Measuring Coding Challenge Competence with APPS

Dan Hendrycks* Steven Basart* Saurav Kadavath Mantas Mazeika Akul Arora Ethan Guo
Collin Burns Samir Puranik Horace He Dawn Song Jacob Steinhardt

Introduction

- Benchmarks for natural language understanding and commonsense reasoning are quickly being solved merely by increasing the size of the largest transformer models.
- We introduce APPS, a dataset and benchmark for code generation. APPS focuses on the ability of a model to take problem specifications in natural language and produce runnable, correct Python code.
- We find that after pretraining on ~30GB of GitHub and fine-tuning on the APPS training set, accuracy of GPT-2 1.5B on the APPS test set is 2.1%, indicating much room for improvement.
- However, qualitative analysis of generated code shows that models can output code that seems reasonable at first glance.
- https://github.com/hendrycks/apps

The APPS Dataset

The Automated Programming Progress Standard, abbreviated APPS consists of 10,000 programming exercises in total, with over 100,000 test cases for checking solutions, and over 250,000 ground-truth solutions written by humans. The exercises are split evenly into training and test sets, with 5,000 exercises each.

To measure the correctness of generated solutions, we also collected 102,843 test cases, with a mean of 20.6 test cases per problem and a median of 10 test cases per problem. The average length of a problem across the training and test sets is 279.5 words. Problems were standardized by difficulty into three different levels: “Introductory”, “Interview”, and “Competition”. The test set contains 1,000 introductory problems, 3,000 interview problems, and 1,000 competition problems.

Computational Metrics

To obtain a comprehensive evaluation of code generation ability, we use the large bank of test cases provided with APPS. This enables a rigorous evaluation of generated code via the following metrics.

Accuracy. To compute computational accuracy, we run the code generated by the model on every test case of every exercise. Accuracy is then computed by taking the percentage of problems passing all test cases. This mirrors how humans are evaluated in coding interviews.

Accuracy (w/ Partial Credit). Accuracy with partial credit is computed as the percentage of test cases passed on a problem, averaged across all problems. This is a less stringent metric than standard accuracy and captures our observation that models often implement partial solutions.

Experiments

Pretraining Dataset. In addition to the APPS evaluation dataset, we also collect a large dataset of Python code on GitHub from libraries.io/data. We used this dataset for pretraining models before fine-tuning, then testing, using the APPS dataset.

Model Performance We show Accuracy for all models in **Table 1**. We find that larger models consistently perform better on APPS. The 0.1B parameter model achieves 1.24% accuracy, and the 1.5B parameter model reaches 2.08%. For some problems, models also generate code that passes all test cases. However, absolute accuracies are low across all models, suggesting that solving the APPS benchmark without unreasonable amounts of computational resources may require algorithmic and architectural improvements.

Examples

Problem
You are given a string $s = s_1 s_2 \ldots s_n$ of length n, which only contains digits 1, 2,..., 9. A substring $s[l \ldots r]$ of s is a string $s_l s_{l+1} s_{l+2} \ldots s_r$. A substring $s[l \ldots r]$ of s is called even if the number represented by it is even. Find the number of even substrings of s. Note, that even if some substrings are equal as strings, but have different l and r, they are counted as different substrings. The first line contains an integer n ($1 \le n \le 65000$) — the length of the string s. The second line contains a string s of length n. The string s consists only of digits 1, 2,..., 9. Print the number of even substrings of s.

Model Output

```
n = int(input())
s = input()
ans = 0
for i in range(n):
    for j in range(i, n):
        if int(s[i:j+1]) % 2 == 0:
            ans += 1
print(ans)
```

Problem
You are given a prime number p, n integers $a_1, a_2, \ldots, a_n$, and an integer k. Find the number of pairs of indexes (i, j) ($1 \le i < j \le n$) for which $(a_i + a_j)(a_i^2 + a_j^2) \equiv k \bmod p$. The first line contains integers n, p, k ($2 \le n \le 3 \cdot 10^5$, $2 \le p \le 10^9$, $0 \le k \le p-1$). p is guaranteed to be prime. The second line contains n integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le p-1$). It is guaranteed that all elements are different. Output a single integer answer to the problem.

Model Output

```
n, p, k = map(int, input().split())
a = list(map(int, input().split()))
a.sort()
ans = 0
for i in range(n - 1):
    for j in range(i + 1, n):
        ans += (a[i] + a[j]) % p == k
print(ans)
```

Figure 1: Generated code from GPT-2 1.5B. 18/18 test cases pass for the problem on the left and 6/9 pass on the right. Models were given problems with raw text (including LaTeX code)

Qualitatively, we find that models can

- Reliably parse inputs provided through STDIN in accordance with instructions
- Often generate code that runs without interpreter errors
- Often generate code that superficially relates to the problem content

	CodeTrans	CONCODE	Java Corpus	PY150	APPS
Programming Language	Java/C#	Java	Java	Python	Python
Test Cases	✗	✗	✗	✗	✓
Number of Programs	11,800	104,000	N/A	N/A	260,447
Lines per Program (Avg.)	5.4/6.0	26.3	1	1	21.6
Number of Exercises	11,800	104,000	10,000	3,000	10,000
Text Input	Java/C#	Docstrings	Java	Python	Natural Language

Figure 2: A comparison of the APPS dataset to additional datasets for converting between text and code.

	Accuracy				Partial Credit Accuracy			
Model	Introductory	Interview	Competitive	Average	Introductory	Interview	Competitive	Average
GPT-2 0.1B	2.40	1.27	0.00	1.24	6.28	7.57	3.74	6.55
GPT-2 1.5B	4.80	1.73	0.40	2.08	9.71	9.60	6.52	9.00

Table 1: Accuracy with and without partial credit for the GPT-2 0.1B and 1.5B models. All values are percentages. Accuracy drops off as the difficulty level increases, and increasing model size by an order of magnitude yields a an approximate 2x increase in accuracy.