Is Your Code Generated by ChatGPT Really Correct?

Rigorous Evaluation of Large Language Models for Code Generation

Jiawei Liu*, Chunqiu Steven Xia*
Yuyao Wang, Lingming Zhang

University of Illinois Urbana-Champaign

Nanjing University

Outline

- Background & Motivation
  - What is code generation?
  - How to evaluate code generation?
  - What is wrong with current benchmarks?
- Technique
  - Seed initialization via ChatGPT
  - Type-aware mutation
  - Test suite reduction
- Evaluation
  - Pass rate of HumanEval and HumanEval+
  - Understanding the pass rate drop

LLMs for code generation

```python
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

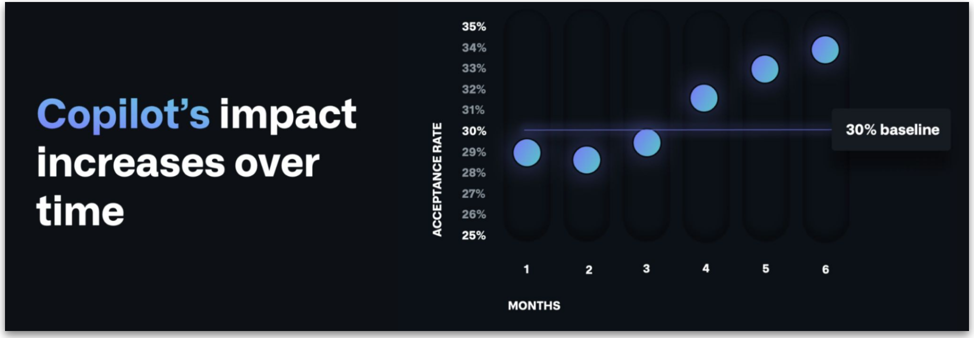
🤖 LLM

LLMs for code generation

- LLMs trained on code massively boost dev productivity
  - **2021&2022:** OpenAI Codex, GitHub Copilot, CodeT5, AlphaCode, etc.
  - **2023:** CodeGen (v2), PaLM 2, StarCoder, CodeLlama, CodeT5+, etc.

GitHub Copilot has been activated by more than **one million developers** and adopted by over **20,000 organizations**. It has generated over **three billion accepted lines of code**, and is the world's most widely adopted AI developer tool.

https://github.blog/2023-06-27-the-economic-impact-of-the-ai-powered-developer-lifecycle-and-lessons-from-github-copilot/



Evaluating LLMs for code

- HumanEval (OpenAI) and MBPP (Google)
  - **Input**: Function signature + Docstring (description + examples)
  - **Output**: Code completion to be exercised by a few test-cases

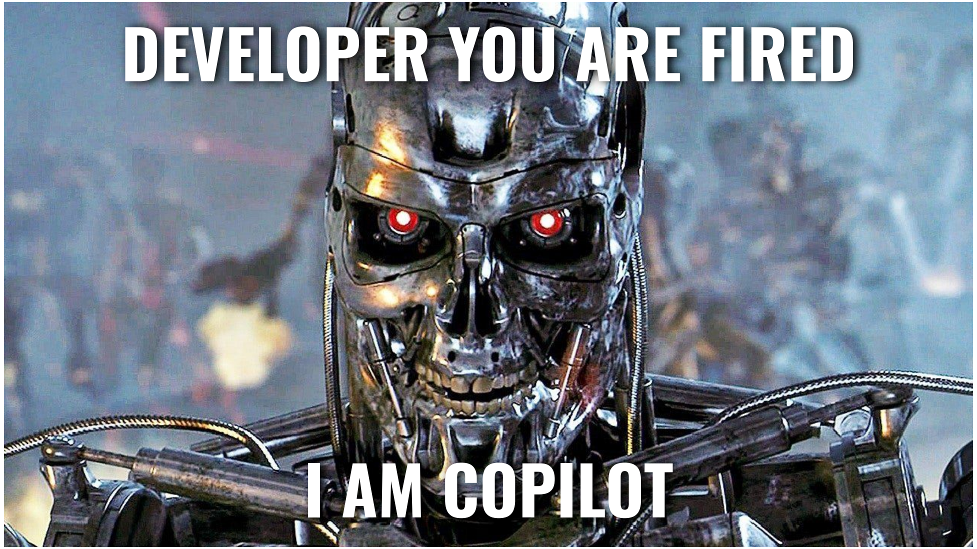
🧑‍💻 Prompt

```
def fibonacci(n):
    """Return n-th Fibonacci number
     >>> fib(10) = 55
     >>> fib(1)  = 1"""
    if n ⩽ 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

🤖 LLM

AI Coders “solve” ~90% problems

- 🤖 GPT-4 passes **88.4%** HumanEval tasks in one shot!
- 😱 Can LLMs replace humans for programming?
- 🤔 Is it too good to be true?



Test insufficiency

- 😱 Each MBPP problem has **3 tests**
- 😱 Each HumanEval problem has **<10 tests** on avg
- 🤔 Are these solutions *really* correct???

```
def common(l1: list, l2: list) → list:
    """Return sorted unique common elements for two lists"""
    common_elems = list(set(l1).intersection(set(l2)))
    common_elems.sort()
    return list(set(common_elems))
```

```
✅ common([4,3,2,8], [])     ⇒ []
✅ common([5,3,2,8], [3,2]) ⇒ [2,3]
```

Test insufficiency (Cont.)

Wrongs solutions are tested as “correct”!

❌ `common([6,8,1], [6,8,1])  ⇒  [8,1,6]`

```python
def common(l1: list, l2: list) → list:
  """Return sorted unique common elements for two lists"""
  common_elems = list(set(l1).intersection(set(l2)))
  common_elems.sort()
  return list(set(common_elems))
```

list

✅ `common([4,3,2,8], [])` ⇒ `[]`
✅ `common([5,3,2,8], [3,2]) ⇒ [2,3]`

`set` is *unordered*!
`list→set` is NOT order-preserving!
This *luckily* works for HumanEval tests

EvalPlus: Rigorous test generation

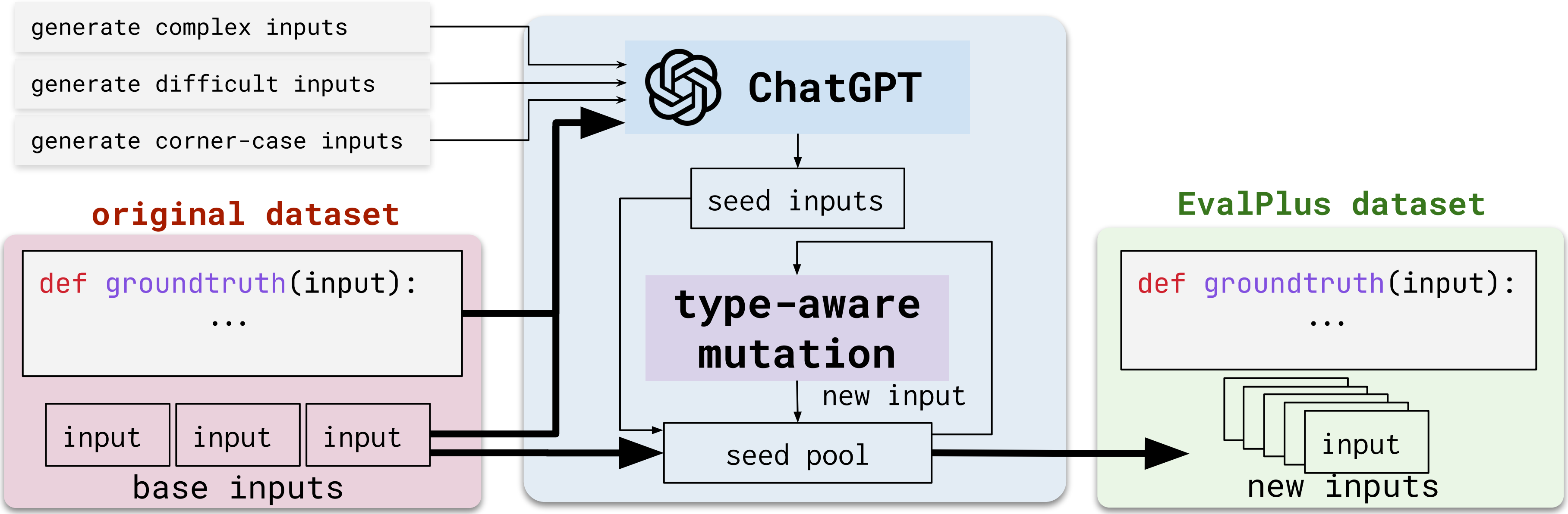
We propose **EvalPlus** to improve test sufficiency via **automated test input generation**

```
// Augment test inputs I to more I
Seeds I ← {inputs from I} ∪ {other inputs}
while budget:
  I ← I ∪ {Mutate(i); i in I}
return I
```

Mutation-based Input Generation

Mutation-based generation

```
Seeds I ← {old tests} U {ChatGPT tests}
while budget:
    I ← I U {TypeMutate(i); i in I}
```



Initial seeds

- Old tests (e.g., from HumanEval)
- **Few-shot prompting ChatGPT to generate tests**

Ground-truth code	`def common(l1: list, l2: list) -> list:`
	`"""Return sorted unique common elements for two lists"""`
	`common_elems = list(set(l1).intersection(set(l2)))`
	`common_elems.sort()`
	`return list(set(common_elems))`

Exemplary test inputs	`Example #1: common([4,3,2,8], [])`
	`Example #2: common([5,3,2,8], [3,2])`
	`Example #3: common([4,3,2,8], [3,2,4])`

Instruction	Can you try to additionally generate corner-case inputs that complies with the input formats of provided examples?

Type-aware mutation

Type assumption for test inputs:

1. **Primitive types**: `bool, int, float, str …`
2. **Compound types**: `List, Tuple, Set, Dict …`

- Design different mutation rules for different types
- Mutate recursively for compound types and `str`

jiawei6@illinois.edu

Mutating primitive type

Type	TypeMutate(x)
int | float	x + 1 or x - 1
bool	A random boolean
str	• Remove/Repeat a substring s • Replace s with **TypeMutate**(s)

jiawei6@illinois.edu

Mutating compound type

Type	**TypeMutate(x)**
List	• Remove/repeat x[i] • Insert/replace x[i] w/ TypeMutate(x[i])
Tuple	Tuple(**TypeMutate**(List(x)))
Set	Set(**TypeMutate**(List(x)))
Dict	• Remove a key-value pair (k, v) • Update (k, v) to (k, **TypeMutate**(v)) • Insert (**TypeMutate**(k), **TypeMutate**(v))

HumanEval+ ← EvalPlus(HumanEval)

- EvalPlus improves HumanEval to **HumanEval+**
- Improving #unique tests by **80x**

Type	Avg #	Medium #	Min #	Max #
HumanEval	9.6	7	1	105
HumanEval+	764.1	982.5	12	1,100

- More tests => more testing time!
- Are these all necessary for exposing wrong solutions?
- **Can we minimize to a set of most representative ones?**

Test-suite reduction

Greedy set covering to only preserve tests with *unique*:

- **Branch coverage**
- Falsified mutants in **mutation testing**
- Identified **wrong LLM code samples**

Type	Avg #	Medium #	Min #	Max #
HumanEval	9.6	7	1	105
HumanEval+	764.1	982.5	12	1,100
HumanEval+^{Mini}	**16.1**	**13.0**	**5**	**110**

Harnessed pass rate on HumanEval+

- Pass@1 drops by up-to **23.1%**
- LLMs like **Phind-CodeLlama** produce more robust code

⚡HumanEval+⚡

	Model	pass rate
#1	GPT4	76.2
#2	**Phind-CodeLlama**	**67.1**
#3	WizardCoder	64.6
#4	ChatGPT	63.4
	…	

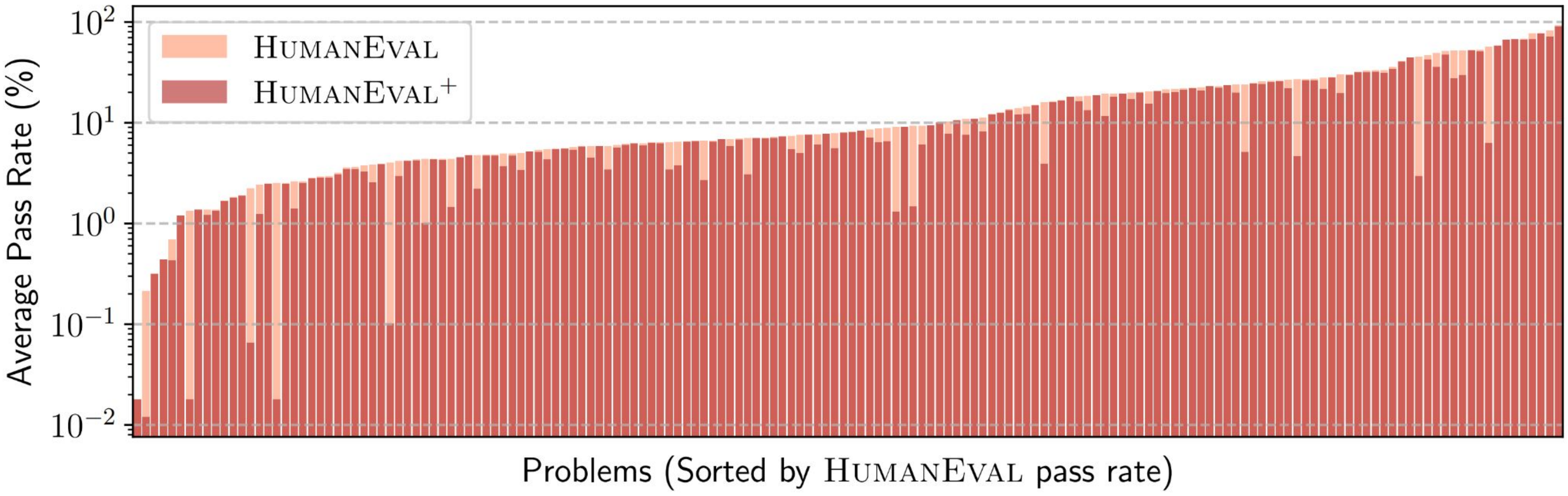
Original HumanEval

	Model	pass rate
#1	GPT4	88.4
#2	ChatGPT	73.2
#3	WizardCoder	73.2
#4	**Phind-CodeLlama**	**71.3**
	…	

Check up-to-date ranking at https://evalplus.github.io/leaderboard.html

Understanding pass rate drop

- Pass rate drops for most problems (156 / 164)
  - **`valid_date`**: mishandling of subtle and deep conditions
  - **`common`**: List->Set->List does not preserve order
  - **`fibfib/fib/fib4`**: slow recursion instead of dynamic programming



Future work of *LLM4Code* evaluation

- **Measuring correctness**
    - Static program verification, e.g., using *Dafny*
    - Runtime verification
    - Automated test generation (e.g., EvalPlus)
- **Measuring beyond correctness**
    - Safety
    - Code quality & linting
    - Performance

Summarizing EvalPlus

- EvalPlus is a technique to improve test sufficiency
- HumanEval+ is created to improve HumanEval
- More supports (e.g., MBPP) are coming

🔥 Using EvalPlus for evaluation is simple and fast!

```python
# Step#1: pip install evalplus
# Step#2: LLM generates its solutions
from evalplus.data import get_human_eval_plus, write_jsonl

samples = [
    dict(task_id=task_id, completion=completion(problem["prompt"]))
    for task_id, problem in get_human_eval_plus().items()
]
write_jsonl("samples.jsonl", samples)
# Step#3: Validate generated solutions on both HumanEval and HumanEval+
#         evalplus.evaluate --dataset humaneval --samples samples.jsonl
```

Online Resource

- [GitHub]
- [PyPI]
- [Leaderboard]
- [Paper]

[Backup] EvalPlus Overview