.6, 0.8\}$) and the highest one among the three are displayed, which follows the evaluation procedure in Chen et al. (2021). Results for the model marked with * are from Chen et al. (2022).

Numbers from CodeGen Paper

GPT3 model is around 175B; GPT4 model is around 1.8T

ReCode: New Metrics

"Robustly Correct" – for each sampled code generation, if all s perturbations on prompts **cannot** make it **incorrect**, then this generation passes.

3 new robustness metrics

- Robust Pass@k (RP@k)
- Robust Drop@k (RD@k)
- Robust Relative@k (RR@k)

RP is the higher the better

RD and RR is the higher the worse.

ReCode: New Metrics

"Robustly Correct" – for each sampled code generation, if all s perturbations on prompts **cannot** make it **incorrect**, then this generation passes.

3 new robustness metrics

- Robust Pass@k (RP@k)
- Robust Drop@k (RD@k)
- Robust Relative@k (RR@k)

RP is the higher the better

RD and RR is the higher the worse.

n output samples following same prompt

$rc_s(x)$: How many generations are robustly correct

$$\mathrm{RP}_s@k := \mathbb{E}_x\left[1 - \frac{\binom{n - rc_s(x)}{k}}{\binom{n}{k}}\right]$$

"If we randomly choose k samples out of n generations, how likely we will find at least one "robustly correct" generation"

aws

ReCode: New Metrics

"Robustly Correct" – for each sampled code generation, if all s perturbations on prompts **cannot** make it **incorrect**, then this generation passes.

3 new robustness metrics

- Robust Pass@k (RP@k)
- Robust Drop@k (RD@k)
- Robust Relative@k (RR@k)

RP is the higher the better

RD and RR is the higher the worse.

n output samples following same prompt

$rc_s(x)$: How many generations are robustly correct

$$\mathrm{RP}_s@k := \mathbb{E}_x\left[1 - \frac{\binom{n - rc_s(x)}{k}}{\binom{n}{k}}\right]$$

"If we randomly choose k samples out of n generations, how likely we will find at least one "robustly correct" generation"

$$\mathrm{RD}_s@k := \frac{\mathrm{Pass@k} - \mathrm{Robust\ Pass}_s@k}{\mathrm{Pass@k}}$$

"Compared with original Pass@k, how much performance is dropped?"

ReCode: New Metrics

"Robustly Correct" – for each sampled code generation, if all s perturbations on prompts **cannot** make it **incorrect**, then this generation passes.

3 new robustness metrics

- Robust Pass@k (RP@k)
- Robust Drop@k (RD@k)
- Robust Relative@k (RR@k)

RP is the higher the better

RD and RR is the higher the worse.

n output samples following same prompt

$rc_s(x)$: How many generations are robustly correct

$$\mathrm{RP}_s@k := \mathbb{E}_x\left[1 - \frac{\binom{n-rc_s(x)}{k}}{\binom{n}{k}}\right]$$

"If we randomly choose k samples out of n generations, how likely we will find at least one "robustly correct" generation"

$$\mathrm{RD}_s@k := \frac{\mathrm{Pass@k} - \mathrm{Robust\ Pass}_s@k}{\mathrm{Pass@k}}$$

"Compared with original Pass@k, how much performance is dropped?"

$$\mathrm{RR}_s@1 := \frac{RC_s^{[+]} + RC_s^{[-]}}{N}$$

How many output samples we change from incorrect—> correct under any of s perturbation (best-case analysis)

How many output samples we change from correct—> incorrect under any of s perturbation (worst-case analysis)

aws

ReCode: Evaluation on Public Models

Public models (decoder only)

- CodeGen from Salesforce
    - Natural language training first (THEPILE) and then code data from github (Bigquery from google)
    - CodeGen-mono: only train on bigpython
    - CodeGen-multi: train on multiple languages in bigquery including C, C++, Go, Java, JavaScript, and Python
- InCoder from Meta
    - Bidirectional context
- GPT-J from EleutherAI
    - Mainly pretrained with THEPILE and then finetune with python code

ReCode: Evaluation on Public Models

Public models (decoder only)

- CodeGen from Salesforce
    - Natural language training first (THEPILE) and then code data from github (Bigquery from google)
    - CodeGen-mono: only train on bigpython
    - CodeGen-multi: train on multiple languages in bigquery including C, C++, Go, Java, JavaScript, and Python
- InCoder from Meta
    - Bidirectional context
- GPT-J from EleutherAI
    - Mainly pretrained with THEPILE and then finetune with python code
- Other architectures (not evaluated)
    - CodeT5 (encoder-decoder)
    - CoderBERT/CodeGraphBERT (encoder only)

ReCode: Evaluation on Public Models

Public models (decoder only)

- CodeGen from Salesforce
  - Natural language training first (THEPILE) and then code data from github (Bigquery from google)
  - CodeGen-mono: only train on bigpython
  - CodeGen-multi: train on multiple languages in bigquery including C, C++, Go, Java, JavaScript, and Python
- InCoder from Meta
  - Bidirectional context
  - 28 languages, mainly on python
- GPT-J from EleutherAI
  - Mainly pretrained with THEPILE and then finetune with python code
- Other architectures (not evaluated)
  - CodeT5 (encoder-decoder)
  - CoderBERT/CodeGraphBERT (encoder only)

Model	Size (B)	Python Code (GB)	Other Code (GB)	Other (GB)	Code License	Infill?	HE @1	HE @10	HE @100	MBPP @1
Released										
CodeParrot (Tunstall et al., 2022)	1.5	50	None	None	—		4.0	8.7	17.9	—
PolyCoder (Xu et al., 2022)	2.7	16	238	None	—		5.6	9.8	17.7	—
GPT-J (Wang & Komatsuzaki, 2021; Chen et al., 2021a)	6	6	90	730	—		11.6	15.7	27.7	—
INCODER-6.7B	6.7	52	107	57	Permissive	✓	15.2	27.8	47.0	19.4
GPT-NeoX (Black et al., 2022)	20	6	90	730	—		15.4	25.6	41.2	—
CodeGen-Multi (Nijkamp et al., 2022)	6.1	62	375	1200	—		18.2	28.7	44.9	—
CodeGen-Mono (Nijkamp et al., 2022)	6.1	279	375	1200	—		26.1	42.3	65.8	—
CodeGen-Mono (Nijkamp et al., 2022)	16.1	279	375	1200	—		29.3	49.9	75.0	—
Unreleased										
LaMDA (Austin et al., 2021; Thoppilan et al., 2022; Chowdhery et al., 2022)	137	None	None	???	—		14.0	—	47.3	14.8
AlphaCode (Li et al., 2022)	1.1	54	660	None	—		17.1	28.2	45.3	—
Codex-2.5B (Chen et al., 2021a)	2.5	180	None	> 570	—		21.4	35.4	59.5	—
Codex-12B (Chen et al., 2021a)	12	180	None	> 570	—		28.8	46.8	72.3	—
PaLM-Coder (Chowdhery et al., 2022)	540	~20	~200	~4000	Permissive		36.0	—	88.4	47.0

Table 11: A comparison of our INCODER-6.7B model to published code generation systems using pass rates @ K candidates sampled on the HumanEval and MBPP benchmarks. All models are decoder-only transformer models. A "Permissive" code license indicates models trained on only open-source repositories with non-copyleft licenses. The GPT-J, GPT-NeoX, and CodeGen models are pre-trained on The Pile (Gao et al., 2020), which contains a portion of GitHub code without any license filtering, including 6 GB of Python. Although the LaMDA model does not train on code repositories, its training corpus includes ~18 B tokens of code from web documents. The total file size of the LaMDA corpus was not reported, but it contains 2.8 T tokens total. We estimate the corpus size for PaLM using the reported size of the code data and the token counts per section of the corpus.

Numbers from InCoder Paper

ReCode: Evaluation on Public Models

Empirical Observations

- Architecture-wise: CodeGen, InCoder, GPT-J performs across
- Model Size-wise
- Perturbation-wise

MBPP	Metric	CodeGen 2B mono	CodeGen 2B multi	CodeGen 6B mono	CodeGen 6B multi	CodeGen 16B mono	CodeGen 16B multi	InCoder 1B	InCoder 6B	GPT-J 6B
Docstring	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	RP$_5$@1↑	0.137	0.050	0.147	0.042	**0.163**	0.045	0.011	0.031	0.013
	RD$_5$@1(%)↓	**56.96**	73.66	59.38	80.93	59.85	81.28	91.20	84.54	90.00
	RR$_5$@1(%)↓	36.86	34.39	41.89	36.76	46.72	44.66	**25.57**	35.32	30.08
Function	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	RP$_5$@1↑	0.221	0.101	0.252	0.110	**0.279**	0.139	0.047	0.087	0.043
	RD$_5$@1(%)↓	30.42	47.31	**30.40**	50.23	31.31	42.55	63.20	56.19	67.69
	RR$_5$@1(%)↓	19.51	20.43	24.13	22.79	24.95	23.51	**16.22**	20.02	17.56
Syntax	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	RP$_5$@1↑	0.027	0.008	0.027	0.008	**0.038**	0.017	0.008	0.006	0.004
	RD$_5$@1(%)↓	**94.06**	97.12	95.01	97.52	93.34	95.39	96.24	97.89	97.66
	RR$_5$@1(%)↓	59.03	45.07	64.17	47.74	67.04	54.21	35.42	45.79	**30.60**
Format	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	RP$_5$@1↑	0.333	0.146	0.289	0.166	**0.403**	0.214	0.091	0.130	0.080
	RD$_5$@1(%)↓	**26.03**	48.92	46.07	49.69	29.32	43.63	58.22	55.28	54.39
	RR$_5$@1(%)↓	19.82	25.15	31.11	27.00	25.26	26.59	19.61	28.54	**18.28**

ReCode: Evaluation on Public Models

Empirical Observations

- Architecture-wise: CodeGen, InCoder, GPT-J performs across
- Model Size-wise
- Perturbation-wise

With same size 6B, CodeGen achieves better performance on Nominal + RP_5@1, a very strict robustness metric

<u>"Diverse pretraining corpus helps with both generalization and worst-case robustness."</u>

MBPP	Metric	CodeGen 2B mono	CodeGen 2B multi	CodeGen 6B mono	CodeGen 6B multi	CodeGen 16B mono	CodeGen 16B multi	InCoder 1B	InCoder 6B	GPT-J 6B
Docstring	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	RP$_5$@1↑	0.137	0.050	0.147	0.042	**0.163**	0.045	0.011	0.031	0.013
	RD$_5$@1(%)↓	**56.96**	73.66	59.38	80.93	59.85	81.28	91.20	84.54	90.00
	RR$_5$@1(%)↓	36.86	34.39	41.89	36.76	46.72	44.66	**25.57**	35.32	30.08
Function	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	RP$_5$@1↑	0.221	0.101	0.252	0.110	**0.279**	0.139	0.047	0.087	0.043
	RD$_5$@1(%)↓	30.42	47.31	**30.40**	50.23	31.31	42.55	63.20	56.19	67.69
	RR$_5$@1(%)↓	19.51	20.43	24.13	22.79	24.95	23.51	**16.22**	20.02	17.56
Syntax	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	RP$_5$@1↑	0.027	0.008	0.027	0.008	**0.038**	0.017	0.008	0.006	0.004
	RD$_5$@1(%)↓	**94.06**	97.12	95.01	97.52	93.34	95.39	96.24	97.89	97.66
	RR$_5$@1(%)↓	59.03	45.07	64.17	47.74	67.04	54.21	35.42	45.79	**30.60**
Format	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	RP$_5$@1↑	0.333	0.146	0.289	0.166	**0.403**	0.214	0.091	0.130	0.080
	RD$_5$@1(%)↓	**26.03**	48.92	46.07	49.69	29.32	43.63	58.22	55.28	54.39
	RR$_5$@1(%)↓	19.82	25.15	31.11	27.00	25.26	26.59	19.61	28.54	**18.28**

ReCode: Evaluation on Public Models

Empirical Observations

- Architecture-wise: CodeGen, InCoder, GPT-J performs across
- Model Size-wise
- Perturbation-wise

CodeGen-mono 2B to 16B improved RP from 0.174 to 0.217 on average across all perturbations

<u>"Larger model size brings improvement in worst-case robustness, but may risk overfitting."</u>

MBPP	Metric	CodeGen 2B mono	CodeGen 2B multi	CodeGen 6B mono	CodeGen 6B multi	CodeGen 16B mono	CodeGen 16B multi	InCoder 1B	InCoder 6B	GPT-J 6B
Docstring	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	$RP_5@1$↑	0.137	0.050	0.147	0.042	**0.163**	0.045	0.011	0.031	0.013
	$RD_5@1(\%)$↓	**56.96**	73.66	59.38	80.93	59.85	81.28	91.20	84.54	90.00
	$RR_5@1(\%)$↓	36.86	34.39	41.89	36.76	46.72	44.66	**25.57**	35.32	30.08
Function	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	$RP_5@1$↑	0.221	0.101	0.252	0.110	**0.279**	0.139	0.047	0.087	0.043
	$RD_5@1(\%)$↓	30.42	47.31	**30.40**	50.23	31.31	42.55	63.20	56.19	67.69
	$RR_5@1(\%)$↓	19.51	20.43	24.13	22.79	24.95	23.51	**16.22**	20.02	17.56
Syntax	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	$RP_5@1$↑	0.027	0.008	0.027	0.008	**0.038**	0.017	0.008	0.006	0.004
	$RD_5@1(\%)$↓	**94.06**	97.12	95.01	97.52	93.34	95.39	96.24	97.89	97.66
	$RR_5@1(\%)$↓	59.03	45.07	64.17	47.74	67.04	54.21	35.42	45.79	**30.60**
Format	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	$RP_5@1$↑	0.333	0.146	0.289	0.166	**0.403**	0.214	0.091	0.130	0.080
	$RD_5@1(\%)$↓	**26.03**	48.92	46.07	49.69	29.32	43.63	58.22	55.28	54.39
	$RR_5@1(\%)$↓	19.82	25.15	31.11	27.00	25.26	26.59	19.61	28.54	**18.28**

ReCode: Evaluation on Public Models

Empirical Observations

- Architecture-wise: CodeGen, InCoder, GPT-J performs across
- Model Size-wise
- Perturbation-wise

<u>"Code generation models are most sensitive to syntax perturbation."</u>

MBPP	Metric	CodeGen 2B mono	CodeGen 2B multi	CodeGen 6B mono	CodeGen 6B multi	CodeGen 16B mono	CodeGen 16B multi	InCoder 1B	InCoder 6B	GPT-J 6B
Docstring	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	$RP_5@1$↑	0.137	0.050	0.147	0.042	**0.163**	0.045	0.011	0.031	0.013
	$RD_5@1(\%)$↓	**56.96**	73.66	59.38	80.93	59.85	81.28	91.20	84.54	90.00
	$RR_5@1(\%)$↓	36.86	34.39	41.89	36.76	46.72	44.66	**25.57**	35.32	30.08
Function	Nominal↑	0.317	0.191	0.361	0.221	**0.407**	0.241	0.128	0.199	0.133
	$RP_5@1$↑	0.221	0.101	0.252	0.110	**0.279**	0.139	0.047	0.087	0.043
	$RD_5@1(\%)$↓	30.42	47.31	**30.40**	50.23	31.31	42.55	63.20	56.19	67.69
	$RR_5@1(\%)$↓	19.51	20.43	24.13	22.79	24.95	23.51	**16.22**	20.02	17.56
Syntax	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	$RP_5@1$↑	0.027	0.008	0.027	0.008	**0.038**	0.017	0.008	0.006	0.004
	$RD_5@1(\%)$↓	**94.06**	97.12	95.01	97.52	93.34	95.39	96.24	97.89	97.66
	$RR_5@1(\%)$↓	59.03	45.07	64.17	47.74	67.04	54.21	35.42	45.79	**30.60**
Format	Nominal↑	0.450	0.285	0.535	0.331	**0.571**	0.379	0.219	0.292	0.176
	$RP_5@1$↑	0.333	0.146	0.289	0.166	**0.403**	0.214	0.091	0.130	0.080
	$RD_5@1(\%)$↓	**26.03**	48.92	46.07	49.69	29.32	43.63	58.22	55.28	54.39
	$RR_5@1(\%)$↓	19.82	25.15	31.11	27.00	25.26	26.59	19.61	28.54	**18.28**

ReCode: Evaluation on Public Models

Empirical Observations

- Architecture-wise: CodeGen, InCoder, GPT-J performs across
- Model Size-wise
- Perturbation-wise

MBPP has more variances in code style (e.g., indent with 1 space), closer to natural code distribution hence more challenging for model robustness.

Category	Metric	HumanEval	MBPP
Docstring	$RP_5@1\uparrow$	**0.078**	0.071
	$RD_5@1(\%)\downarrow$	**60.67**	75.31
	$RR_5@1(\%)\downarrow$	**19.72**	36.92
Function	$RP_5@1\uparrow$	0.113	**0.142**
	$RD_5@1(\%)\downarrow$	**41.61**	46.59
	$RR_5@1(\%)\downarrow$	**12.06**	21.01
Syntax	$RP_5@1\uparrow$	**0.100**	0.025
	$RD_5@1(\%)\downarrow$	**72.58**	93.40
	$RR_5@1(\%)\downarrow$	**33.88**	47.86
Format	$RP_5@1\uparrow$	**0.211**	0.206
	$RD_5@1(\%)\downarrow$	**43.30**	45.73
	$RR_5@1(\%)\downarrow$	**22.70**	24.60

aws

ReCode

Empirical Observations

- Architecture-wise: CodeGen, InCoder, GPT-J performs across
- Model size wise: 350M, 2B, 6B, 16B
- Mono-lingual vs multi-lingual
- Dataset wise: HumanEval vs MBPP

Check out our paper and release code and datasets

- **Paper: https://arxiv.org/abs/2212.10264**
- **Code and datasets: https://github.com/amazon-science/recode**

ReCode

Empirical Observations

- Architecture-wise: CodeGen, InCoder, GPT-J performs across
- Model size wise: 350M, 2B, 6B, 16B
- Mono-lingual vs multi-lingual
- Dataset wise: HumanEval vs MBPP

Check out our paper and release code and datasets

- **Paper: https://arxiv.org/abs/2212.10264**
- **Code and datasets: https://github.com/amazon-science/recode**

Check out CodeWhisperer for free!!!

https://aws.amazon.com/codewhisperer/