Regression Language Models for Code

Yash Akhauri*[1], Xingyou Song*[2], Arissa Wongpanich[2], Bryan Lewandowski[2], Mohamed S. Abdelfattah[1]
[1]Cornell University, [2]Google

Goal

We study code-to-metric regression:

Predicting a numeric outcome of a code execution.

Why challenging?

- Programming languages very open-ended.
- Strictly non-tabular data, very understudied!

Example

Consider the following examples from APPS (Python LeetCode):

Can you predict memory usage precisely?

Problem (Distance Value)

Given two integer arrays `arr1` and `arr2`, and the integer `d`, return the distance value between the two arrays. The distance value is defined as the number of elements `arr1[i]` s.t. there is not any element `arr2[j]` where $|arr1[i] - arr2[j]| \leq d$.

Memory-efficient (O(1) extra space)

```
from typing import List

class Solution:
    def findTheDistanceValue(self, arr1: List[int],
        arr2: List[int], d: int) -> int:
        # O(1) memory.
        count = 0
        for a in arr1:
            far = True
            for b in arr2:
                # No allocations
                if abs(a - b) <= d:
                    far = False
                    # short-circuit
                    break
            if far:
                count += 1
        return count
```

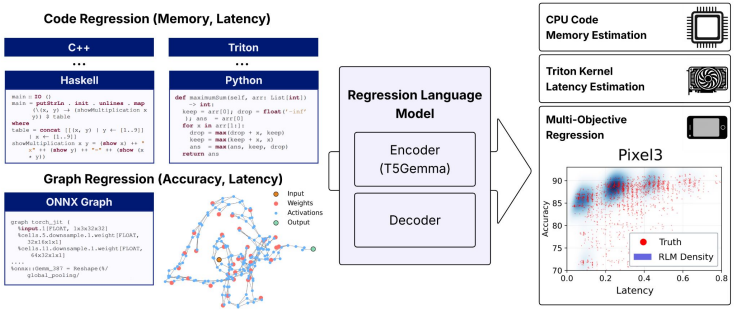
Less memory-efficient (hash structures)

```
from typing import List
from collections import Counter

class Solution:
    def findTheDistanceValue(self, arr1: List[int],
        arr2: List[int], d: int) -> int:
        # overhead: builds dict
        arr1_counts = Counter(arr1)
        # overhead: build hash set
        arr2set     = set(arr2)

        total = 0
        for x in arr1_counts:
            target = range(x - d, x + d + 1)
            # overhead: new set
            if arr2set.intersection(target):
                continue
            total += arr1_counts[x]
        return total
```

Overview

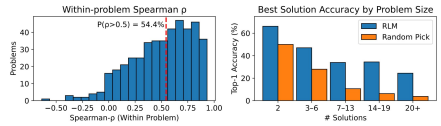


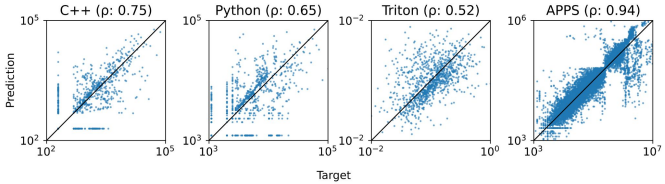
Programming Languages

A single model can predict decently across 24+ programming languages!

- No feature engineering
- No y-normalization

Language	ρ	Language	ρ	Language	ρ	Language	ρ
C++	0.748	Go	0.670	Python	0.647	Kotlin	0.634
C	0.741	D	0.656	OCaml	0.643	Swift	0.630
Lisp	0.625	Lua	0.618	Haskell	0.611	Rust	0.611
Perl	0.592	C#	0.583	Java	0.560	Scala	0.537
Fortran	0.527	TypeScript	0.463	Pascal	0.461	Ruby	0.460
Bash	0.455	F#	0.439	JavaScript	0.395	PHP	0.347
Triton Kernel Latency							0.516
APPS Leetcode Memory							0.930





Computational Graphs

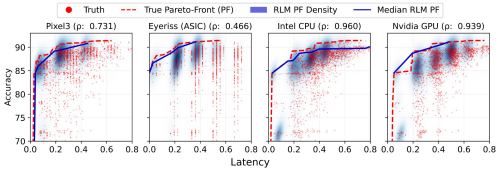
Consider using ONNX Graphs for Neural Architecture Search (NAS):

Matches SoTA Graph Neural Networks (GNNs)!

Method	NASNet	Amoeba	PNAS	ENAS	DARTS	Average
MLP (Adjacency Enc.)	0.002	0.032	0.082	0.021	0.124	0.052
Arch2Vec (Graph Enc.)	0.209	0.107	0.184	0.224	0.333	0.212
CATE (Transformer Enc.)	0.150	0.160	0.217	0.236	0.425	0.238
GNN	0.364	0.376	**0.444**	0.438	0.523	0.429
FLAN (Previous SoTA)	0.344	0.470	0.430	**0.484**	**0.567**	0.459
RLM (Ours)	**0.382**	**0.488**	0.427	0.481	0.528	**0.461**

Multi-Objective Predictions

Decoding twice allows multi-objective predictions across different hardware:



Why Prompting Doesn't Work

Consider in-context learning (ICL) with Gemini.

Each example graph takes 32K tokens; 1M context = 50-shots only.

- Can't absorb data
- Way more expensive
- Not trained for this task

Weight-based updates vs. In-context learning