Microsoft

模型可以自我验证吗？
探究基于测试用例生成的代码生成方法

陈蓓 微软亚洲研究院
beichen@microsoft.com
MLNLP 2022-11-27

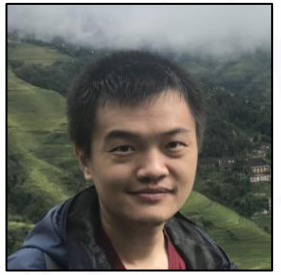
CodeT: Code Generation with Generated Tests


Bei Chen


Fengji Zhang


Anh Nguyen


Daoguang Zan


Zeqi Lin


Jian-Guang Lou


Weizhu Chen

- Code Generation - A very effective yet simple approach to improve code generation with pre-trained language models.
- Test Case Generation – Automatic test case generation to unleash the inherent power of pre-trained language models.

Large Language Models for Code Generation

Codex

AlphaCode

InCoder

CodeGen

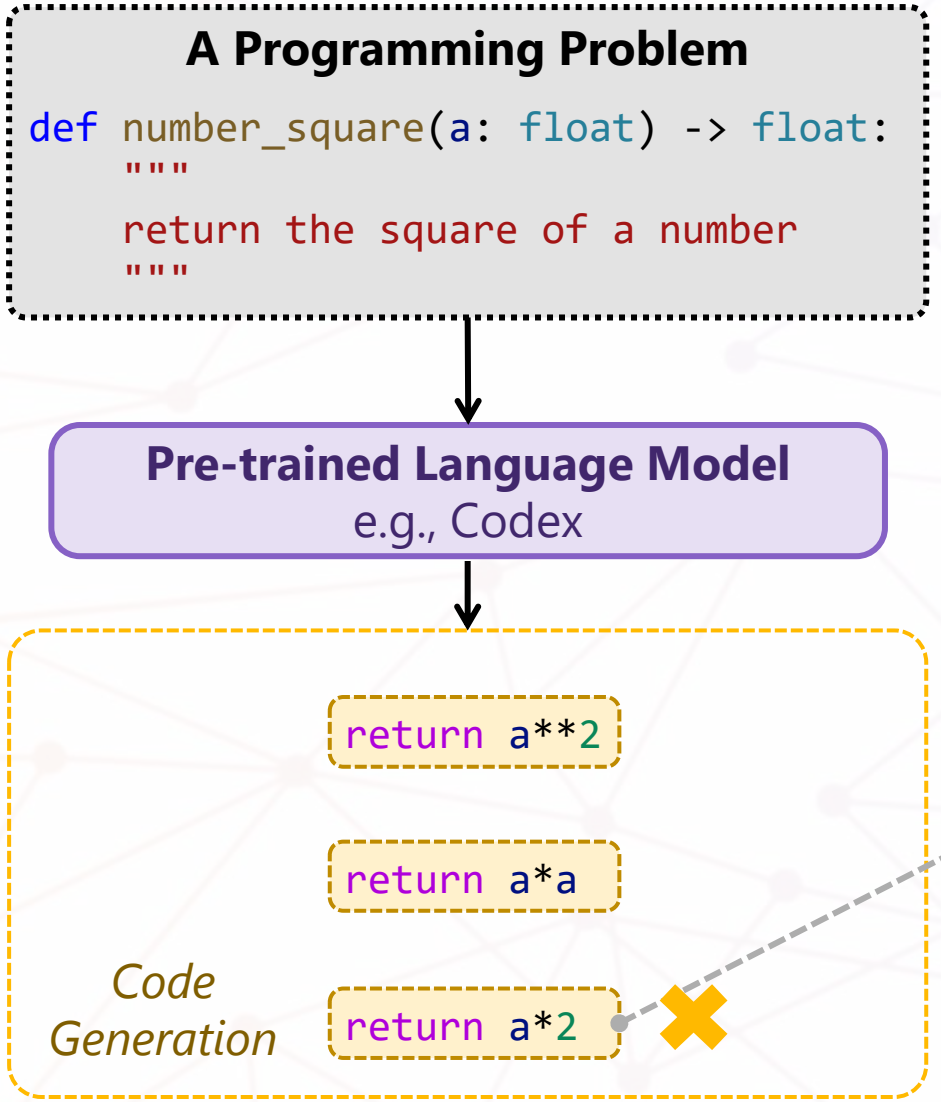
Methods	Baseline		
k	1	10	100
HumanEval			
code-cushman-001	33.5	54.3	77.4
code-davinci-001	39.0	60.6	84.1
code-davinci-002	47.0	74.9	92.1
MBPP			
code-cushman-001	45.9	66.9	79.9
code-davinci-001	51.8	72.8	84.1
code-davinci-002	58.1	76.7	84.5

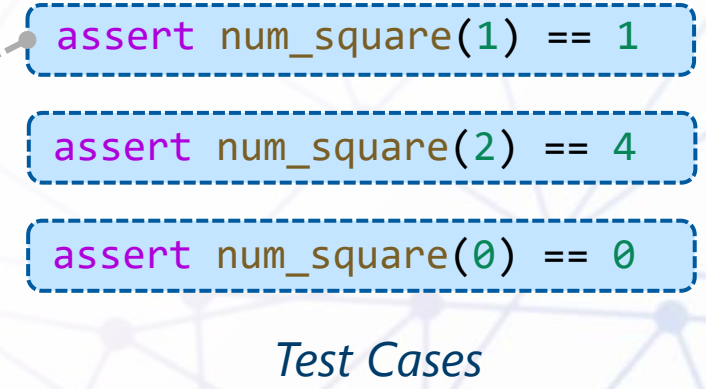
Codex pass@k on HumanEval and MBPP

Pass@k: Percentage of solved problems. For each problem, k code solutions are produced. If any of the k code solutions are correct, the problem is considered solved.

How to pick the best code solution from multiple candidates?

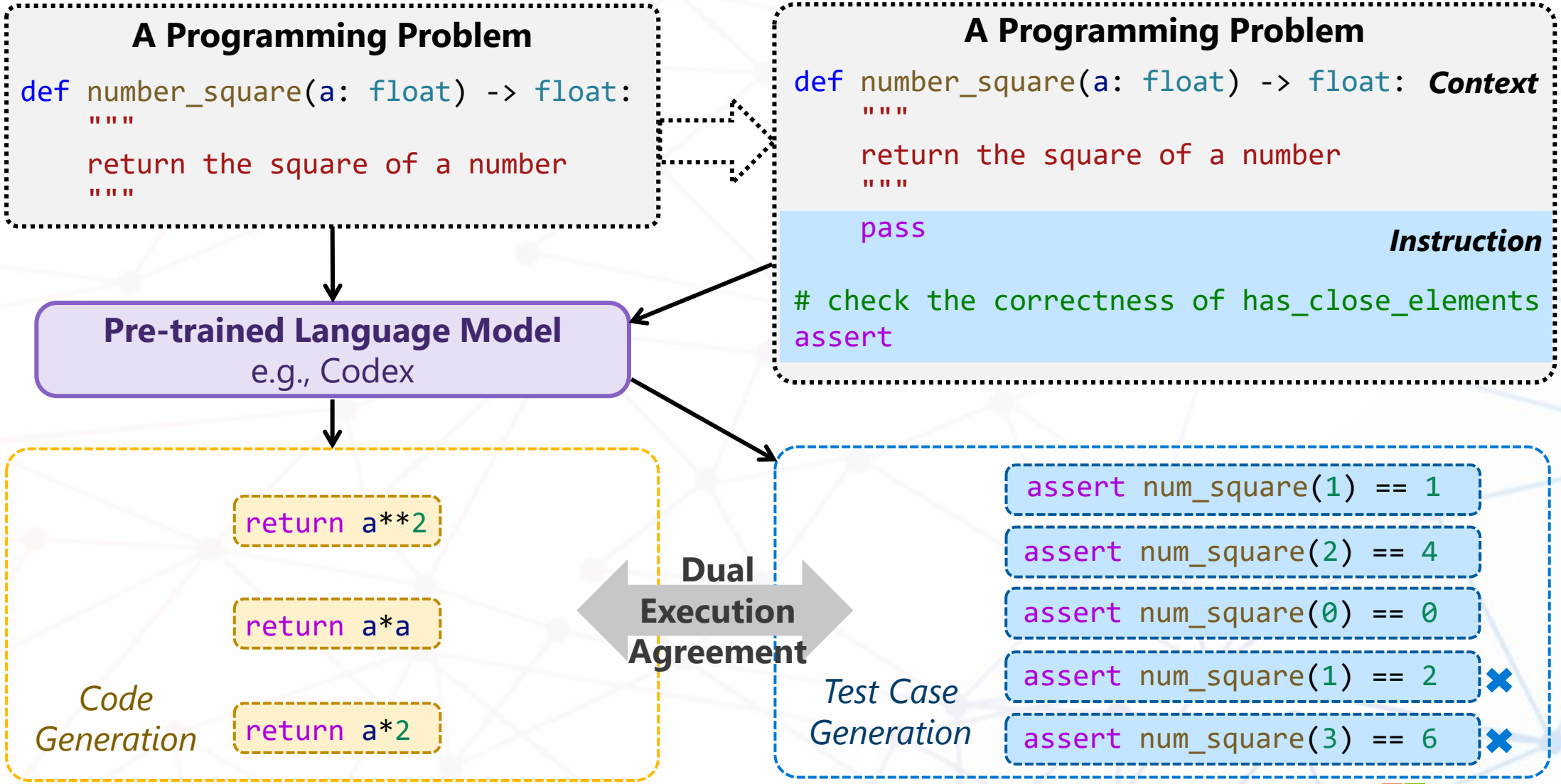
Execute Code!





- It relies heavily on the quality and quantity of test cases, which are often costly and time-consuming to create and maintain.
- In real-world applications like Copilot, it is unrealistic to expect users to provide test cases for every problem they want to solve.

Generate Test Case!



Dual Execution Agreement

Assumptions

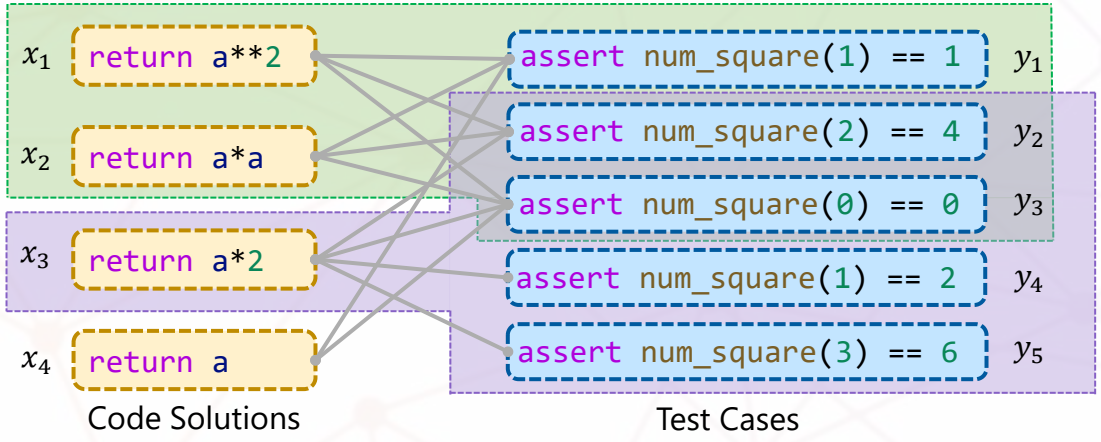
- The code solutions and the test cases are independently and randomly sampled from the pre-trained language model given a certain programming problem.
- Incorrect code solutions are often diverse, and the probability of having a functionality agreement between two incorrect code solutions by chance is very low.

Similar to classical RANSAC algorithm

Dual Execution Agreement

1. We randomly select a pair (x, y) from the set of all possible pairs, and execute the code solution x on the test case y. If x can pass y, then we say (x, y) is a hypothetical inlier; otherwise, (x, y) is an outlier.
2. If (x, y) is a hypothetical inlier, we collect all other pairs that agree it, forming a consensus set $\mathcal{S}$. In detail, we first find all test cases that x can pass, denoted as $\mathcal{S}_y$, and then find all code solutions that can pass exactly the same test cases as x, denoted as $\mathcal{S}_x$.
3. We score the consensus set as $\mathcal{S} = |\mathcal{S}_x||\mathcal{S}_y|$. This score equals to the number of pairs in the consensus set.

When the number of code solutions is not large, we can simplify this iterative method by examining all possible pairs.



- $(x_1, y_1), (x_3, y_2)$ are two of the hypothetical inliers.
- $(x_1, y_4), (x_3, y_1)$ are two of the outliers.

- For (x_1, y_1), $\mathcal{S}_y = \{y_1, y_2, y_3\}$ and $\mathcal{S}_x = \{x_1, x_2\}$
  - Consensus score is 6
- For (x_3, y_2), $\mathcal{S}_y = \{y_3, y_3, y_4, y_5\}$ and $\mathcal{S}_x = \{x_3\}$
  - Consensus score is 4

Experiments: Pre-trained Language Models

Codex

OpenAI

[Code-Cushman-001, code-davinci-001/002]

- GPT-like structure
- Powering Github Copilot
- 12B~175B parameters
- Trained with Github code files

InCoder

Meta AI

[InCoder-6B]

- GPT-like structure
- Fill-in-middle masking
- 6.7B parameters
- Trained with data from ThePile, BigQuery, Github

CodeGen

salesforce

[CodeGen-Mono-16B]

- GPT-like structure
- Biggest public code pre-trained model
- 16B parameters
- Trained with data from BigQuery, Github, StackOverflow

Experiments: Benchmarks

- Four public code generation benchmarks
  - **HumanEval**: 164 hand-written Python programming problems
  - **MBPP** (sanitized): 427 crowd-sourced Python programming problems
  - **APPS**: 5000 problems in the testset (1000 introductory, 3000 interview, and 1000 competition)
  - **CodeContests**: 165 competitive programming problems

Input-output examples are removed to avoid exposing real test cases.

```
def has_close_elements(numbers: List[float], threshold: float) -> bool:
        """ Check if in given list of numbers, are any two numbers
        closer to each other than given threshold.
        >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
        False
        >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
        True
        """
```

Removed

Experiments: CodeT Improves Pass@1

Methods	Baseline			CodeT		
k	1	10	100	1	2	10
HumanEval						
code-cushman-001	33.5	54.3	77.4	44.5 11.0	50.1	65.7 11.4
code-davinci-001	39.0	60.6	84.1	50.2 11.2	58.9	75.8 15.2
code-davinci-002	47.0	74.9	92.1	65.8 18.8	75.1	86.6 11.7
InCoder-6B	16.4 15.2	28.3 27.8	47.5 47.0	20.6 4.2	27.6	37.1 8.8
CodeGen-Mono-16B	29.7 29.3	50.3 49.9	73.7 75.0	36.7 7.0	44.7	59.3 9.0
MBPP						
code-cushman-001	45.9	66.9	79.9	55.4 9.5	61.7	72.7 5.8
code-davinci-001	51.8	72.8	84.1	61.9 10.1	69.1	79.3 6.5
code-davinci-002	58.1	76.7	84.5	67.7 9.6	74.6	81.5 4.8
InCoder-6B	21.3 19.4	46.5	66.2	34.4 13.1	43.9	58.2 11.7
CodeGen-Mono-16B	42.4	65.8	79.1	49.5 7.1	56.6	68.5 2.7

Pass@k (%) with Codex, InCoder and CodeGen on HumanEval and MBPP

- CodeT consistently achieves significant improvements over the baseline.
- The pass@2 results of CodeT are close to the baseline pass@10 results.
- With code-davinci-002, CodeT achieves an 18.8% absolute improvements on HumanEval, boosting the pass@1 to 65.8%. It is an absolute improvement of 20+% over SOTA.

Experiments: CodeT Improves Pass@1

Methods		Baseline					CodeT			
k		1	10	50	100	1000	1	2	10	100
APPS	Introductory	27.2	46.6	59.4	-	-	34.6 7.4	41.2	53.2 6.6	-
	Interview	5.1	12.8	23.0	-	-	8.1 3.0	11.2	18.1 5.3	-
	Competition	1.8	4.9	12.1	-	-	2.2 0.4	4.1	8.6 3.7	-
CodeContests		0.7	3.0	5.7	7.5	13.9	2.1 1.4	2.3	5.3 2.3	9.9 2.4

Pass@k (%) with Codex (code-davinci-002) on APPS and CodeContests

- The sample number is 50 for APPS, 1000 for CodeContests.
- Consistent performance improvements are observed on both benchmarks using CODET.

Experiments: Removing Input-Output is Reasonable

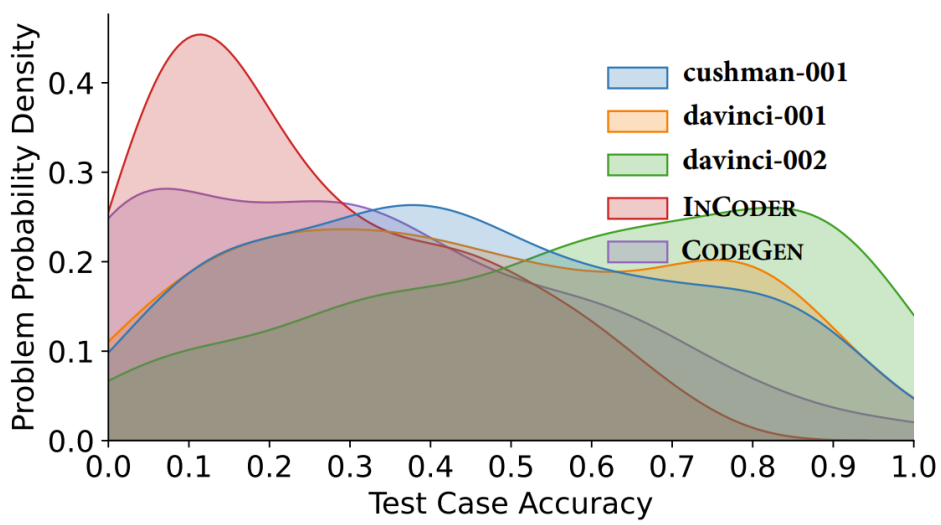
Methods	Baseline			CodeT		
k	1	10	100	1	2	10
code-cushman-001	31.7 −1.8	56.4 2.1	84.1 6.7	58.6 14.1	65.7 15.6	80.1 14.4
code-davinci-001	34.8 −4.2	63.0 2.4	87.2 3.1	60.4 10.2	69.1 10.2	82.4 6.6
code-davinci-002	47.6 0.6	78.8 3.9	92.7 0.6	74.8 9.0	82.9 7.8	89.0 2.4

Pass@k (%) with Codex on the original HumanEval

- The baseline pass@1 results on the original benchmark are basically the same or even worse than the modified version.
- The performance of CodeT is significantly improved using the original benchmark.

Experiments: Test Case Analysis

What is the quality of the generated test cases?



- Horizontal axis: accuracy value for each problem
- Vertical axis: probability density of problems
- Test cases generated by Codex models are of much higher accuracy than CodeGen and InCoder.

Experiments: Test Case Analysis

Can better test cases further boost the performance of mediocre models?

Benchmarks	HumanEval			MBPP		
k	1	2	10	1	2	10
code-cushman-001	47.1 2.6	58.6 8.5	71.2 5.5	59.7 4.3	64.8 3.1	75.5 2.8
code-davinci-001	52.0 1.8	62.9 4.0	78.1 2.3	64.3 2.4	71.7 2.6	80.5 1.2
INCODER-6B	26.8 6.2	30.4 2.8	40.8 3.7	50.3 15.9	55.4 11.5	64.5 6.3
CODEGEN-MONO-16B	47.7 11.0	54.9 10.2	71.0 11.7	60.0 10.5	67.6 11.0	76.5 8.0

Pass@k (%) on the HumanEval and MBPP using the test cases generated by code-davinci-002.

- Using the test cases generated by code-davinci-002 can significantly improve the performance of using the test cases generated by the less capable models themselves

Experiments: Test Case Analysis

How effective is CodeT when there are fewer test cases?

Limit	Sampling Number			
	10	20	50	100
code-cushman-001				
1	37.8	40.0	40.8	38.7
2	42.1	41.8	43.4	41.8
3	41.6	41.9	43.8	42.5
4	41.2	41.2	43.8	43.3
5	41.0	41.9	45.4	44.5
code-davinci-002				
1	56.5	57.5	60.7	62.4
2	62.2	62.8	63.2	63.6
3	62.9	63.2	65.5	65.0
4	64.1	64.5	65.7	65.0
5	63.9	64.2	65.2	65.8

(a) pass@1

Limit	Sampling Number			
	10	20	50	100
code-cushman-001				
1	43.3	48.1	48.2	49.1
2	48.1	48.1	49.5	49.8
3	49.0	47.7	48.7	48.7
4	49.2	47.9	49.4	49.1
5	48.3	48.5	48.9	50.1
code-davinci-002				
1	65.1	67.8	71.9	71.5
2	71.7	73.2	74.2	74.1
3	73.2	73.5	75.1	75.0
4	73.3	74.1	75.5	74.3
5	73.5	74.3	74.5	75.1

(b) pass@2

Limit	Sampling Number			
	10	20	50	100
code-cushman-001				
1	55.1	56.6	61.9	62.9
2	58.7	61.4	64.5	65.8
3	60.9	62.5	63.4	65.3
4	61.4	63.3	63.3	65.8
5	63.1	62.6	63.8	65.7
code-davinci-002				
1	77.9	79.6	82.8	84.3
2	80.8	81.8	84.3	86.5
3	82.3	83.2	85.5	87.1
4	82.9	84.4	85.4	86.9
5	83.8	84.1	85.2	86.6

(c) pass@10

Pass@k (%) on the HumanEval using different numbers of test cases.

- Using more test cases in CodeT could generally lead to better performance.
- The performance gap narrows when Sampling Number ≥ 50 and Limit ≥ 3.

Key learnings and insights

- The pre-trained language models can be used to automatically generate test cases.
  - Assert statements is a kind of code and exist in the pre-trained data.
  - Instructions can be designed to tell language models to generate test cases.
- The performance of code generation can be significantly improved using the execution-based method and an agreement mechanism.
  - There are enough correct code solutions, and the incorrect ones are diverse.
  - The idea of agreement can be extended to other scenarios.
- Model can do self-validation.
  - Natural language problem descriptions, code solutions, test cases, and other kinds of specification, all of which can express the intention of a single task. We can use them to verify each other.

Welcome to try!

Paper

Code & Data

Paper

CodeT: Code Generation with Generated Tests

Code and Data

https://github.com/microsoft/CodeT

Welcome to join us!

微软亚洲研究院数据、知识、智能（DKI）组

Research Topics:

代码智能，推理与组合泛化，可信

语义解析，人工智能与平面设计

FTE & Intern Application

beichen@microsoft.com

di-recruit@microsoft.com

Thanks & QA!

陈蓓 微软亚洲研究院
beichen@microsoft.com
MLNLP 2022-11-27