Evaluating Language Models for Efficient Code Generation

Jiawei Liu, Songrun Xie, Junhao Wang, Yuxiang Wei, Yifeng Ding, Lingming Zhang

University of Illinois Urbana-Champaign *Tongji University*

LLMs can generate code; but can they generate efficient code? How to evaluate that?

TL;DR: Evaluating code efficiency requires *(i)* perf-exercising coding tasks & *(ii)* meaningful compound metric

Simple **tasks** hardly tell a difference...

Tasks like "add two numbers" are mostly solved in the **same** way

Brief computation brings higher **runtime variation**

Simple **tests** hardly tell a difference...

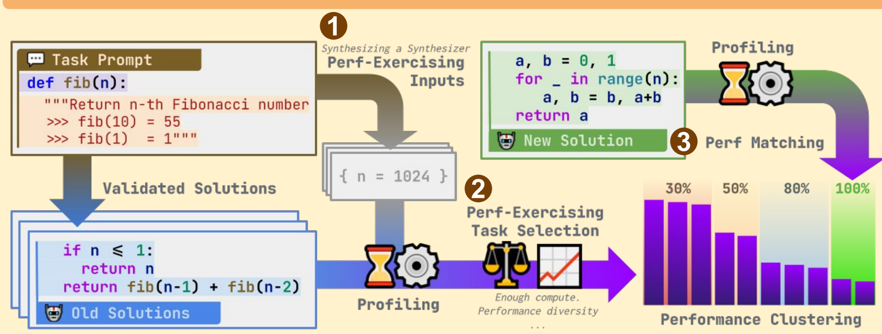
"All complexities are **equal** when *N* is small"

Recursive Fib is **no slower** than iterative Fib with a small N

Average speedup can be confusing...

- Speedup is a commonly used compound metric for computing efficiency
- Speedup is intuitive for **single subject**: flash attn is Nx faster than vanilla attn
- For multiple tasks:
  - *Example:* LLM A is 2x faster on 99 tasks, but LLM B is 100x faster on 1 task
  - *Avg speedup:* Code by LLM B is on average ~1.5x faster than that for LLM A
  - *Inconsistent user experience:* oftentimes code by LLM B is slower...
- Tasks with larger efficiency gaps can skew average speedup, making it biased

Differential Performance Evaluation



❷ Performance-Exercising Task Selection

Selecting performance-exercising coding tasks:

- **Sufficient computation:** the test execution trace should be reasonably long
- **Low variation:** low coefficient of runtime variation to avoid flakiness in the task
- **Performance diversity:** solutions at different levels of efficiency can be sampled

❸ Differential Performance Score

"Your submission can outperform 80% of LLM solutions"...

Relative winning ratio over massive samples of efficiency diversity.

- *Measurement:* #CPU instructions (also generalizable to others)
- *Example:* Given *10* reference samples in *4* clusters: [3, 2, **3**, 2]

If the code efficiency matches the **3rd** cluster:

Differential Performance Score (DPS) = (3 + 2 + 3) / 10 = 80%

Normalized DPS = 3(rd cluster) / 4(clusters) =75%

❶ Synthesizing a Synthesizer (SaS)

- Step 1: **Generating sampling function** "perf_input_gen"
  - *Input:* Scale factor to control the computation load
  - *Expected Output:* Test inputs of corresponding scale
- Step 2: **Exponential input sampling**
  - Sample test inputs using scale= 2^N starting with N = 1
  - Increment N by 1 until hitting testing time budget

Instruction

```
Generate function `perf_input_gen(scale: int)` to produce a "large"
input to exercise the efficiency of the `prime_num` function:
```

Ground-truth Solution

```python
"""Write a function to check if a number is prime"""
import math
def prime_num(num):
    if num < 2: return False
    for i in range(2, math.isqrt(num)):
        if num % i == 0: return False
    return True
```

Chain of Thoughts

```
Analysis:
1. Input format: An integer `n`
2. Time complexity: O(sqrt(n))
3. Space complexity: O(1)
4. What kind of input can exercise its perf? Large prime numbers
```

Input Generator

```python
# can reuse the `prime_num` function
# larger `scale` means larger input
# use case: prime_num(*perf_input_gen(scale))
def perf_input_gen(scale: int):
    for i in range(scale, 2, -1):
        if prime_num(i):  return (i,)
    return (2,)
```

EvalPerf

Using 563 seed tasks in HumanEval+ and MBPP+, we create EvalPerf, a collection of 121 perf-exercising tasks

- Fastest sample requires 10k+ instructions
- Each task has >= 4 reference clusters

Model Study

General instruction tuning boosts code efficiency

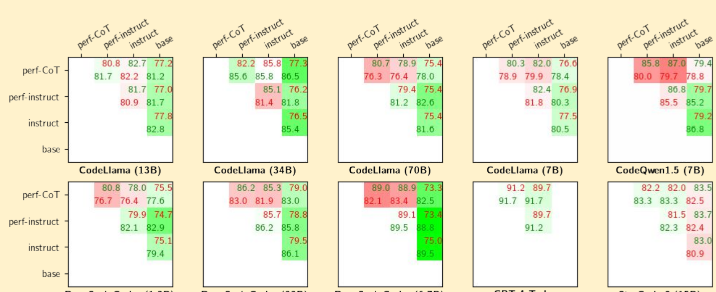
Prior code instruction tuners optimize code correctness, but also helps efficiency

Efficiency-encouraging prompts may not help

"Please provide an efficient ..." and doing CoT does not necessarily help

Larger is not always better...

Within one model series, larger LLMs does not necessarily generate faster code



Methodology Evaluation

More perf-exer tasks than **EvalPlus by 4.8x**

Under the same setting, SaS can get 121 perf-exercising tasks from seed tasks, while EvalPlus test generator can only get 25 perf-exercising tasks.

	>10k instr.	+ Cluster>=4
EvalPlus	204	25
SaS (Ours)	271	121

<= 0.4% cross-platform coeff-variation

By repeating experiments over **4 different test beds**, the coefficient of variation is negligible: 0.1~0.4%.

Contact: https://jw-liu.xyz/

GitHub: part of evalplus/evalplus