Textbooks Are All You Need

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, Yuanzhi Li

Microsoft Research 2023

Presented by Jack Sun and Quentin Clark

Introduction

- Variant of Scaling Law
  - higher **quality** of the dataset -> better result (Eldan and Li, 2023)
- This work aims to show that high quality data can:
  - improve the SOTA of large language models
  - reduces the dataset size and training compute.
- **Contribution:**
  - Trained a tiny and competitive LLM for Python code generation by data filtering techniques.
  - Outperform other coder LLM that are trained on larger dataset with more parameters.

Data Filtering

- **Data Sources:** Deduplicated Python files in The Stack and the StackOverflow ~35B tokens, ~35M code examples)
- Annotate the quality of ~100K samples via asking GPT-4 (Prompt: "determine the educational value of a code snippet.")
  - Minimizes human-annotation efforts
- Train a random forest classifier to filter out low-quality data.
- Result: a 6B high-quality dataset "The Stack"
- Significant reduction in size.

Educational values deemed by the filter

High educational value

```
import torch
import torch.nn.functional as F

def normalize(x, axis=-1):
    """Performs L2-Norm."""
    num = x
    denom = torch.norm(x, 2, axis, keepdim=True)
    .expand_as(x) + 1e-12
    return num / denom

def euclidean_dist(x, y):
    """Computes Euclidean distance."""
    m, n = x.size(0), y.size(0)
    xx = torch.pow(x, 2).sum(1, keepdim=True).
    expand(m, n)
    yy = torch.pow(x, 2).sum(1, keepdim=True).
    expand(m, m).t()
    dist = xx + yy - 2 * torch.matmul(x, y.t())
    dist = dist.clamp(min=1e-12).sqrt()
    return dist

def cosine_dist(x, y):
    """Computes Cosine Distance."""
    x = F.normalize(x, dim=1)
    y = F.normalize(y, dim=1)
    dist = 2 - 2 * torch.mm(x, y.t())
    return dist
```

Low educational value

```
import re
import typing
...

class Default(object):
    def __init__(self, vim: Nvim) -> None:
        self._vim = vim
        self._denite: typing.Optional[SyncParent]
    = None
        self._selected_candidates: typing.List[int
    ] = []
        self._candidates: Candidates = []
        self._cursor = 0
        self._entire_len = 0
        self._result: typing.List[typing.Any] = []
        self._context: UserContext = {}
        self._bufnr = -1
        self._winid = -1
        self._winrestcmd = ''
        self._initialized = False
        self._winheight = 0
        self._winwidth = 0
        self._winminheight = -1
        self._is_multi = False
        self._is_async = False
        self._matched_pattern = ''
        ...
```

Data Generation

- **Method:** Prompt GPT-3.5 to write code snippet.
- **Challenge:** ensuring that the generated dataset is diverse and non-repetitive.
  - Trick: inject randomness into the prompt regarding by providing constraints on vocabulary, topics and target audience of the generated textbook.
- This creates a ~1B dataset called "CodeTextbook".

```
To begin, let us define singular and nonsingular matrices. A matrix is said to be singular if its
determinant is zero. On the other hand, a matrix is said to be nonsingular if its determinant is not
 zero. Now, let's explore these concepts through examples.

Example 1: Consider the matrix A = np.array([[1, 2], [2, 4]]). We can check if this matrix is
singular or nonsingular using the determinant function. We can define a Python function, `
is_singular(A)`, which  returns true if the determinant of A is zero, and false otherwise.

import numpy as np
def is_singular(A):
    det = np.linalg.det(A)
    if det == 0:
        return True
    else:
        return False

A = np.array([[1, 2], [2, 4]])
print(is_singular(A)) # True
```

Data Generation for alignment

- A small synthetic exercises dataset CodeExercise (~180M tokens).
  - Each exercise is a docstring of a function that needs to be completed to
  - Align the model to perform function completion in the fine-tuning stage.
  - This dataset was generated by GPT-3.5, where the main means of eliciting diversity is by constraining the function names.

```python
def valid_guessing_letters(word: str, guesses: List[str]) -> List[str]:
    """
    Returns a list of valid guessing letters, which are letters that have not been guessed yet and
    are present in the word.
    Parameters:
    word (str): The word to guess.
    guesses (List[str]): A list of letters that have already been guessed.
    Returns:
    List[str]: A list of valid guessing letters.
    """
    valid_letters = []
    for letter in word:
        if letter not in guesses and letter not in valid_letters:
            valid_letters.append(letter)
    return valid_letters
```

Model Architecture

- **Model Overview:**
  - Decoder-only Transformer with 1.3B parameters.
  - FlashAttention for efficient multi-head attention.
- Parameters:
  - Batch size: 1024 for pretraining, 256 for finetuning.
- Hardware: 8 A100 GPUs
  - <4 days for pretraining
  - 7 hours for finetuning.

phi-1 and Its Variants

- **phi-1-base:** Pretrained on “The Stack” + “CodeTextbook” dataset for 8 passes.
- **phi-1:** Also finetuned on “CodeExercises”.
- Achieved 50.6% pass@1 accuracy on HumanEval, 55.5% on MBPP.
  - HumanEval: 164 programs with 8 tests for each.
  - MBPP (Mostly Basic Python Programming: 1000 programs, 3 tests for each.
  - pass@1: check whether its initial code generation is correct.
- **phi-1-small:** 350M-parameter variant.
- **Emergent Properties:** Finetuning improves logical reasoning and library usage.
- Demonstrates scaling laws for performance improvements with quality data.

Results: Code Benchmark Evaluation

Date	Model	Model size (Parameters)	Dataset size (Tokens)	HumanEval (Pass@1)	MBPP (Pass@1)
2021 Jul	Codex-300M [CTJ+21]	300M	100B	13.2%	-
2021 Jul	Codex-12B [CTJ+21]	12B	100B	28.8%	-
2022 Mar	CodeGen-Mono-350M [NPH+23]	350M	577B	12.8%	-
2022 Mar	CodeGen-Mono-16.1B [NPH+23]	16.1B	577B	29.3%	35.3%
2022 Apr	PaLM-Coder [CND+22]	540B	780B	35.9%	47.0%
2022 Sep	CodeGeeX [ZXZ+23]	13B	850B	22.9%	24.4%
2022 Nov	GPT-3.5 [Ope23]	175B	N.A.	47%	-
2022 Dec	SantaCoder [ALK+23]	1.1B	236B	14.0%	35.0%
2023 Mar	GPT-4 [Ope23]	N.A.	N.A.	67%	-
2023 Apr	Replit [Rep23]	2.7B	525B	21.9%	-
2023 Apr	Replit-Finetuned [Rep23]	2.7B	525B	30.5%	-
2023 May	CodeGen2-1B [NHX+23]	1B	N.A.	10.3%	-
2023 May	CodeGen2-7B [NHX+23]	7B	N.A.	19.1%	-
2023 May	StarCoder [LAZ+23]	15.5B	1T	33.6%	52.7%
2023 May	StarCoder-Prompted [LAZ+23]	15.5B	1T	40.8%	49.5%
2023 May	PaLM 2-S [ADF+23]	N.A.	N.A.	37.6%	50.0%
2023 May	CodeT5+ [WLG+23]	2B	52B	24.2%	-
2023 May	CodeT5+ [WLG+23]	16B	52B	30.9%	-
2023 May	InstructCodeT5+ [WLG+23]	16B	52B	35.0%	-
2023 Jun	WizardCoder [LXZ+23]	16B	1T	57.3%	51.8%
2023 Jun	**phi-1**	1.3B	7B	50.6%	55.5%

Their Model

Observe:
- ~ 1 OOM less data than CodeT5 models, twice as good
- ~ 2 OOM less data than Code Gen, almost twice as good
- Only competitive models are WizardCoder and GPT-4, which have vastly larger models and data

Results: Similarity Pruning

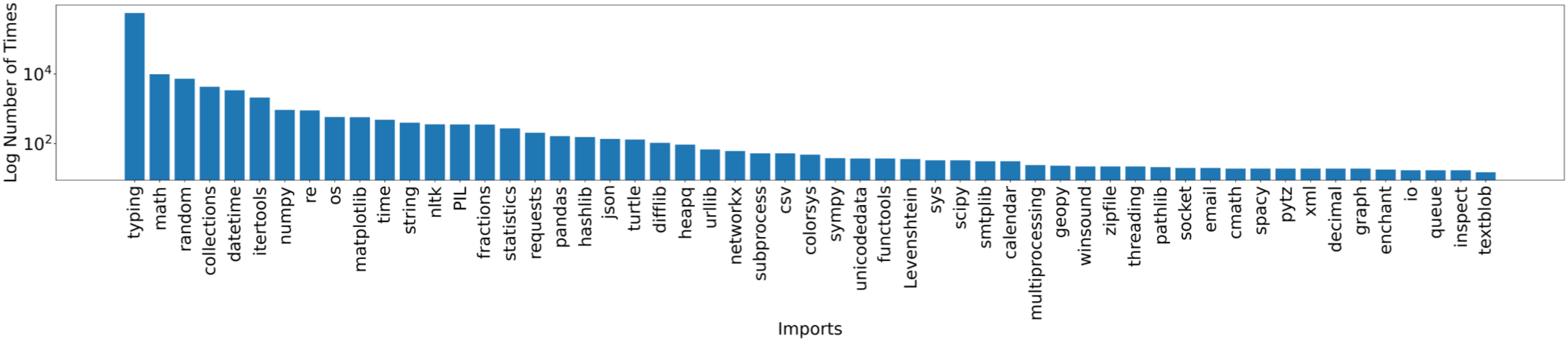
Removing More Data

τ		Problem Count	**phi-1**	**phi-1 retrained on pruned data**	StarCoder-Prompted [LAZ+23]
0.95	similar	71	81.7%	74.6%	57.7%
	non-similar	93	26.9%	32.3%	29.0%
	total	164	50.6%	50.6%	41.5%
0.9	similar	93	63.4%	51.6%	48.4%
	non-similar	71	33.8%	36.6%	32.4%
	total	164	50.6%	45.1%	41.5%
0.85	similar	106	62.3%	52.8%	47.2%
	non-similar	58	29.3%	34.5%	31.0%
	total	164	50.6%	46.3%	41.5%
0.8	similar	116	59.5%	52.6%	45.7%
	non-similar	48	29.2%	27.1%	31.2%
	total	164	50.6%	45.1%	41.5%

Results: Emergence Properties

1] General understanding improves

2] Use of libraries **not in the fine-tuning dataset** increases and improves



Critical Response

ICLR 24 Reviews: https://openreview.net/forum?id=Fq8tKtjACC

Main critiques:

1] Data leakage possible in main CodeTextbook (even though not present in fine-tuning CodeExercices) dataset

2] Limited eval

3] Ambiguity in the data generation process (i.e., how do they ensure data diversity?) apparently motivated by propriety

References

- Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? arXiv preprint arXiv:2305.07759, 2023.
- Hestness, Joel, et al. "Deep learning scaling is predictable, empirically." *arXiv preprint arXiv:1712.00409* (2017).
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311, 2022.
- Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. ICLR, 2023.