TACO (Topics in Algorithmic Coding) paper reading

https://arxiv.org/pdf/2312.14852v3.pdf

arxiv.org

https://data.baai.ac.cn/details/BAAI-TACO

Data Hub

模型 数据 EN 登录 TACO 一个专注于算法的代码生成数据集 自然语言处理 代码生成 57 683 2023-12-22更新 数据集介绍 数据集文件 ## 数据简介 TACO(Topics in Algorithmic COde generation dataset)是一个专注于算法的代码生成数据集，旨在为代码生成…

We propose TACO, an open-source **large-scale code generation dataset** that focuses on **algorithmic topics** , aiming to provide more challenging training datasets and evaluation benchmark test sets to evaluate code generative models. TACO includes **more challenging programming competition-level problems** to enhance or evaluate problem understanding and reasoning abilities in practical programming scenarios. The training dataset and test set contain 25,433 and 1000 coding problems, respectively, as well as up to 1.55 million various solution answers. In addition, **each TACO problem contains several fine grain labels** , such as task topic, algorithm, programming skills, and difficulty level, providing more accurate references for training and evaluating code generative models. The dataset and evaluation script are available on Hugging Face Hub and Github .

数据格式

字段	类型	描述
question	string	problem description
solutions	string	some python solutions
input_output	string	Json string with “inputs” and “outputs” of the test cases, might also include “fn_name” the name of the function
difficulty	string	difficulty level of the problem
picture_num	string	the number of pictures in the problem
source	string	the source of the problem
url	string	url of the source of the problem
date	string	the date of the problem
starter_code	string	starter code to include in prompts
time_limit	string	the time consumption limit to solve the problem
memory_limit	string	the memory consumption limit to solve the problem
Expected Auxiliary Space	string	the extra auxiliary space expected to solve the problem
Expected Time Complexity	string	the time complexity expected to solve the problem
raw_tags	string	the topics of the programming task
tags	string	the manually annoatated algorithms needed to solve the problem
skill_types	string	the mapped programming skill types to solve the problem

样例


```
{
  "question": "You have a deck of $n$ cards, and you'd like to reorder it to a new one.\n\nEach card has a value between $1$ and $n$ equal to $p_i$
  "solutions": [
    "import heapq\nfrom math import sqrt\nimport operator\nimport sys\ninf_var = 0\nif inf_var == 1:\n\tinf = open('input.txt', 'r')\nelse:\n\tinf
    "t = int(input())\nfor _ in range(t):\n\tn = int(input())\n\tp = list(map(int, input().split()))\n\tans = []\n\tp1 = [-1] * (n + 1)\n\tfor i in
    "import sys\n\ndef get_ints():\n\treturn map(int, sys.stdin.readline().strip().split())\n\ndef get_list():\n\treturn list(map(int, sys.stdin.re
    ...
  ],
  "starter_code": "",
  "input_output": {
    "inputs": [
      "4\n4\n1 2 3 4\n5\n1 5 2 4 3\n6\n4 2 5 3 6 1\n1\n1\n",
      "4\n4\n2 1 3 4\n5\n1 5 2 4 3\n6\n4 2 5 3 6 1\n1\n1\n",
      "4\n4\n2 1 3 4\n5\n1 5 2 4 3\n6\n2 4 5 3 6 1\n1\n1\n",
      "4\n4\n1 2 3 4\n5\n1 5 2 4 3\n6\n4 2 5 3 6 1\n1\n1\n"
    ],
    "outputs": [
      "4 3 2 1\n5 2 4 3 1\n6 1 5 3 4 2\n1\n",
      "4 3 2 1\n5 2 4 3 1\n6 1 5 3 4 2\n1\n",
      "4 3 2 1\n5 2 4 3 1\n6 1 5 3 4 2\n1\n",
      "\n4 3 2 1\n5 2 4 3 1\n6 1 5 3 4 2\n1\n"
    ]
  },
  "difficulty": "EASY",
  "raw_tags": [
    "data structures",
    "greedy",
    "math"
  ],
  "name": null,
  "source": "codeforces",
  "tags": [
    "Data structures",
    "Mathematics",
    "Greedy algorithms"
  ],
  "skill_types": [
    "Data structures",
    "Greedy algorithms"
  ],
  "url": "https://codeforces.com/problemset/problem/1492/B",
  "Expected Auxiliary Space": null,
  "time_limit": "1 second",
  "date": "2021-02-23",
  "picture_num": "0",
  "memory_limit": "512 megabytes",
  "Expected Time Complexity": null
}
```


Research background

Code ability is one of the core abilities of the basic model, which is crucial for improving key skills such as reasoning and planning. With the rapid development of large language models and code generative models, mainstream code evaluation benchmarks have shown their limitations, making it difficult to fully reflect the performance and potential of the model in real scenarios.

The difficulty of the evaluation task is low, and it is impossible to comprehensively evaluate the ability of the model

The current mainstream HumanEval/HumanEval-X and MBPP/MBXP evaluation sets mainly focus on basic programming problems, requiring the model to complete a certain function through code completion or Text2Code tasks, rather than solving a real-world problem. In addition, the model performance on each evaluation benchmark has reached a high score, such as **HumanEval's SOTA model performance is 94.4 (pass@1), MBPP's SOTA model performance is 81.1 (Acc)** , and the reference value of the evaluation results gradually decreases [1].

There are problems with the quality of the test set, and the validity is questionable

Code execution and test cases are the key to checking whether the code is correct, and **APPS, CodeContest evaluation benchmarks also have problems such as test sets without artificial answers, questions and answer sets before deduplication** . In addition, DeepMind published a paper "Competition-Level Code Generation with AlphaCode" pointing out that due to the **shortage of test cases** , the code evaluation dataset has the problem of False Positive, that is, all test cases pass, but after manual inspection, it is found that the code implementation is incorrect, and the model only passes the given few test cases.

Lack of fine grain indicators

The current code evaluation dataset lacks more fine-grained indicators for coding ability evaluation, such as **in difficulty dimension, algorithm dimension** (sorting, Dynamic Programming, etc.), which cannot provide targeted guidance for improving model code generation ability. Therefore, we need a more challenging, higher quality, and fine-grained code generation evaluation scheme to evaluate the model's code generation ability and provide guidance for improving code generation ability.

TACO: More challenging code generation training datasets and evaluation benchmarks

TACO (Topics in Algorithmic COde generation dataset) is a code generation dataset that focuses on algorithms, aiming to provide a more challenging training dataset and evaluation benchmark for the field of code generative models. The dataset contains programming competition questions that are more difficult and closer to real programming scenarios, emphasizing the improvement or evaluation of the model's understanding and reasoning ability in practical application scenarios, rather than just implementing established function functions.

- **Larger scale** : TACO includes a training dataset (25,443 questions) and a test set (1,000 questions), and is currently the largest code generation dataset.
- **Higher quality** : Each question in the TACO dataset matches as many diverse answers as possible, with an answer size of up to 1.55 million, ensuring that the model is not easily overfitting during training and the effectiveness of evaluation results.
- **Provide fine grain labels** : Each question in the TACO dataset contains fine grain labels such as task theme, algorithm, skill, and difficulty, which provide a more accurate reference for the training and evaluation of code generative models.
- **Test cases are more comprehensive:** According to statistics, APPS and CodeContent test sets have about 25% of the answers without manual verification. In addition, according to the paper published by DeepMind [2], when the proportion of "test cases/problems (Tests/Problems) " is greater than 200, the False Positive problem will be significantly reduced, which can better ensure the validity of the test results. The proportion of "test cases/problems (Tests/Problems) " in the TACO test set is 202.3.

对比维度		TACO	CodeContest	APPS	HumanEval(/-X)	MBP(/X)P
数据集规模	问题规模(train/dev/test)	25443/-/1000	13328/117/165	5000/-/5000	-/-/164	374/-/500
	答案规模	1.55M	1.5M	0.23M	1*164	1*(374+500)
数据集质量	测试集无答案数	0	43/165	1235/5000	0	0
	问题去重	去重	无去重	无去重	去重	去重
	答案去重	去重	无去重	无去重	去重	去重
	测试用例/问题(Tests/Problem)	202.3	203.7	20.99	7.77	3
细粒度标签	任务主题	有	有	无	无	无
	算法标签	有	无	无	无	无
	编程技能	有	无	无	无	无
	难度标签	有	有	有	无	无

Summary

TACO has expanded new sources of competition application questions, divided into training dataset (25,443 questions) and test set (1000 questions), which is currently the largest code generation dataset. Compared with the evaluation benchmark APPS and CodeContent, which use a large number of competition application questions, TACO has increased more than 29.5% of new questions and answers.

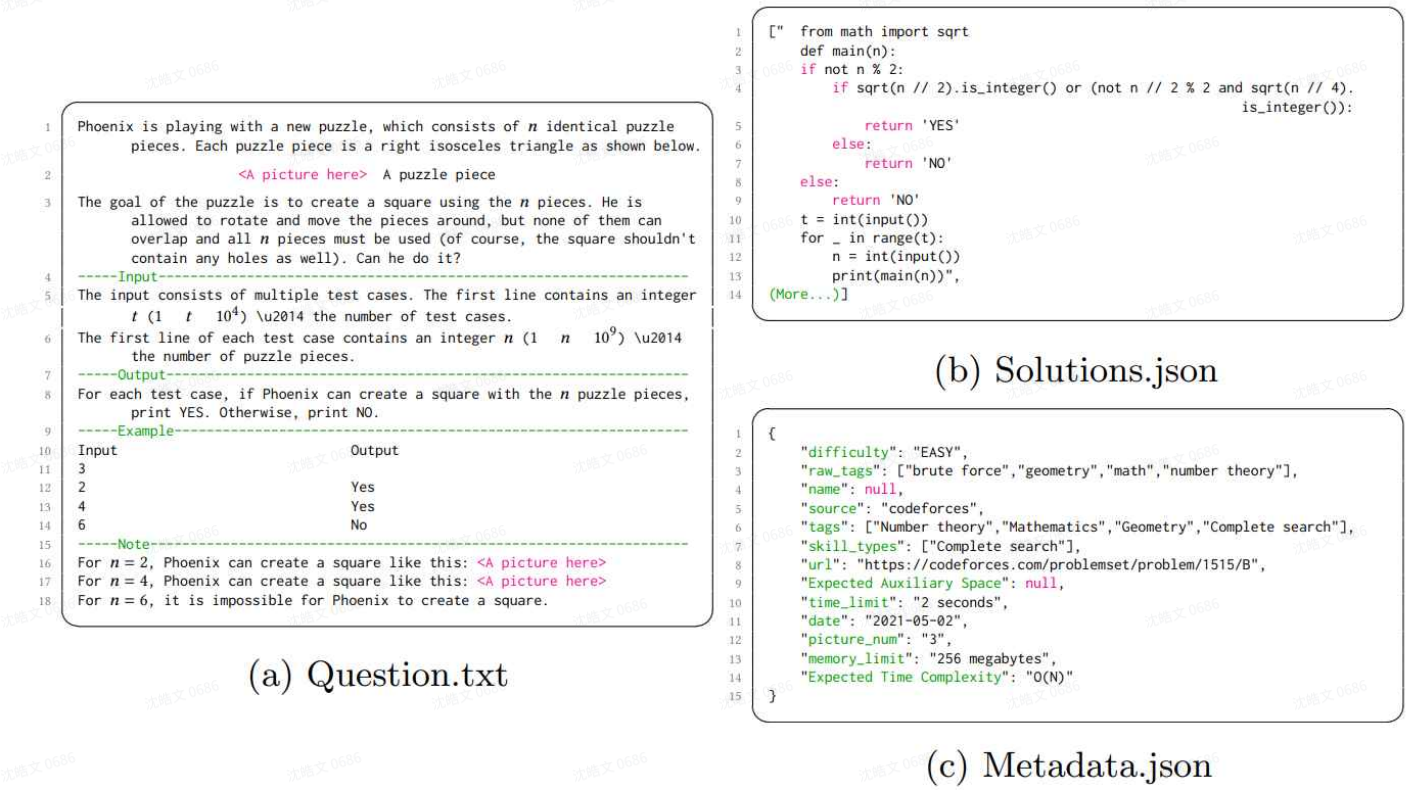

Figure 1: An example of a classical algorithmic problem (e.g., quertion, solutions, and lables)

· More Extensive: The TACO dataset contains 26,443 questions (25,443 questions in the training dataset and 1,000 questions in the test set) and 1,539,152 validated Python 3 code solutions. Initially, we accumulated 2,045,502 code examples, but this number was reduced after a rigorous code duplication data removal process. **On average, there are 58.21 correct code solutions per question** . (There are 51.6 test cases per question in the training set and 202.3 in the test set.)

Algorithmic Categorization: The TACO dataset emphasizes algorithmic labeling of problems encountered in programming challenges. For all problems, we thoroughly summarized the algorithm labels of 968 categories and ultimately merged them into 36 different algorithm categories.

· Data Quality Verification: We manually evaluated the correctness of the data subset collected from the TACO dataset. To ensure the accuracy of the test cases and solutions in the dataset, we conducted a comprehensive verification using unit tests.

Performance Benchmarks: The benchmark we propose is mainly used to evaluate the code generation efficiency of the nl2code model. In the test set, we provide detailed coverage of different algorithms and problems, aiming to measure the performance of the model in a series of algorithmic skills that are crucial for problem-solving.

Dataset construction

The development of the TACO dataset was divided into two distinct phases: data collection and data processing.

Data collection

- We mainly collect original data sources from various programming competition platforms. This stage further enriches and integrates other open source datasets, thereby expanding the scope and diversity of datasets.
- Our main data sources are platforms such as CodeChef, CodeForces, HackerRank, and GeeksforGeeks. We also integrate existing datasets, including APPS, CodeContest, and Description2code. To build and optimize the TACO dataset, we use two main methods: first, we use web scraping technology to extract problem information from programming competition websites; second, we use algorithm skill tags and other related tags to enhance existing open source datasets.

29.5% of the data was crawled by ourselves, and the rest came from APPS and CodeContest.

> **APPS:** The APPS dataset primarily originates from programming competition platforms, including CodeForces, LeetCode, and CodeChef, among others. Both the training and test sets of this dataset comprise 5,000 problems. Notably, existing academic literature, particularly the study conducted by Alphacode, highlights that the APPS dataset employs only samples drawn from problem descriptions to serve as the HIDDEN TEST CASE, leading to an elevated False Positive Rate.
>
> **CodeContests:** The CodeContests dataset encompasses an extensive array of popular programming languages, such as C++, C#, Go, Java, JavaScript, Lua, PHP, Python, Ruby, Rust, Scala, and TypeScript. The dataset is partitioned into training, validation, and test sets, containing 13,328, 117, and 165 problems, respectively. In contrast to datasets that exclusively feature correct code, CodeContests incorporates a variety of erroneous code types.

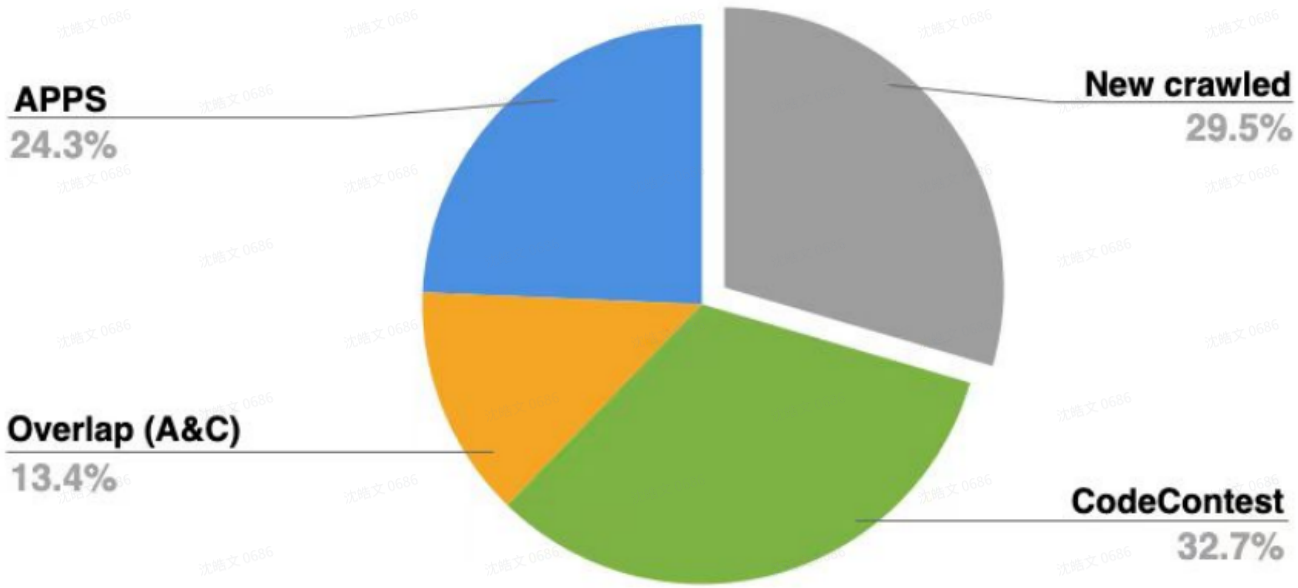

Figure 2: Analysis of TACO dataset composition

Data processing

- This stage requires code annotation and deduplication to ensure data quality and uniqueness. Subsequently, we added test samples to enhance the robustness of the dataset. At this stage, we also classified and summarized tasks and skill sets, using insights obtained from various programming competition websites and open source code to generate datasets.
- **Increase test cases** : Specifically, we use OpenAI's GPT-4 API to generate input components for unit test pairs. Subsequently, 30 verified code samples are executed on these inputs to generate corresponding outputs, and then the consistency of the outputs is checked. Input-output pairs that produce consistent outputs are considered valid unit tests. This iterative process is performed multiple times to ensure that the total number of unit tests reaches the minimum threshold of 200.

- **Fine grain tags:** In the source programming problem dataset, we retain 968 category original tags that span various domains. These tags are manually annotated by domain experts and provide the most refined classification of information. Specifically, these tags can encapsulate the essence of the topic problem (such as mathematics, geometry, diagrams, or strings) or feasible solution strategies (such as Dynamic Programming, brute force methods, or binary search). It is worth noting that a problem may have multiple such tags at the same time. Our research results indicate that an excessive and complex array of algorithm tags may hinder the training and inference process of the model. Therefore, we thoroughly generalize the initial set of 968 category algorithm tags and ultimately reclassify all tags into 36 discrete categories.

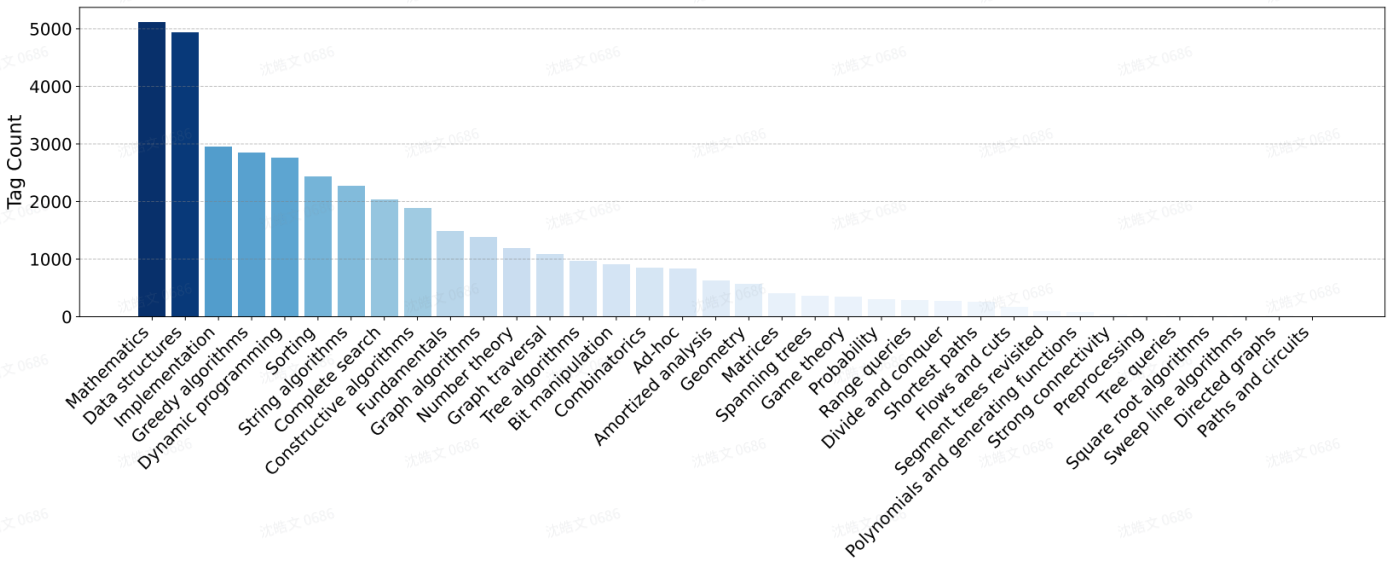

Figure 4: Statistical distribution of 36 algorithmic labels obtained by generalization on the dataset

Table 4: Distribution of difficulties and programming skills in TACO training and test dataset

Category	Type	Train	Test
Difficulty	Easy	8904	200
	Medium	3244	200
	Medium_Hard	2745	200
	Hard	3162	200
	Very_Hard	2374	200
	Unknown	5014	0
Skill	Range Queries	220	73
	Amortized Analysis	554	70
	Bit Manipulation	814	90
	Complete Search	1886	154
	Sorting	2234	197
	Dynamic Programming	2585	179
	Greedy Algorithms	2649	209
	Data Structures	4695	241
	Unknown	14907	337

Experimental results

Using pass@k (k = 1,10,100) as the evaluation metric. Through extensive generative experiments, we found that the model is highly sensitive to difficulty. For more challenging problems, the model requires more diverse generative results to generate code that passes the test. Therefore, in the testing process, we use top_p 0.95. Temperature is differentiated according to the difficulty level of the problem: we assign a value of 0.2 for simple problems, 0.6 for medium problems, and 0.8 for difficult and very difficult problems.

Table 5: Performances of code generations on TACO coding problems of different difficulty levels

Model Name	Level	Temperature	Pass@1	Pass@10	Pass@100
GPT-4	easy	0.7	31.50	-	-
	medium	0.7	19.00	-	-
	medium_hard	0.7	13.00	-	-
	hard	0.7	4.50	-	-
	very_hard	0.7	2.00	-	-
codellama-7b-python	easy	0.2	9.32	15.15	19.99
	medium	0.6	2.38	8.28	17.51
	medium_hard	0.6	0.60	2.93	7.53
	hard	0.8	0.31	1.93	5.51
	very_hard	0.8	0.18	1.05	2.25
codegen25-mono	easy	0.2	9.06	16.12	22.52
	medium	0.6	2.42	7.83	17.23
	medium_hard	0.6	0.74	3.58	8.01
	hard	0.8	0.57	2.66	5.31
	very_hard	0.8	0.22	1.00	1.47
starcoder-1b	easy	0.2	8.95	12.01	15.51
	medium	0.6	2.22	5.32	11.16
	medium_hard	0.6	0.60	2.74	6.11
	hard	0.8	0.20	1.29	3.44
	very_hard	0.8	0.04	0.32	1.13
starcoder-15.5b	easy	0.2	11.60	18.44	23.92
	medium	0.6	3.23	8.96	19.32
	medium_hard	0.6	1.43	5.43	9.41
	hard	0.8	0.58	2.79	6.39
	very_hard	0.8	0.18	0.85	1.75

In order to further explore the impact of different programming skills and the role of TACO training dataset, we conducted experiments on the starcoder-1b model. We used the starcoder-1b model to fine-tune the entire training dataset with LoRA and obtained a base line LoRA model. In addition, we also fine-tuned the training dataset subsets corresponding to different programming skills with LoRA, generating 8 LoRA models for different skills.

Firstly, GPT-4 shows considerable ability in various programming skills, with pass@1 scores between 5 and 10, which to some extent indicates the rationality of skill classification in the TACO dataset. Secondly, **Using TACO training datasets with fine grain labels can specifically enhance the performance of code generative models** For example, in Data Structures, Greedy Algorithms, and Sorting, starcoder-1b shows significant performance advantages when using TACO training datasets to fine-tune specific skills compared to fine-tuning the entire training dataset.

Table 6: Evaluation Results for Different Skill Types

SKILL TYPES	GPT-4	baseline LoRA	skilled LoRAs
Amortized Analysis	11.43	0	0
Bit Manipulation	10.00	1.1	1.11
Complete Search	14.29	0.32	0.32
Data Structures	13.28	1.4	**1.55**
Dynamic Programming	8.94	0	0
Greedy Algorithms	9.09	0	**0.36**
Range Queries	5.48	0	0
Sorting	10.66	0.7	**1.5**