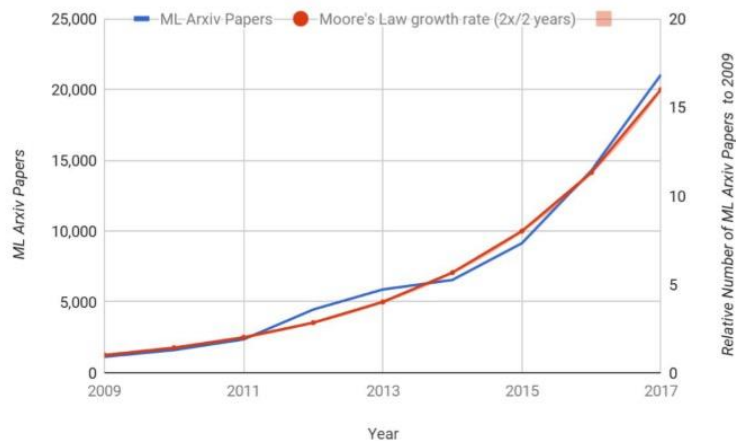


# Textbooks Are All You Need

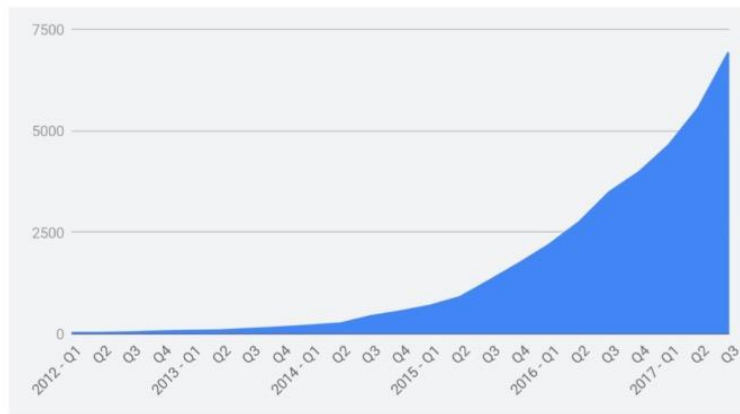
Nicolas Jorquera

Microsoft Research

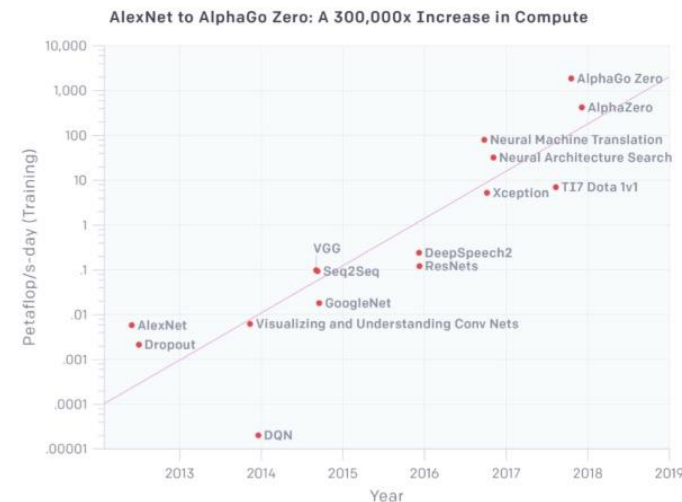
# Exponential Growth in Deep Learning



ArXiv papers about ML  
~18 months

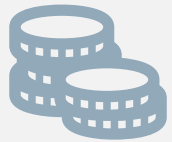


Google project directories  
~18 months



FLOPs to train a model  
~3.5 months

# Introduction



The computational and environmental costs of scaling up models




The inefficiencies of learning from standard code datasets

# Challenges

# Proposed Solution

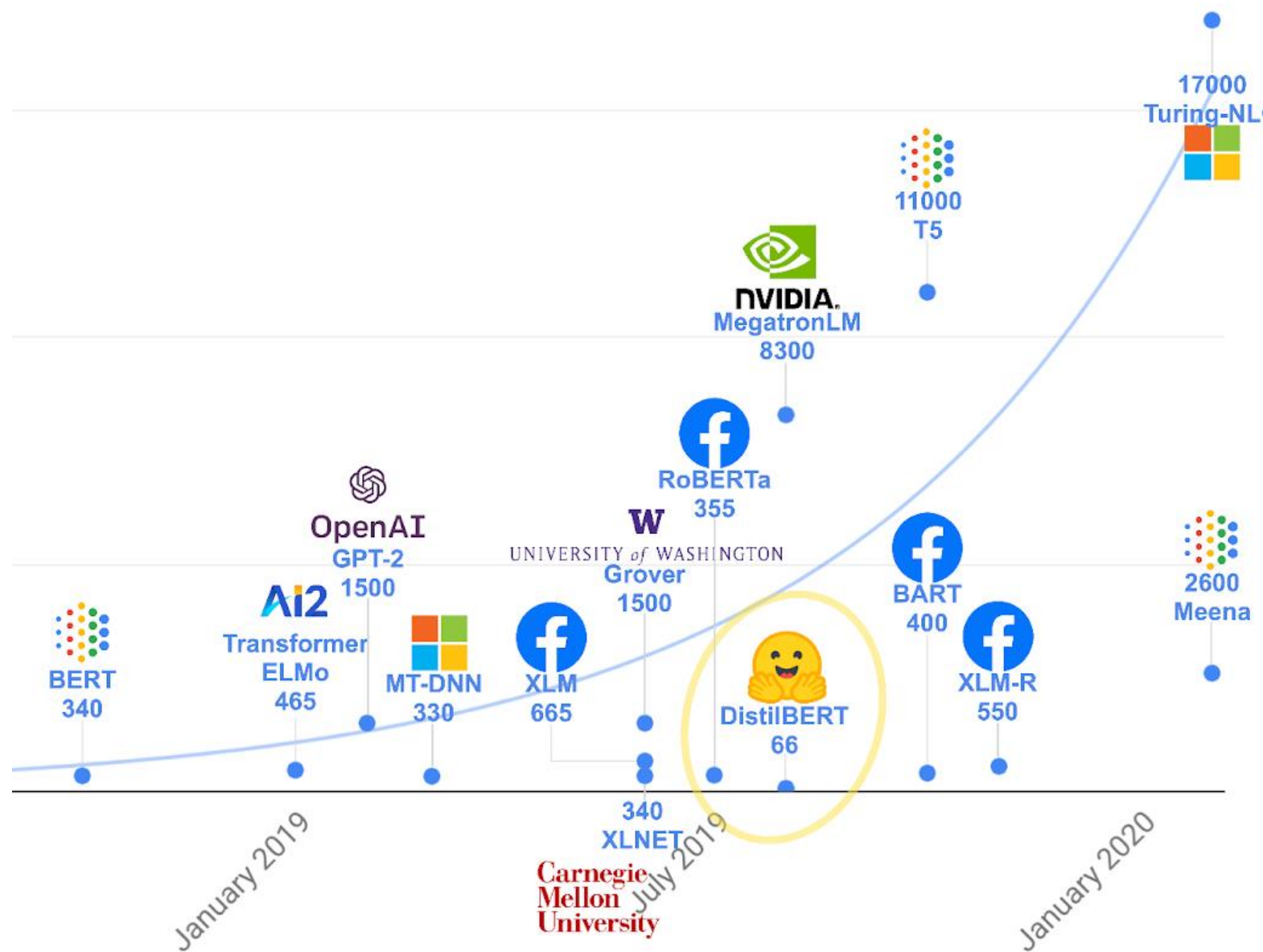
The study challenges the prevailing notion in the AI community that bigger models and more data always lead to better performance.



Instead, it emphasizes the profound impact of the quality of training data, especially in the context of code generation.

# Scaling Laws

- NLP model size and computation have been increasing exponentially



# Datasets – Traditional Approaches

Standard datasets:  
The Stack and  
StackOverflow

Limitations: Noise,  
ambiguity, and lack  
of instructional  
quality

# CodeTextbook



The idea behind "textbook quality" data



Characteristics: Clarity, self-contained, instructive, balanced



Potential benefits for model training

The following example demonstrates the synthetically generated textbook text:

```
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
```

*This dataset consists of less than 1B tokens of GPT-3.5 generated Python textbooks, synthesized to provide a high-quality source of natural language heavy text interleaved with relevant code snippets.*



# CodeExercises



Dataset of coding exercises



Purpose: Fine-tuning and specialization



Emphasis on high-quality coding tasks

The following example demonstrates the synthetically generated textbook text:

```
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
```

*This is a small synthetic exercises dataset consisting of less than 180M tokens of Python exercises and solutions. Each exercise is a docstring of a function that needs to be completed. The goal of this dataset is to align the model to perform function completion tasks based on natural language instructions.*



Baseline: Model trained  
on standard datasets



phi-1 model



The smaller counterpart:  
phi-1-small

## Model Overview

## Model trained on the standard dataset

- Parameters: Not explicitly mentioned, but it's compared with the phi-1 model.
- Training Data: Trained on the standard dataset of deduplicated Python files from The Stack (plus StackOverflow for the 1.3B parameter model).
- Performance: The performance of this model is used as a baseline to demonstrate the effectiveness of the phi-1 and phi-1-small models.



## **phi-1**

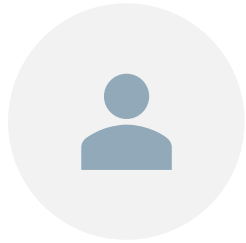
- Parameters: 1.3 billion parameters.
- Training Data: Initially trained on the CodeTextbook dataset and then fine-tuned on the CodeExercises dataset.
- Performance: Achieves a performance of 51% on the HumanEval benchmark.

## **phi-1 – small**

- Parameters: 350 million parameters.
- Training Data: Trained with the same pipeline as phi-1.
- Performance: Achieves a performance of 45% on the HumanEval benchmark [Page 1].



# Model Architecture



Transformer  
Architecture



Flash Attention



Sequence Length  
of 2048

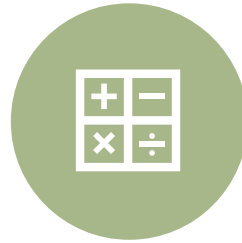


Next Token  
Prediction Loss

# Model Architecture



AdamW Optimizer



Linear-Warmup-  
Linear-Decay Learning  
Rate Schedule:



Dropout Rates



Training on Nvidia-  
A100 GPUs using  
Deepspeed

# Model Fine Tuning

CodeExercise

Batch Size of  
256

Learning Rate  
12-4 w/ 50 steps  
of warmup

Weight Decay of  
0.01

Trained for a  
total of 6000  
steps during  
fine-tuning



# Model Performance

- HumanEval benchmark results
- Comparison with larger models
- The impact of high-quality data on performance

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

*phi-1-base struggles with the logical relationships in the prompts, while phi-1 can interpret the question and generate the answer correctly. In this example, even our 350M phi-1-small model shows some level of understanding of the problem even though the resulting solution is wrong.*

### Prompt:

```
""" Alice, Bob and Charles are playing games. Alice first pick a random number from 333 to 678, Bob then pick a number starting from Alice's number to 888, Charles then pick a number starting from 123 to Bob's number. Alice gets one point if Alice's number minus Charles's number is divisible by the floor of the square root of Bob's number, otherwise Bob gets one point. Simulate Alice's and Bob's points in 30 iterations.
"""
```

### phi-1

```
points = [0, 0]
for i in range(30):
    alice_num = random.randint(
        333, 678)
    bob_num = random.randint(
        alice_num, 888)
    charles_num = random.randint(
        123, bob_num)
    if (alice_num - charles_num) %
        points[0] += 1
    else:
        points[1] += 1
return f"Alice: {points[0]}, Bob: {points[1]}"
```

### phi-1-base

```
self.points = 0
self.alice = 333
self.bob = 678
self.charles = 888
self.bob_points = 0
self.alice_points = 0
self.iterations = 30
self.alice_turn = True
self.bob_turn = True
self.alice_turn_points = 0
self.bob_turn_points = 0
self.alice_turn_points_history = []
self.bob_turn_points_history = []
self.alice_turn_points_history.append(self.alice_points)
self.bob_turn_points_history.append(self.bob_points)
```

### phi-1-small

```
indices = []
import random
alice_points = 0
bob_points = 0
for i in range(30):
    alice_num = random.randint(
        333, 678)
    bob_num = random.randint(123,
        bob_num)
    if (alice_num - bob_num) %
        alice_points += 1
    if (i %
        bob_points += 1
print(f"Alice gets {alice_points} points, while Bob gets {bob_points} points.")
```

**Prompt:**

```
""" Write a tkinter application, there is a textfield on top, a submit button at the
button and a recover button next to it. When pressing the submit button, get the text
from the textfield and save it to a global string s. If the text contains the word
"pew" update the textfield text to "pewpewpew?", otherwise, add "no pew..." to the end.
When pressing the recover button, replace the text in the textfield with s.
"""
```

**phi-1**

```
...
global s
s = ""
def submit():
    global s
    text = textfield.get()
    if "pew" in text:
        textfield.delete(0, tk.END)
        textfield.insert(0, "pewpewpew...")
    else:
        textfield.delete(0, tk.END)
        textfield.insert(0, text + "no pew...")
    s = text

def recover():
    global s
    textfield.delete(0, tk.END)
    textfield.insert(0, s)

textfield = tk.Entry(root)
textfield.pack()
submit_button = tk.Button(root, text="Submit",
    command=submit)
submit_button.pack()
recover_button = tk.Button(root, text="Recover",
    , command=recover)
recover_button.pack()
root.mainloop()
```

**phi-1-base**

```
global s
s = s.replace("pew", "pewpewpew")
if "pew" in s:
    s = s.replace("pew", "pewpewpew")
else:
    s = s + "no pew..."
text.delete(0.0, END)
text.insert(0.0, s)
def main():
    global s
    s = "pewpewpew"
    text.delete(0.0, END)
    text.insert(0.0, s)
    text.tag_add("button", "<Button-1>", "<Button-3>")
    text.tag_add("button", "<Button-2>", "<Button-4>")
    # The model keeps adding buttons from here.
```

**phi-1-small**

```
import tkinter as tk
root = tk.Tk()
global s
text = ""
while True:
    event, values = tk.popup()
    if event == "submit":
        text += values.split()[0] + " "
    elif event == "textfield":
        s += values.split()[0] + " "
    elif event == "submit-button-text":
        text = values.split()[0] + " "
    elif event == "text-button-text":
        s = text
    if "pew" in text:
        text = text.replace("pew", "pewpewpew")
    if "no pew" in text:
        break
tk.destroy()
```

*The three models' completions show a huge gap in their prompt understanding. Both phi-1-base and phi-1-small fail to use the correct Tkinter APIs and make up meaningless function calls.*

*On the other hand, phi-1 implements the GUI and all the functions correctly*

# Model Evaluation – Emergent Properties




Comparing phi-1 and phi-1-small



The role of parameters in model emergence

# Model Limitations



Sensitivity to  
prompt length  
and style

Domain-  
specific  
knowledge gaps

Comparison  
with multi-  
language  
models

# Key Takeaways



The transformative power  
of high-quality data



The potential of smaller  
models with the right data



Reimagining the path to  
improved sustainable LLM  
performance