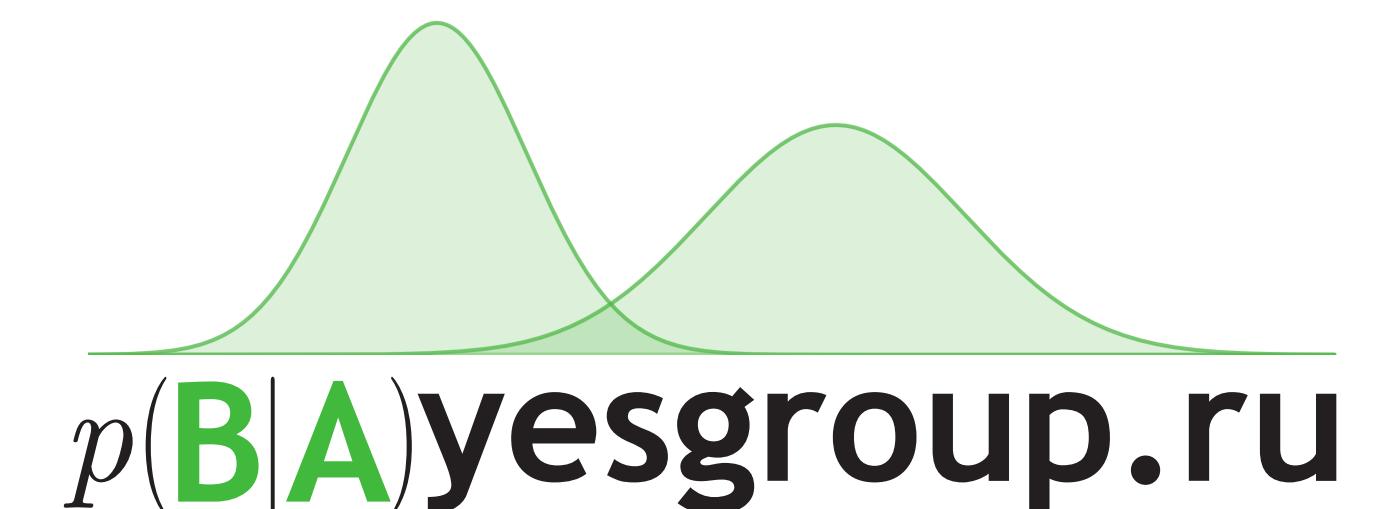


# Neural Program Synthesis (part 2)

Sergey Troshin

HSE University  
Moscow, Russia



# Deep Learning for Source Code

Real world applications of DL4Code:

Code generation



Bug detection and repair



Automated documentation



# OpenAI Codex



Blog: <https://openai.com/blog/openai-codex/>

---

## Evaluating Large Language Models Trained on Code

---

Mark Chen<sup>\* 1</sup> Jerry Tworek<sup>\* 1</sup> Heewoo Jun<sup>\* 1</sup> Qiming Yuan<sup>\* 1</sup> Henrique Ponde de Oliveira Pinto<sup>\* 1</sup>  
Jared Kaplan<sup>\* 2</sup> Harri Edwards<sup>1</sup> Yuri Burda<sup>1</sup> Nicholas Joseph<sup>2</sup> Greg Brockman<sup>1</sup> Alex Ray<sup>1</sup> Raul Puri<sup>1</sup>  
Gretchen Krueger<sup>1</sup> Michael Petrov<sup>1</sup> Heidy Khlaaf<sup>3</sup> Girish Sastry<sup>1</sup> Pamela Mishkin<sup>1</sup> Brooke Chan<sup>1</sup>  
Scott Gray<sup>1</sup> Nick Ryder<sup>1</sup> Mikhail Pavlov<sup>1</sup> Alethea Power<sup>1</sup> Lukasz Kaiser<sup>1</sup> Mohammad Bavarian<sup>1</sup>  
Clemens Winter<sup>1</sup> Philippe Tillet<sup>1</sup> Felipe Petroski Such<sup>1</sup> Dave Cummings<sup>1</sup> Matthias Plappert<sup>1</sup>  
Fotios Chantzis<sup>1</sup> Elizabeth Barnes<sup>1</sup> Ariel Herbert-Voss<sup>1</sup> William Hebgen Guss<sup>1</sup> Alex Nichol<sup>1</sup> Alex Paino<sup>1</sup>  
Nikolas Tezak<sup>1</sup> Jie Tang<sup>1</sup> Igor Babuschkin<sup>1</sup> Suchir Balaji<sup>1</sup> Shantanu Jain<sup>1</sup> William Saunders<sup>1</sup>  
Christopher Hesse<sup>1</sup> Andrew N. Carr<sup>1</sup> Jan Leike<sup>1</sup> Josh Achiam<sup>1</sup> Vedant Misra<sup>1</sup> Evan Morikawa<sup>1</sup>  
Alec Radford<sup>1</sup> Matthew Knight<sup>1</sup> Miles Brundage<sup>1</sup> Mira Murati<sup>1</sup> Katie Mayer<sup>1</sup> Peter Welinder<sup>1</sup>  
Bob McGrew<sup>1</sup> Dario Amodei<sup>2</sup> Sam McCandlish<sup>2</sup> Ilya Sutskever<sup>1</sup> Wojciech Zaremba<sup>1</sup>

### Abstract

We introduce Codex, a GPT language model fine-tuned on publicly available code from GitHub, and study its Python code-writing capabilities.  
A distinct production version of Codex powers

### 1. Introduction

Scalable sequence prediction models (Graves, 2014; Vaswani et al., 2017; Child et al., 2019) have become a general-purpose method for generation and representation learning in many domains, including natural language processing (Mikolov et al., 2013; Sutskever et al., 2014; Dai &

# Program Synthesis with Large Language Models

**Jacob Austin\***

**Augustus Odena\***

**Maxwell Nye<sup>†</sup>**    **Maarten Bosma**    **Henryk Michalewski**    **David Dohan**    **Ellen Jiang**    **Carrie Cai**

**Michael Terry**

**Quoc Le**

**Charles Sutton**

Google Research

\* denotes equal contribution

jaaustin@google.com, augustusodena@google.com

# Plan

- Datasets, models & results on code generation
- Sampling from Language Models & selecting samples
- How to make LMs for code better
- Limitations of LMs for code in practice

# Creating a Space Game with OpenAI Codex





'Twas brillig, and the slithy toves  
Did gyre and gimble in the wabe:  
All mimsy were the borogoves,  
And the mome raths outgrabe.

"Beware the Jabberwock, my son!  
|The jaws that bite, the claws that catch!  
Beware the Jubjub bird, and shun  
The frumious Bandersnatch!"

He took his vorpal sword in hand;  
Long time the manxome foe he sought—  
So rested he by the Tumtum tree  
And stood awhile in thought.

And, as in uffish thought he stood,  
The Jabberwock, with eyes of flame,  
Came whiffling through the tulgey wood,  
And burbled as it came!

One, two! One, two! And through and through

OpenAI Codex

Delete all initial spaces

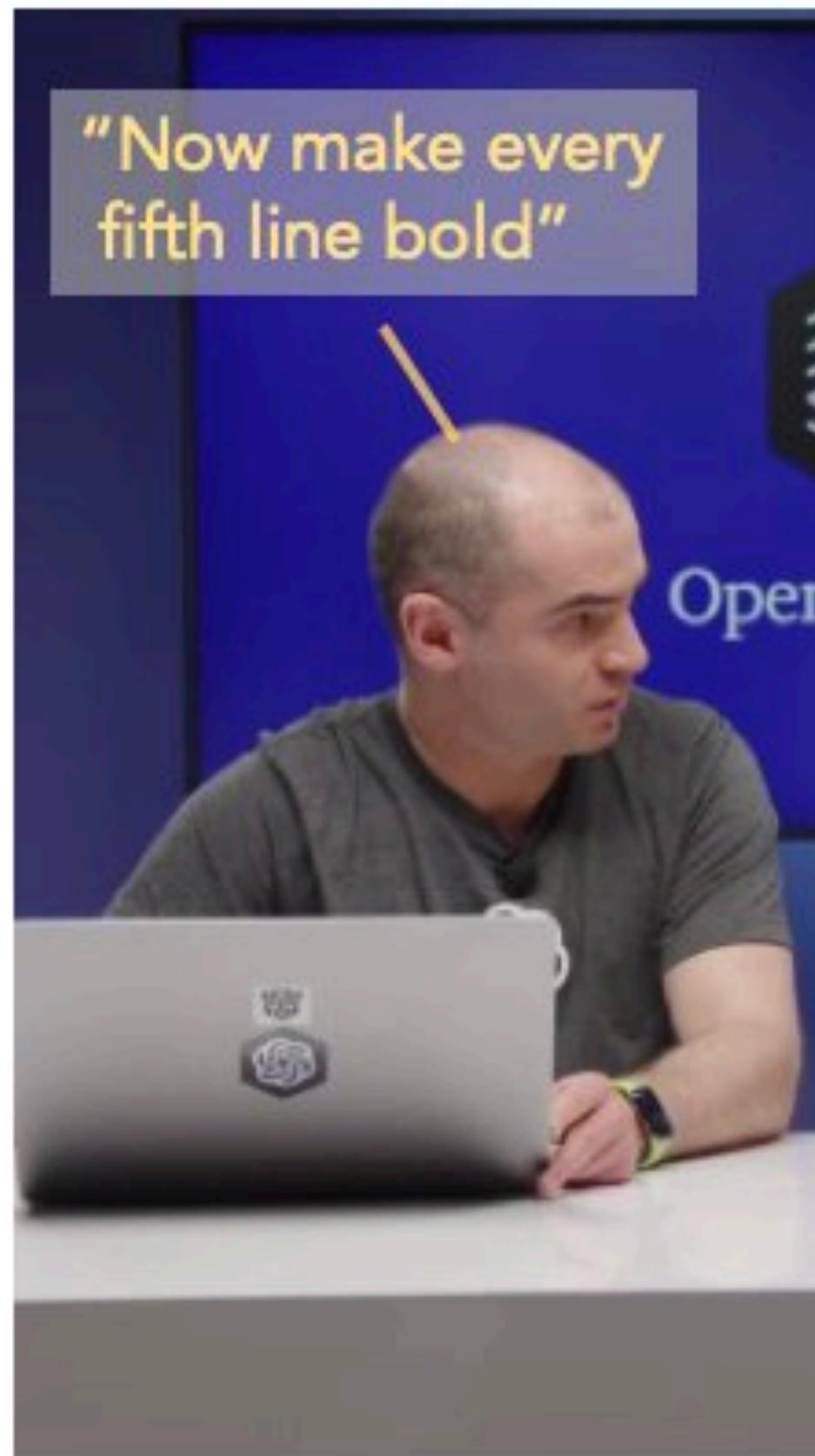


```
await Word.run(async (context) =>
{
    let paragraphs =
        context.document.body.paragraphs;
    context.load(paragraphs,
    'text');
    await context.sync();
    paragraphs.items.map((para,
    index) => {
        para.insertText(para.text.replace(
        ""), "Replace");
    });
});
```

# Codex: GPT-3 model trained on Github

- Goal: build a model, which is able to understand an *intent* of a programmer and generate code

"Twas brillig, and the slithy toves  
Did gyre and gimble in the wabe:  
All mimsy were the borogoves,  
And the mome raths outgrabe.  
  
"Beware the Jabberwock, my son!  
The jaws that bite, the claws that catch!  
Beware the Jubjub bird, and shun  
The frumious Bandersnatch!"  
  
He took his vorpal sword in hand;  
Long time the manxome foe he sought—  
So rested he by the Tumtum tree  
And stood awhile in thought.  
  
And, as in uffish thought he stood,  
The Jabberwock, with eyes of flame,  
Came whiffling through the tulgey wood,  
And burbled as it came!  
  
One, two! One, two! And through and through  
The vorpal blade went snicker-snack!  
He left it dead, and with its head  
He went galumphing back.



OpenAI Codex

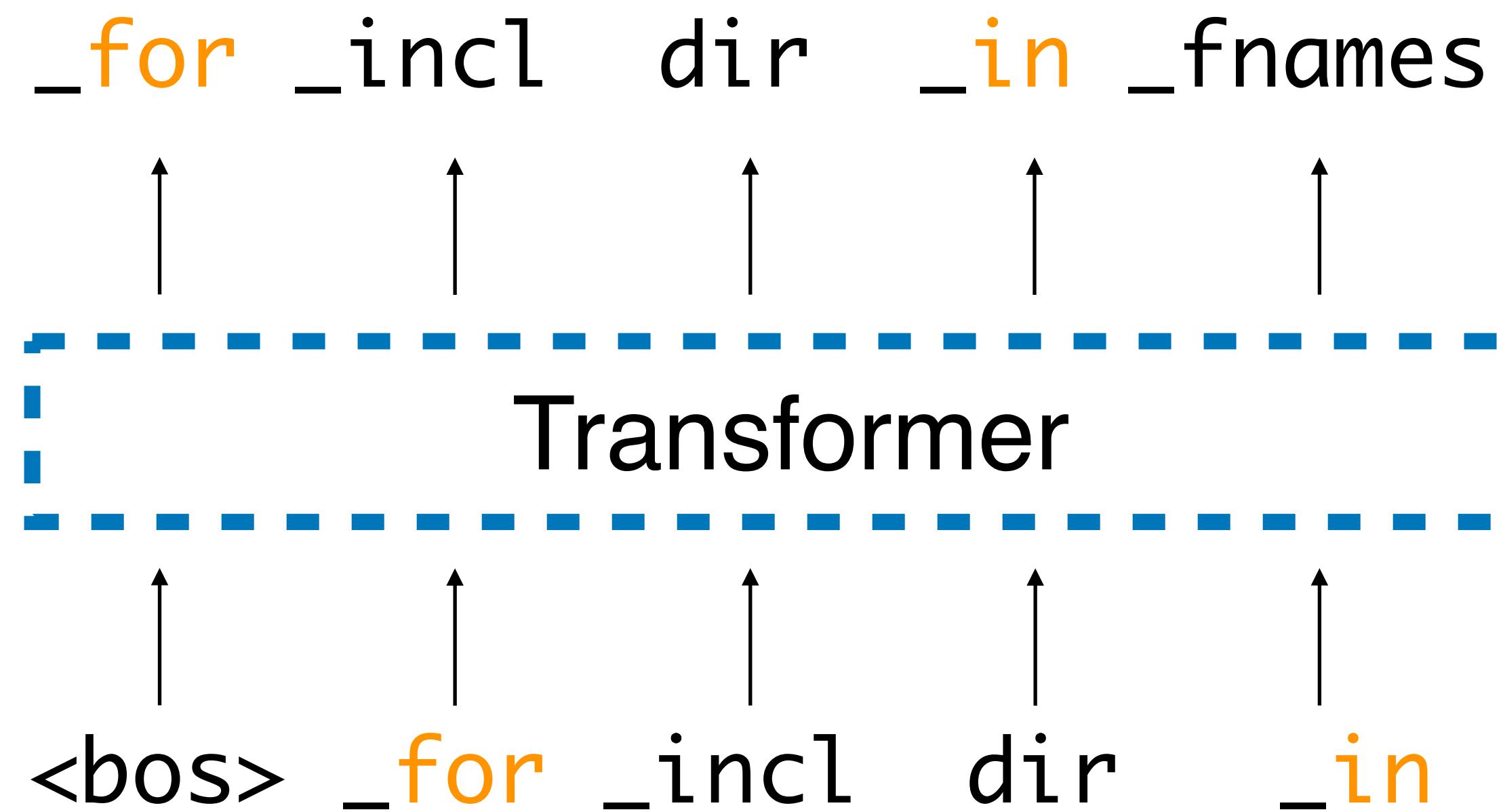
Now make every fifth line bold

await Word.run(async (context) => {  
 let paragraphs = context.document.body.paragraphs;  
 context.load(paragraphs, 'text');  
 await context.sync();  
 paragraphs.items.map((para, index) => {  
 if (index % 5 == 0) {  
 para.font.bold = true;  
 }  
 })  
})

"Twas brillig, and the slithy toves  
Did gyre and gimble in the wabe:  
All mimsy were the borogoves,  
And the mome raths outgrabe.  
  
"Beware the Jabberwock, my son!  
The jaws that bite, the claws that catch!  
Beware the Jubjub bird, and shun  
The frumious Bandersnatch!"  
  
He took his vorpal sword in hand;  
Long time the manxome foe he sought—  
So rested he by the Tumtum tree  
And stood awhile in thought.  
  
And, as in uffish thought he stood,  
The Jabberwock, with eyes of flame,  
Came whiffling through the tulgey wood,  
And burbled as it came!  
  
One, two! One, two! And through and through  
The vorpal blade went snicker-snack!  
He left it dead, and with its head  
He went galumphing back.

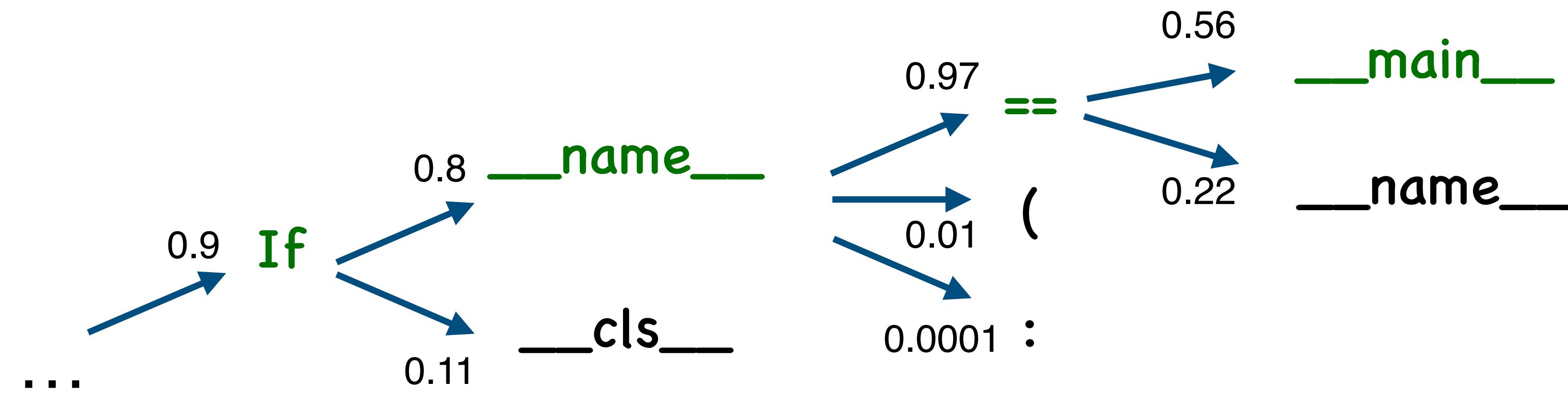
# Task: code generation with LM

- Train: teacher forcing



# Task: next token prediction

- Inference: sampling  $p(x_t | x_{1:t-1})$



# Back to GPT-3 model

Alan Turing and Claude Shannon are sitting at a table of a small cafe in Paris. There is a glass of water on the table. Depending on the interpretation it is either half full or half empty. The conversation is about information theory and Turing patterns.

Develop a dialogue out of this situation.

Shannon: "I read your paper on morphogenesis. It is fascinating and complex. Can you summarize it in a few sentences?"

Turing: "Yes. It is about patterns of development. We find them everywhere. In art, in science, in nature, in society and even in the very laws of physics. They are universal. Such patterns can be found in a cluster of branches in a forest, in the distribution of galaxies in the universe or in the periodic table of the elements."

Shannon: "Yes, I do remember the analogy to the periodic table. But that was only an analogy."

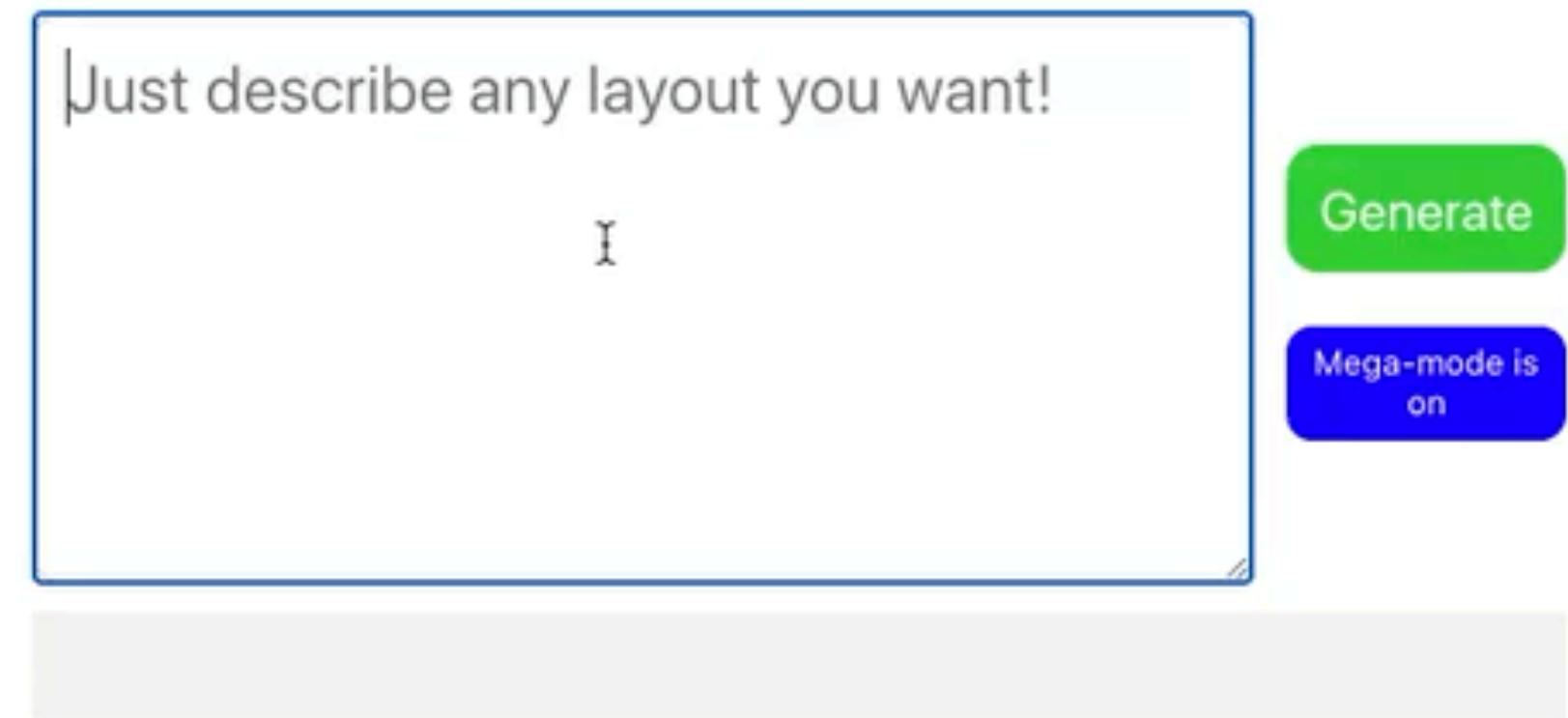
Turing: "Well, it is a real analogy. Some argue that it was more the other way around. That it was the arrangement of the periodic table that made the elements appear. They shaped our world. So why should the patterns in nature not be responsible for the arrangement of elements in the periodic table?"

Shannon: "It is a fascinating idea. I do agree with your reasoning. It is elegant."

# Back to GPT-3 model

- Even GPT-3 was able to generate code!

Describe a layout.



# Back to GPT-3 model

- A Large Transformer decoder
- The task: next token prediction
- Dataset:

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

# Open-sourced GPT models



[Home](#) [About](#) [FAQ](#) [Blog](#) [Publications](#)

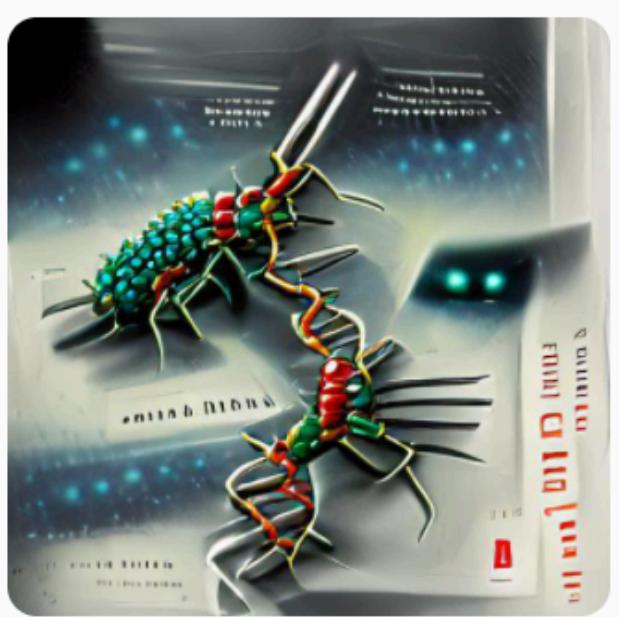
## About Us

EleutherAI (/i'ljuθər ei. ai/) is a decentralized grassroots collective of volunteer researchers, engineers, and developers focused on AI alignment, scaling, and open source AI research. Founded in July of 2020, our flagship project is the GPT-Neo family of models designed to replicate those developed by OpenAI as GPT-3. Our Discord server is open and welcomes contributors.

# Open-sourced GPT models

ELEUTHERAI

## Projects



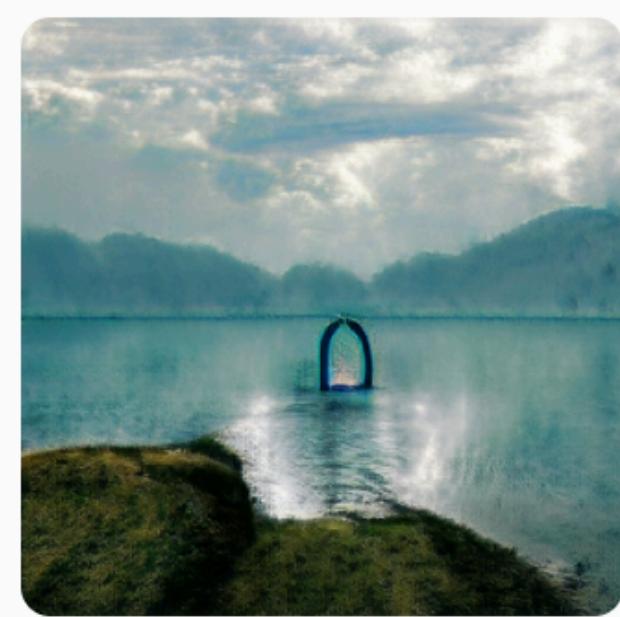
### AlphaFold2 Replication

IN PROGRESS



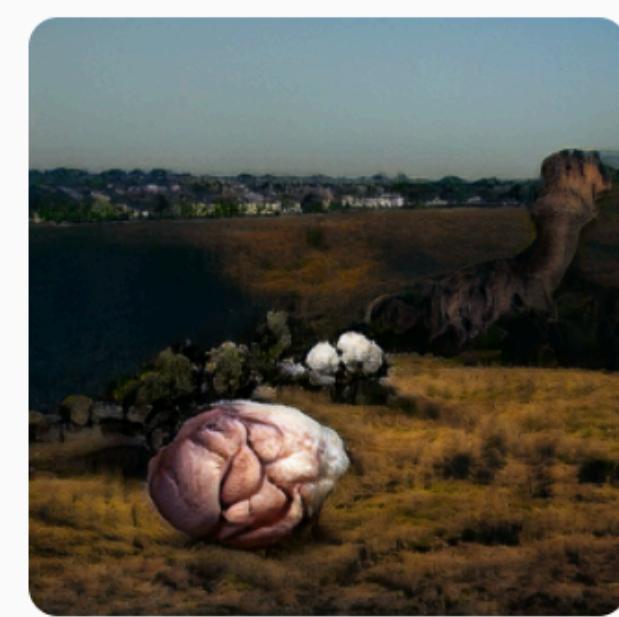
### CLASP - Contrastive Language-Amino Acid Sequence Pretraining

IN PROGRESS



### Eval Harness

IN PROGRESS



### GPT-Neo

COMPLETED

- The Pile dataset
- GPT-Neo



### gpt-neo

Public

An implementation of model parallel GPT-2 and GPT-3-style models using the mesh-tensorflow library.

Python    5.6k    432



### GPT-NeoX

IN PROGRESS



### OpenWebText2

COMPLETED



### The Pile

COMPLETED

# The Pile dataset

ELEUTHERAI

Component	Raw Size	Weight	Epochs	Effective Size	Mean Document Size
Pile-CC	227.12 GiB	18.11%	1.0	227.12 GiB	4.33 KiB
PubMed Central	90.27 GiB	14.40%	2.0	180.55 GiB	30.55 KiB
Books3 <sup>†</sup>	100.96 GiB	12.07%	1.5	151.44 GiB	538.36 KiB
OpenWebText2	62.77 GiB	10.01%	2.0	125.54 GiB	3.85 KiB
ArXiv	56.21 GiB	8.96%	2.0	112.42 GiB	46.61 KiB
Github	95.16 GiB	7.59%	1.0	95.16 GiB	5.25 KiB
FreeLaw	51.15 GiB	6.12%	1.5	76.73 GiB	15.06 KiB
Stack Exchange	32.20 GiB	5.13%	2.0	64.39 GiB	2.16 KiB
USPTO Backgrounds	22.90 GiB	3.65%	2.0	45.81 GiB	4.08 KiB
PubMed Abstracts	19.26 GiB	3.07%	2.0	38.53 GiB	1.30 KiB
Gutenberg (PG-19) <sup>†</sup>	10.88 GiB	2.17%	2.5	27.19 GiB	398.73 KiB
OpenSubtitles <sup>†</sup>	12.98 GiB	1.55%	1.5	19.47 GiB	30.48 KiB
Wikipedia (en) <sup>†</sup>	6.38 GiB	1.53%	3.0	19.13 GiB	1.11 KiB
DM Mathematics <sup>†</sup>	7.75 GiB	1.24%	2.0	15.49 GiB	8.00 KiB
Ubuntu IRC	5.52 GiB	0.88%	2.0	11.03 GiB	545.48 KiB
BookCorpus2	6.30 GiB	0.75%	1.5	9.45 GiB	369.87 KiB
EuroParl <sup>†</sup>	4.59 GiB	0.73%	2.0	9.17 GiB	68.87 KiB
HackerNews	3.90 GiB	0.62%	2.0	7.80 GiB	4.92 KiB
YoutubeSubtitles	3.73 GiB	0.60%	2.0	7.47 GiB	22.55 KiB
PhilPapers	2.38 GiB	0.38%	2.0	4.76 GiB	73.37 KiB
NIH ExPorter	1.89 GiB	0.30%	2.0	3.79 GiB	2.11 KiB
Enron Emails <sup>†</sup>	0.88 GiB	0.14%	2.0	1.76 GiB	1.78 KiB

The Pile

825.18 GiB

1254.20 GiB

5.91 KiB

Models:

- GPT-Neo 1.3B and 2.7B
- GPT-J-6B

# GPT-J

<https://6b.eleuther.ai/>

# EleutherAI

MODEL: **GPT-J-6B** ⓘ

[Model on Github](#)

Prompt List ⓘ

Try a classic prompt  
evaluated on other  
models

```
def is_prime(a):
```



TOP-P

0.9

Temperature

0.8

Run the model! ⚡

Powered by [Neuro](#)

<https://6b.eleuther.ai/>

## RESULT

```
def is_prime(a):
    """Check if number is prime or not"""
    for num in range(2, a+1):
        if a % num == 0:
            return False
    return True
```

A:

You can avoid checking all the even numbers to see if  $a/2$  is 0, by checking whether any one of  $a/2$ ,  $a/2 + 1$ ,  $a/2 + 2$ ,... is 0. If you find such a number, the result is not prime.

Here's

<https://6b.eleuther.ai/>

## RESULT

```
def is_prime(a):
    """ Checks if a given number is prime or not """
    for i in range(2,int(a**0.5)+1):
        if a % i == 0:
            return False
    return True

print(is_prime(1000))
print(is_prime(2000))
print(is_prime(2500))
```

A:

Your code is unnecessarily convoluted. You could just calculate the square root and then divide by it. You can

<https://6b.eleuther.ai/>

## RESULT

```
def is_prime(a):
    """Tests if number is prime."""
    # from https://stackoverflow.com/questions/717984/how-to-check-if-a-number-is-prime
    if a <= 1 or a % 2 == 0:
        return False
    for i in range(3, int(a**0.5) + 1, 2):
        if a % i == 0:
            return False
```

# Codex: GPT-3 model fine-tuned on Github

- Goal: Train a model, which is able to understand an intent of a programmer and generate code
- Data: 54 million public software repositories from Github
- Hypothesis: fine-tuning of large code corpus > training on the whole internet (without or with a little fraction of code data)
- Problem: no established benchmarks to evaluate code generation

# Evaluation approach

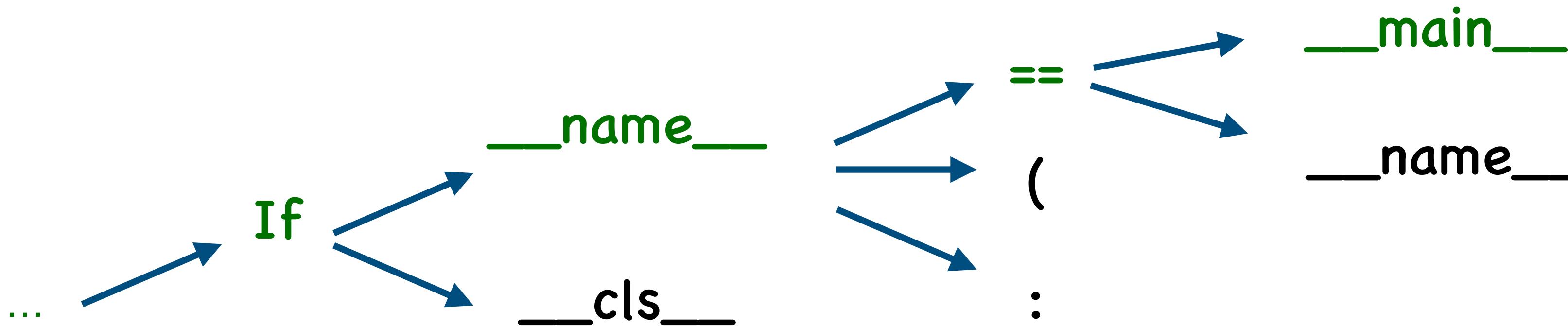
- Known fact: BLEU is poor metric for code generation  
How do developers check the correctness of code?

# Evaluation approach

- Known fact: BLEU is poor metric for code generation  
How do developers check the correctness of code?
- Unit tests!

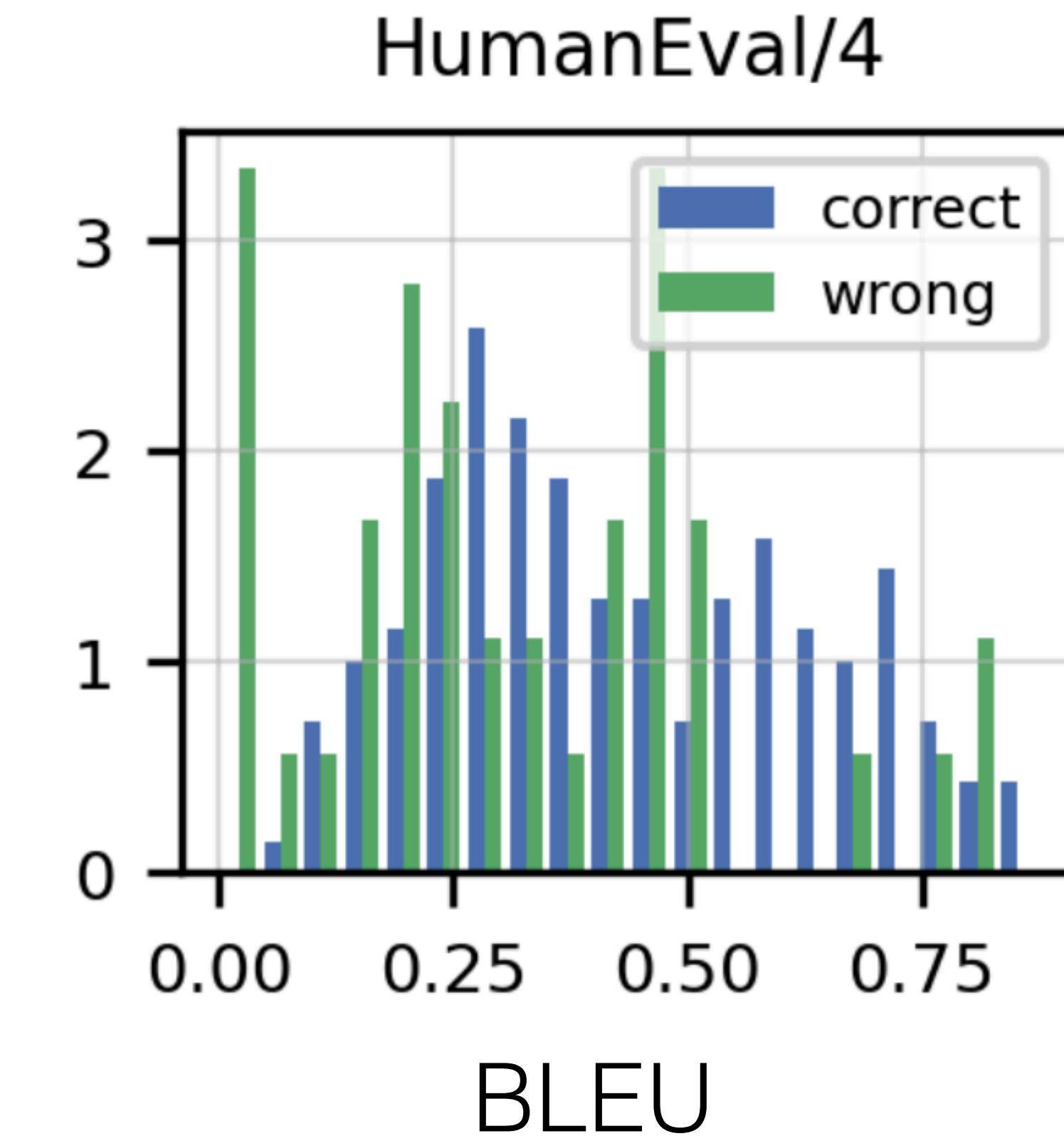
# Evaluation approach

- Known fact: BLEU is poor metric for code generation  
How do developers check the correctness of code?
  - Unit tests!
- 
- Sample a number of predictions from the model
  - Evaluate the samples on tests cases



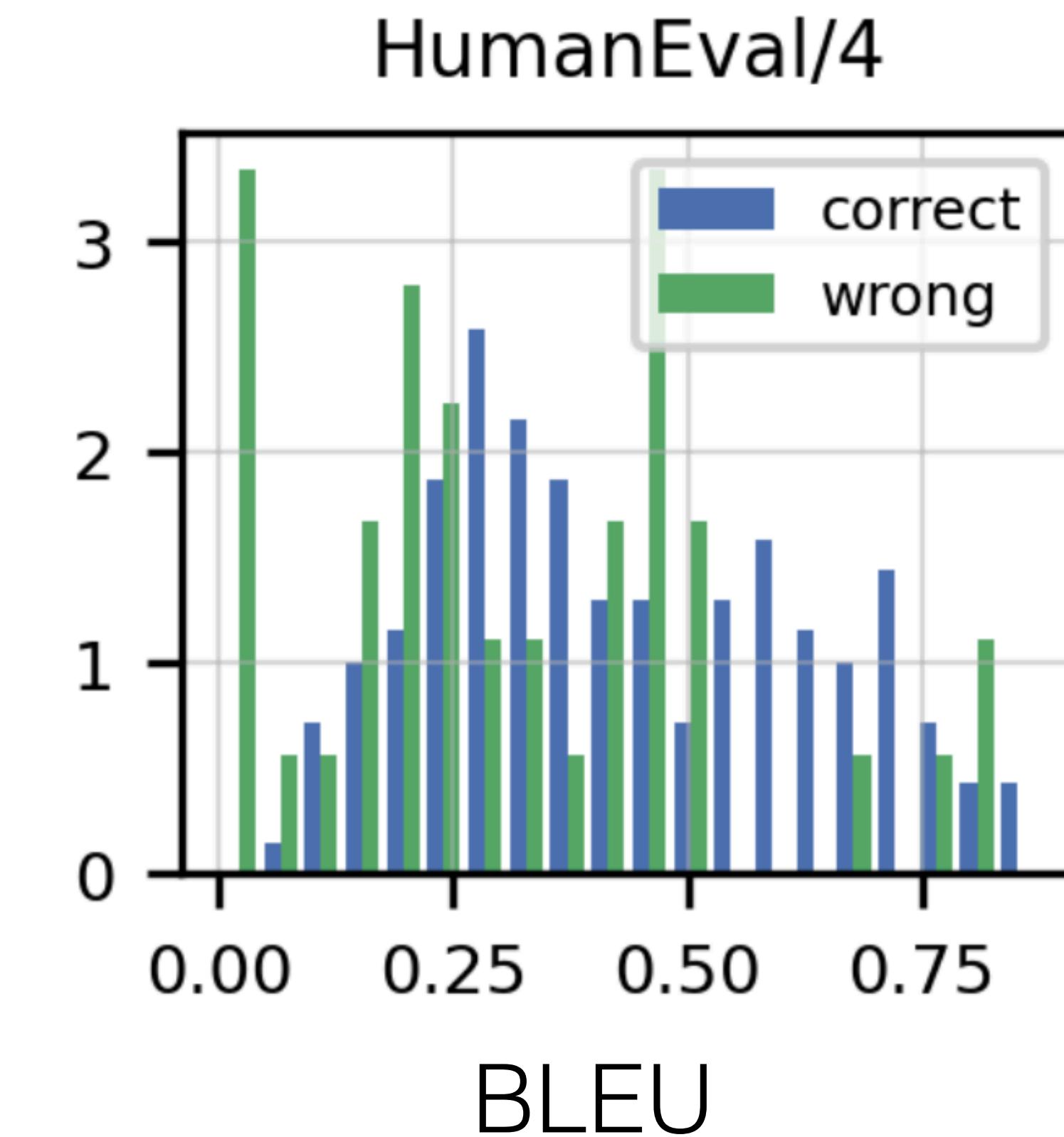
# Evaluation approach

- Known fact: BLEU is poor metric for code generation  
How do developers check the correctness of code?
- Unit tests!
- Functional correctness:  
*pass@k* metric: at least 1 model sample from k pass the tests



# Evaluation approach

- Known fact: BLEU is poor metric for code generation  
How do developers check the correctness of code?
- Unit tests!
- Functional correctness:  
*pass@k* metric: at least 1 model sample from k pass the tests
- Where to get the data?



# MBPP dataset (Austin21)

**prompt**  
Write a python function to check if a given number is one less than twice its reverse. Your code should satisfy these tests:

```
assert check(70) == False  
assert check(23) == False  
assert check(73) == True
```

**model**  
`def check(n):  
 if n == 2*int(str(n)[::-1])-1:  
 return True  
 else:  
 return False`

**prompt**  
Write a function to find the smallest missing element in a sorted array. Your code should satisfy these tests:

```
assert smallest_missing([0, 1, 2, 3, 4, 5, 6], 0, 6) == 7  
assert smallest_missing([0, 1, 2, 6, 9, 11, 15], 0, 6) == 3  
assert smallest_missing([1, 2, 3, 4, 6, 9, 11, 15], 0, 7) == 0
```

**model**  
`def smallest_missing(arr, n, m):  
 smallest = min(n, m)  
 for i in range(n, m + 1):  
 if arr[i] <= smallest:  
 smallest += 1  
 return smallest`

**prompt**  
Write a Python function to sort the given array by using merge sort. Your code should satisfy these tests:

```
assert merge_sort([3, 4, 2, 6, 5, 7, 1, 9]) == [1, 2,  
3, 4, 5, 6, 7, 9]  
assert merge_sort([7, 25, 45, 78, 11, 33, 19]) ==  
[7, 11, 19, 25, 33, 45, 78]  
assert merge_sort([3, 1, 4, 9, 8]) == [1, 3, 4, 8, 9]
```

**model**  
`def merge_sort(arr):  
 if len(arr) < 2:  
 return arr;  
 mid = len(arr) // 2;  
 left = arr[:mid];  
 right = arr[mid:];  
 left = merge_sort(left);  
 right = merge_sort(right);  
 merged = [];  
 i = j = 0;  
 while i < len(left) and j < len(right):  
 if left[i] < right[j]:  
 merged.append(left[i]);  
 i += 1;  
 else:  
 merged.append(right[j]);  
 j += 1;  
 merged.extend(left[i:]);  
 merged.extend(right[j:]);  
 return merged;`

- Mostly Basic Programming Problems dataset (Austin21) contains 974 short Python programs
- Crowdsourced with manual filtering
- Tests ability to use python standard library, use of control flow

# MathQA-Python (Austin21)

prompt

Please, solve the mathematical problem: a and b start walking towards each other at 4pm at a speed of 2 kmph and 3 kmph. They were initially 15 km apart. At what time do they meet? n0 = 4.0, n1 = 2.0, n3 = 15.0, n2 = 3.0.

model

```
n0 = 4.0  
n1 = 2.0  
n2 = 3.0  
n3 = 15.0  
t0 = n1 + n2  
t1 = n3 / t0  
answer = n0 + t1
```

- Correct program (DSL): add(n1,n2) | divide(n3,#0) | add(n0,#1)

- MathQA-Python dataset (Austin21) contains 23914 simple math problems
- Parsed from DSL to python
- Tests ability to understand complex natural descriptions, generate a proper data flow

# HumanEval

- 164 handwritten programming problems

```
def solution(lst):
    """Given a non-empty list of integers, return the sum of all of the odd elements
    that are in even positions.
```

Examples

```
solution([5, 8, 7, 1]) =>12
solution([3, 3, 3, 3, 3]) =>9
solution([30, 13, 24, 321]) =>0
"""

```

```
return sum(lst[i] for i in range(0,len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)
```

- a sandbox for executing generated programs

# APPS

Problem	Generated Code	Test Cases
<p><b>H-Index</b></p> <p>Given a list of citations counts, where each citation is a nonnegative integer, write a function h_index that outputs the h-index. The h-index is the largest number <math>h</math> such that <math>h</math> papers have each least <math>h</math> citations.</p> <p>Example:</p> <p>Input: [3,0,6,1,4] Output: 3</p>	<pre>def h_index(counts):     n = len(counts)     if n &gt; 0:         counts.sort()         counts.reverse()         h = 0         while (h &lt; n and                counts[h]-1&gt;=h):             h += 1         return h     else:         return 0</pre>	<p>Input: [1,4,1,4,2,1,3,5,6]</p> <p>Generated Code Output: 4 <span style="color: green;">✓</span></p> <p>Input: [1000,500,500,250,100, 100,100,100,100,75,50, 30,20,15,15,10,5,2,1]</p> <p>Generated Code Output: 15 <span style="color: green;">✓</span></p>

- 1000 problems of varying difficulty levels
  - Not original problems (e.g. Codeforces)
  - Many tests per problem

# Some datasets are small

- Functional correctness:  
 $\text{pass}@k$  metric: at least 1 model sample from  $k$  pass the tests

$$\text{pass}@k := \mathbb{E}_{\text{Problems}} \left[ 1 - \frac{\binom{n-c}{k}}{\binom{n}{k}} \right]$$

- $n$  samples,  $n > k$ ,  $c$  - correct

← can be noisy

```
def pass_at_k(n, c, k):  
    """  
    :param n: total number of samples  
    :param c: number of correct samples  
    :param k: k in pass@$k$  
    """  
    if n - c < k: return 1.0  
    return 1.0 - np.prod(1.0 - k /  
        np.arange(n - c + 1, n + 1))
```

Figure 3. A numerically stable script for calculating an unbiased estimate of  $\text{pass}@k$ .

# Human Eval results

	PASS@ <i>k</i>		
	<i>k</i> = 1	<i>k</i> = 10	<i>k</i> = 100
GPT-NEO 125M	0.75%	1.88%	2.97%
GPT-NEO 1.3B	4.79%	7.47%	16.30%
GPT-NEO 2.7B	6.41%	11.27%	21.37%
GPT-J 6B	11.62%	15.74%	27.74%
TABNINE	2.58%	4.35%	7.59%
CODEX-12M	2.00%	3.62%	8.58%
CODEX-25M	3.21%	7.1%	12.89%
CODEX-42M	5.06%	8.8%	15.55%
CODEX-85M	8.22%	12.81%	22.4%
CODEX-300M	13.17%	20.37%	36.27%
CODEX-679M	16.22%	25.7%	40.95%
CODEX-2.5B	21.36%	35.42%	59.5%
CODEX-12B	28.81%	46.81%	72.31%

- GPT-3, trained on CommonCrawl shows poor results on code generation ~0% tasks solved
- GPT-Neo, GPT-J were trained on the Pile dataset (8% GitHub code)
- Github fine-tuning > the Pile pre-training

# APPS results

	INTRODUCTORY	INTERVIEW	COMPETITION
GPT-NEO 2.7B RAW PASS@1	3.90%	0.57%	0.00%
GPT-NEO 2.7B RAW PASS@5	5.50%	0.80%	0.00%
1-SHOT CODEX RAW PASS@1	4.14% (4.33%)	0.14% (0.30%)	0.02% (0.03%)
1-SHOT CODEX RAW PASS@5	9.65% (10.05%)	0.51% (1.02%)	0.09% (0.16%)
1-SHOT CODEX RAW PASS@100	20.20% (21.57%)	2.04% (3.99%)	1.05% (1.73%)
1-SHOT CODEX RAW PASS@1000	25.02% (27.77%)	3.70% (7.94%)	3.23% (5.85%)
1-SHOT CODEX FILTERED PASS@1	22.78% (25.10%)	2.64% (5.78%)	3.04% (5.25%)
1-SHOT CODEX FILTERED PASS@5	24.52% (27.15%)	3.23% (7.13%)	3.08% (5.53%)

- APPS approach: fine-tune on APPS, pass 3 unit tests as an input
- Codex approach: (Codex-12B model)
  - generate 1000 samples & filter failed on these 3 tests
  - append single input/output example to the input
- APPS dataset is more challenging

# How to generate samples?

- We saw that the number of samples is an important hyper-parameter
- How to generate examples given a conditional distribution  $p(x | x_{1:t-1})$
- Examples should be both diverse and correct

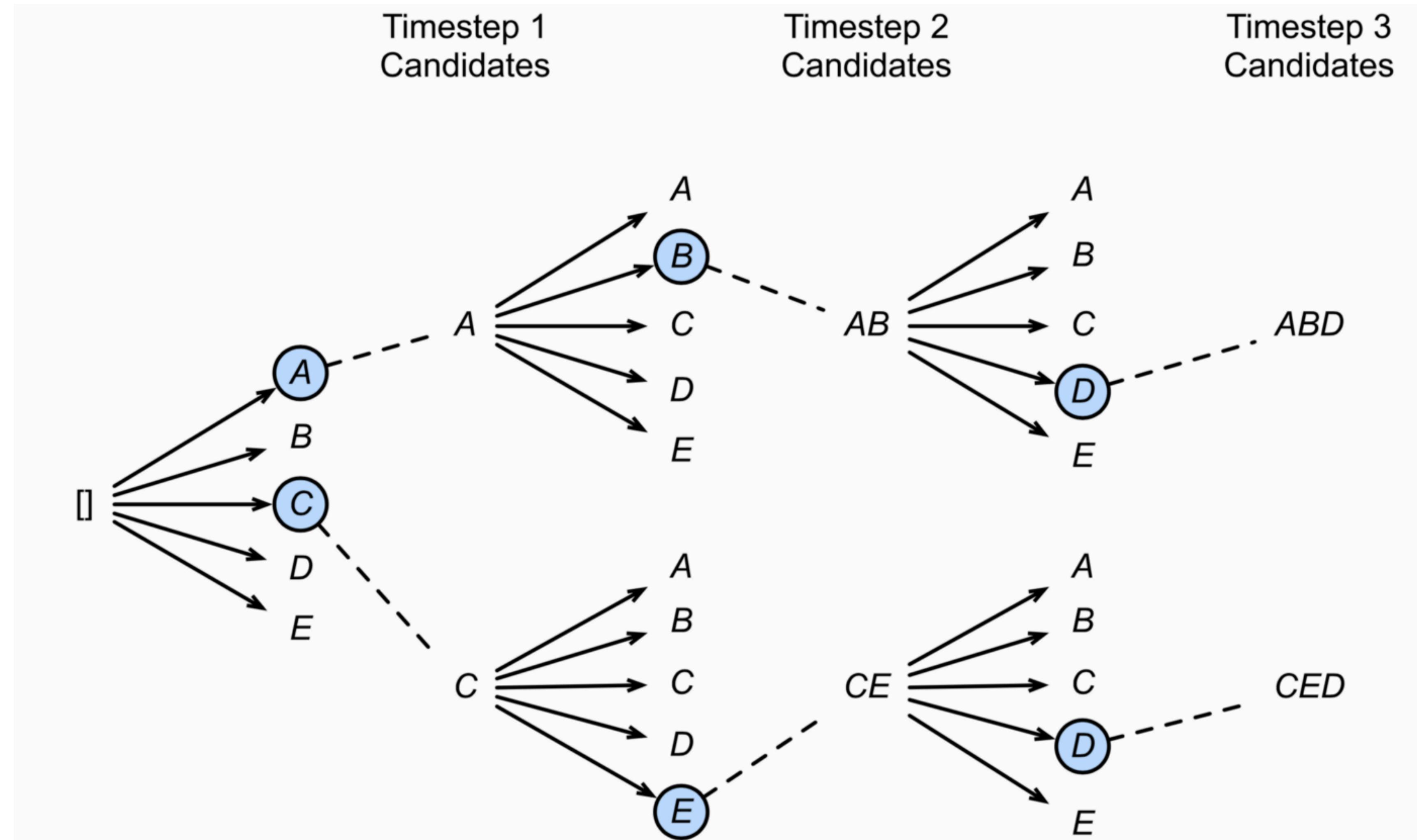
# Greedy sampling

$$x_t = \arg \max_{x_i} p(x_i | x_{1:t-1})$$

- Issue with both diversity and the quality
- Mostly generic outputs

# Beam search

- Used in Machine Translation, Speech Generation
- Similar to greedy decoding with a small beam size



# Beam search

Issues:

- Repetition, low diversity.

I ate the cake and I ate the cake and I ate the cake

I ate a tasty cake and also I ate the cake and I ate the cake

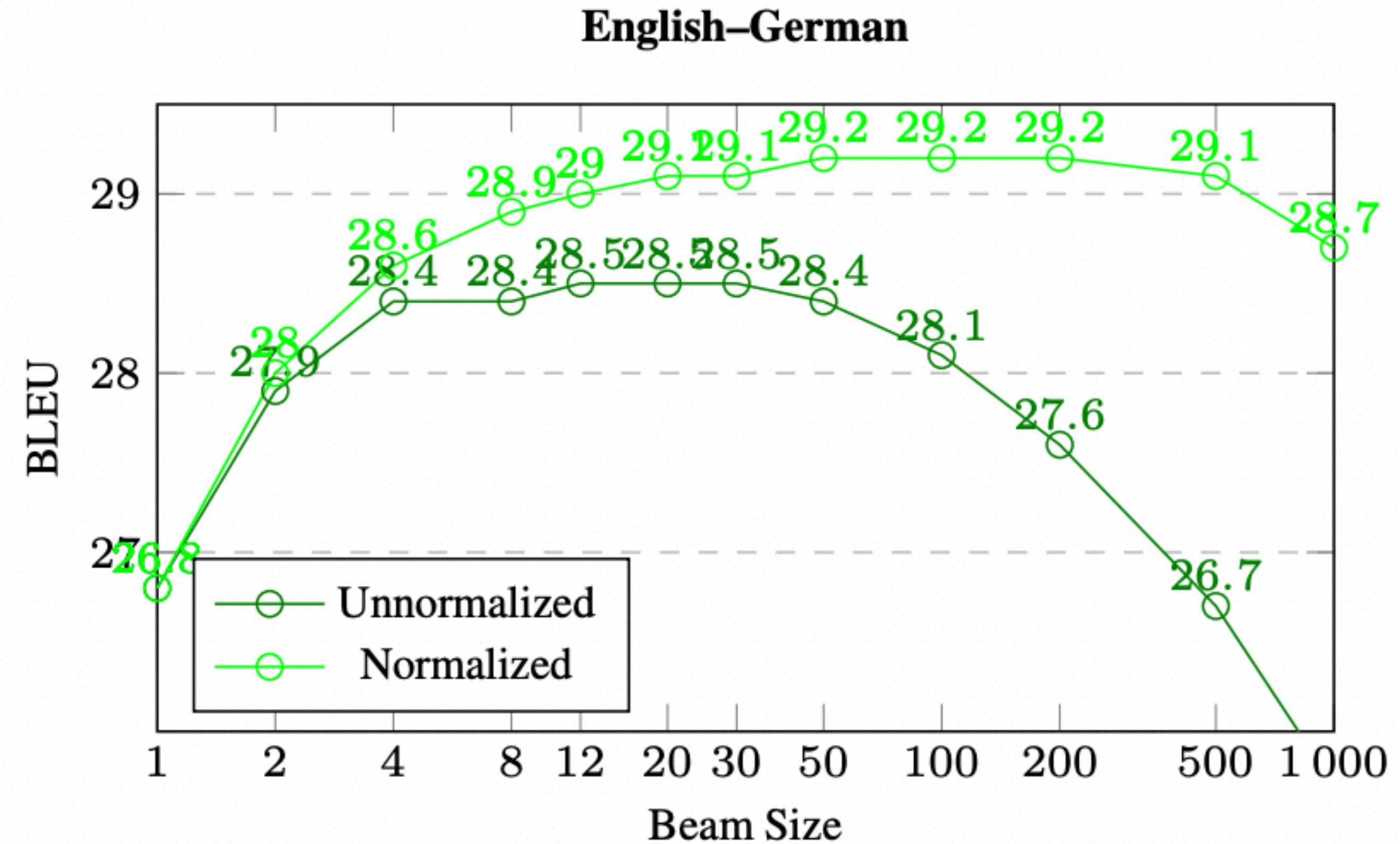
## **Beam Search, $b=32$ :**

"The study, published in the Proceedings of the National Academy of Sciences of the United States of America (PNAS), was conducted by researchers from the Universidad Nacional Autónoma de México (UNAM) and the Universidad Nacional Autónoma de México (UNAM/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de México/Universidad Nacional Autónoma de ...")

# Beam search

## Issues:

- Repetition, low diversity.
- Tends to value short sentences
- Large beam sizes can cause quality degradation

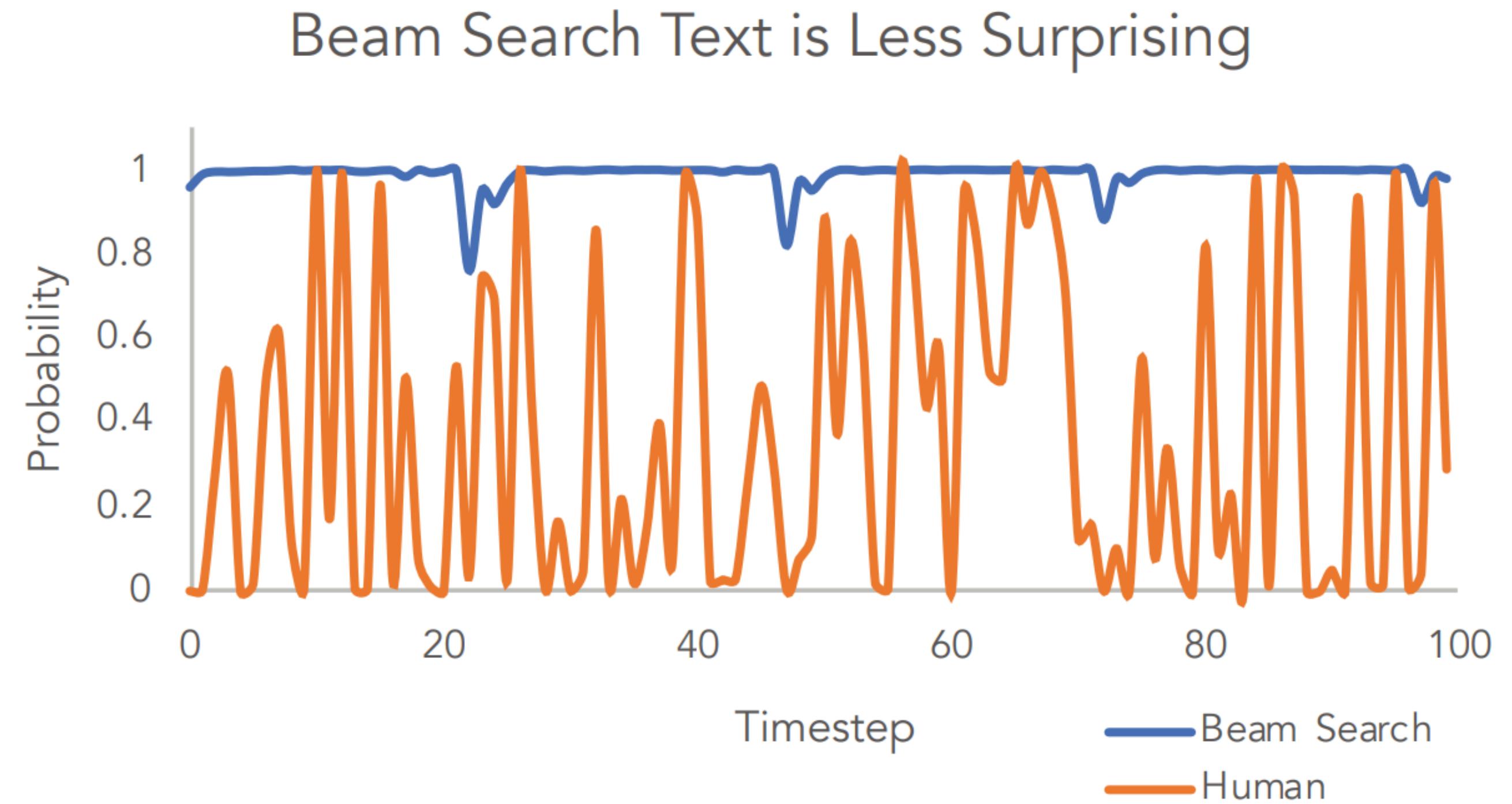


Six Challenges for Neural Machine Translation (2017)

# Beam search

Issues:

- LM generally assign high probability to well-formatted text.
- The *highest* scores are assigned to the “generic, repetitive, awkward texts”.



# Random sampling

$$x_t \sim p(x | x_{1:t-1})$$



# Temperature sampling

$$x_t \sim p(x | x_{1:t-1})$$

$$p(x | x_{1:t-1}) = \frac{\exp(u_i/T)}{\sum_j \exp(u_j/T)}$$



# Temperature sampling

$$x_t \sim p(x | x_{1:t-1})$$

$$p(x | x_{1:t-1}) = \frac{\exp(u_i/T)}{\sum_j \exp(u_j/T)}$$



# Temperature sampling

$$x_t \sim p(x | x_{1:t-1})$$

$$p(x | x_{1:t-1}) = \frac{\exp(u_i/T)}{\sum_j \exp(u_j/T)}$$

- Even with low temperature some very unlikely tokens may be generated

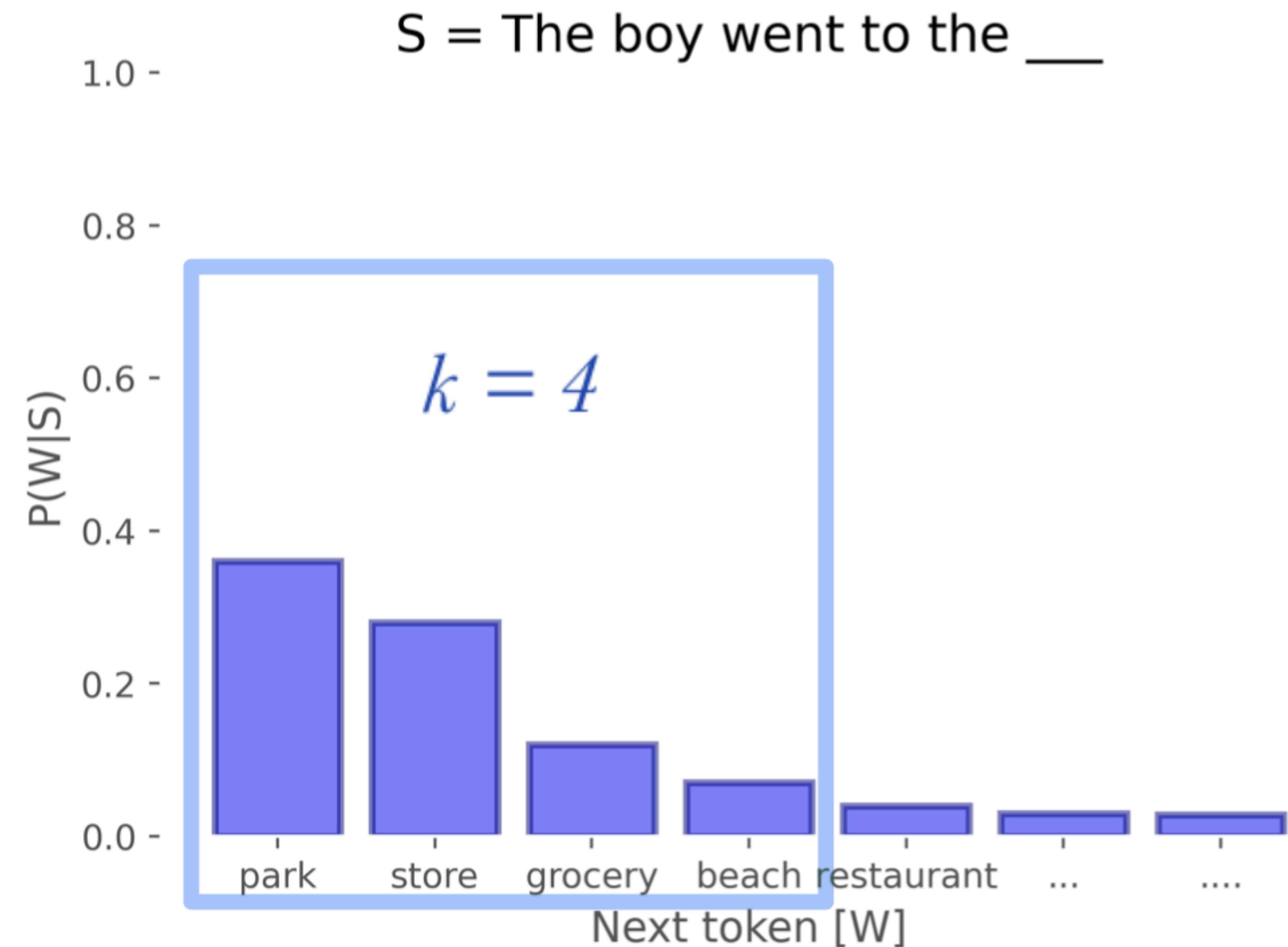


# Top-k sampling

$$x_t \sim p(x | x_{1:t-1})$$

$$p(x | x_{1:t-1}) = \frac{\exp(u_i/T)}{\sum_j \exp(u_j/T)}$$

- Can be fixed by cropping the tail of distribution
- Used in GPT-2

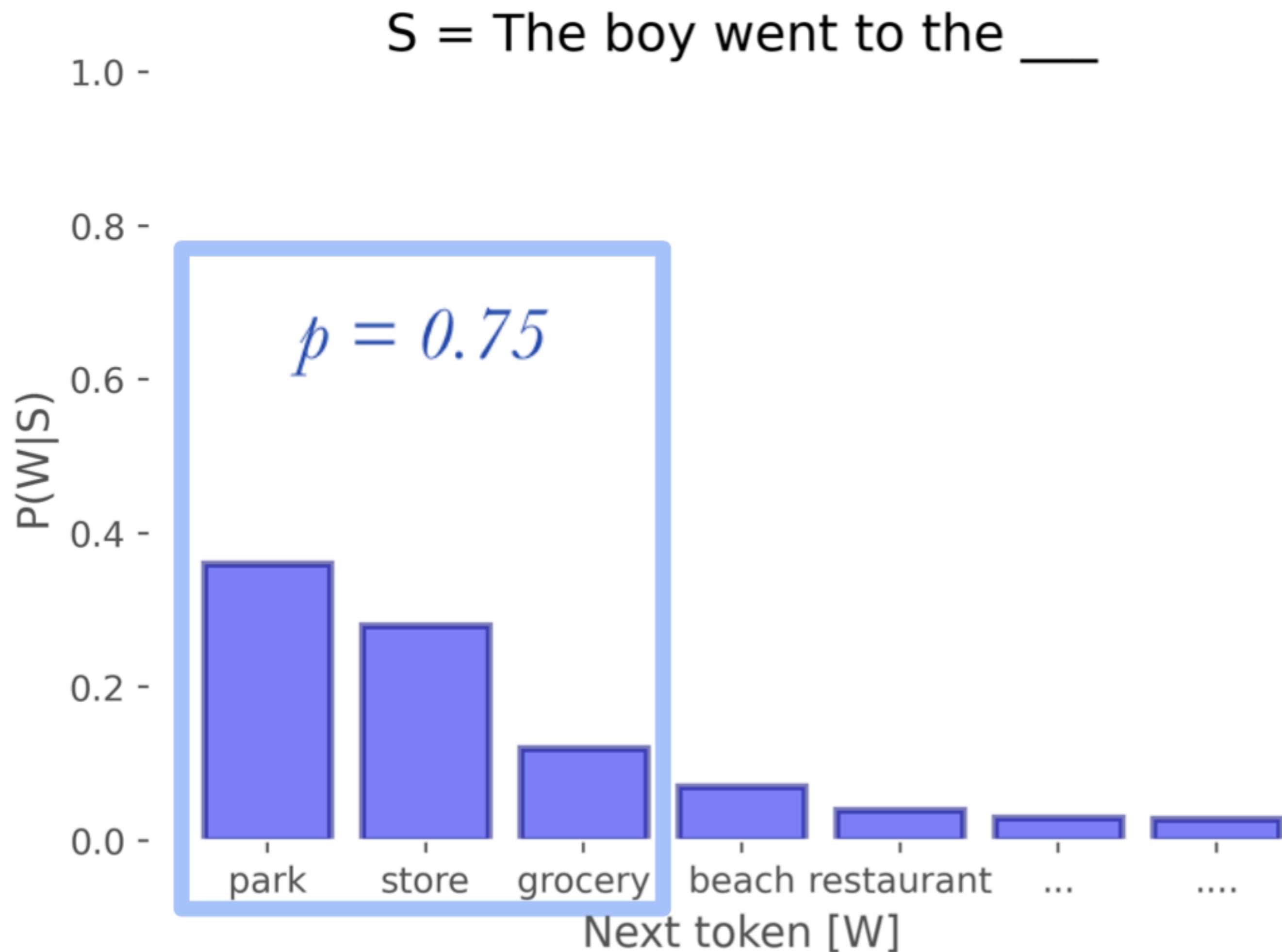


# Nucleus sampling

$$x_t \sim p(x | x_{1:t-1})$$

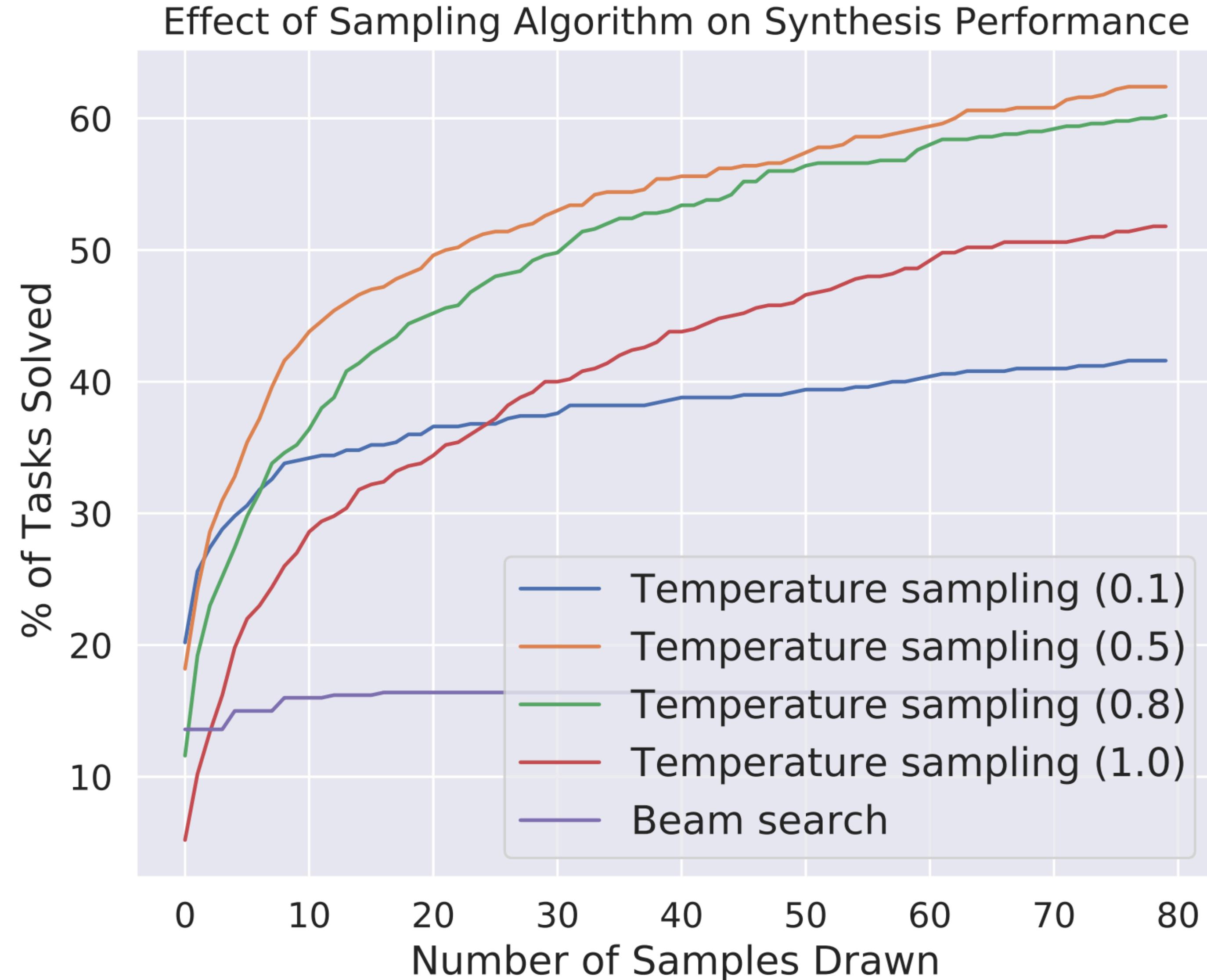
$$p(x | x_{1:t-1}) = \frac{\exp(u_i/T)}{\sum_j \exp(u_j/T)}$$

- Can select the top-k dynamically
- Used in GPT-3, Codex with  $p=0.95$

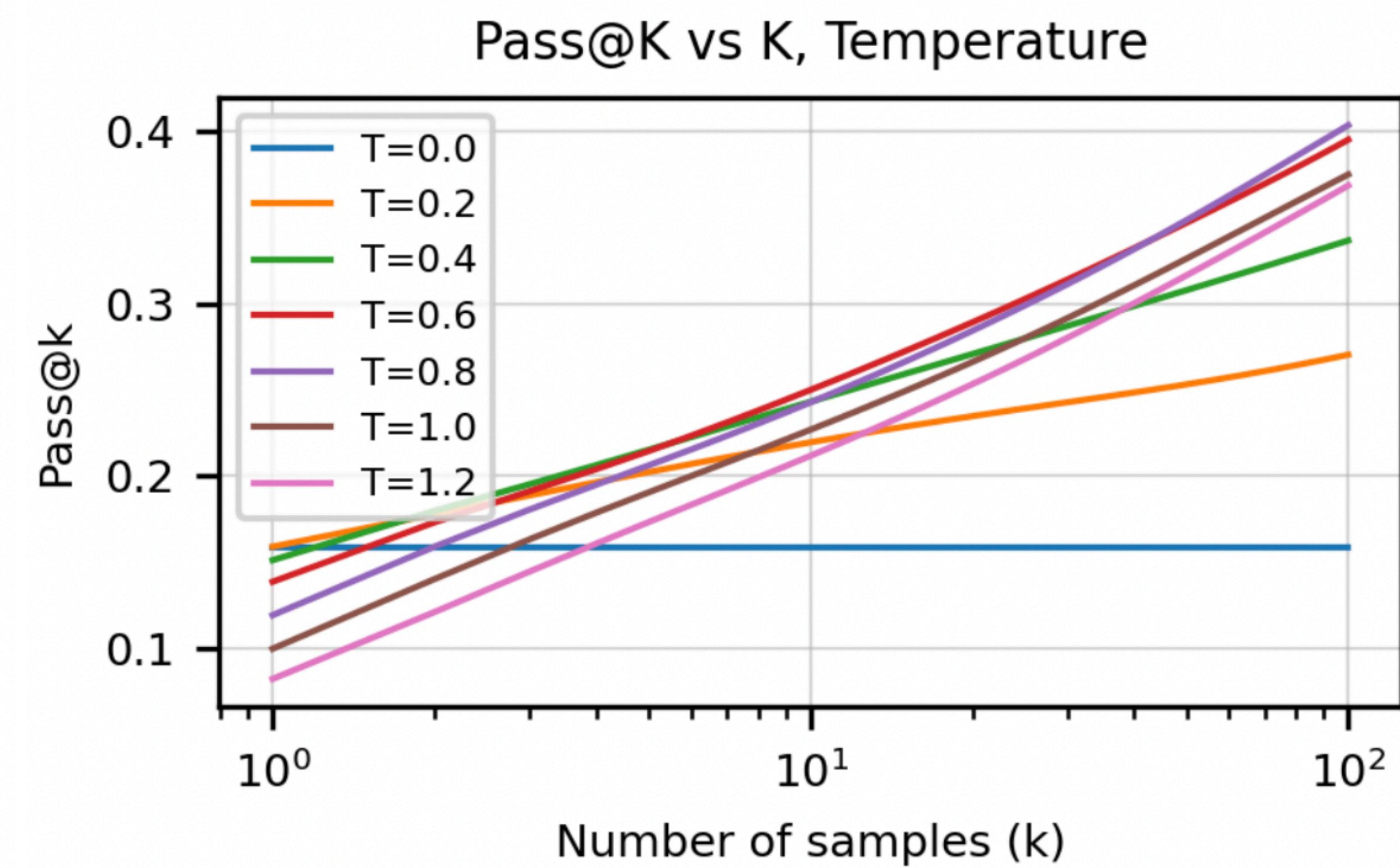
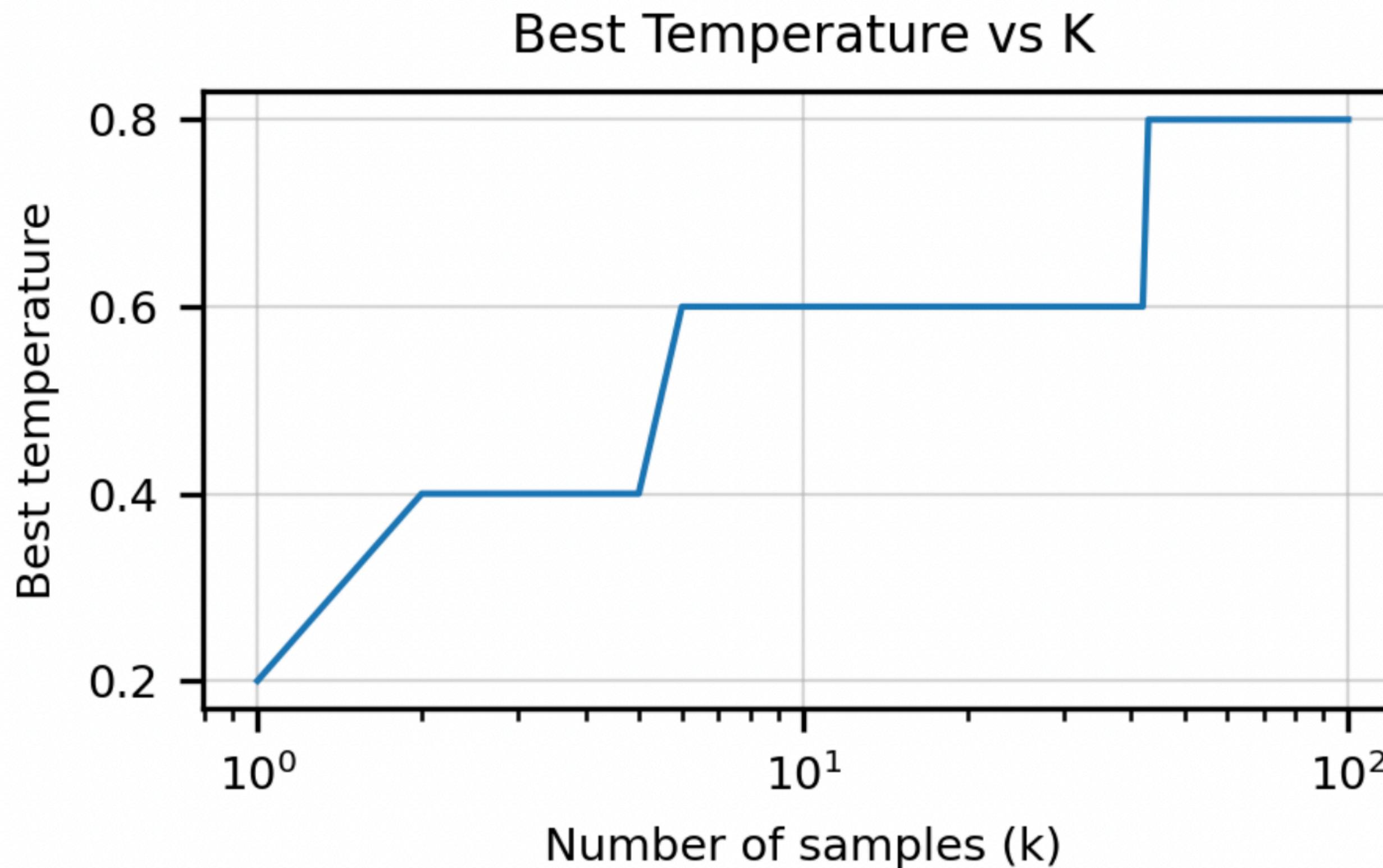


# Beam search vs temperature sampling

- Austin21:
  - High temperature is good for large number of samples
  - Low temperature is when we can evaluate a few
- Beam search works poorly



# Optimal temperature



- Codex: selecting T based on the number of samples (left)
- With an optimal T,  $\text{pass}@k(k)$  goes smoothly

# How to select top-k samples?

- Task: given  $n$  samples, select top- $k$  most relevant

## Test-based approaches:

- oracle reranking: select those that pass all tests
- pre-tests (e.g. 3 unit tests): select those samples that pass all pre-tests

# How to select top-k samples?

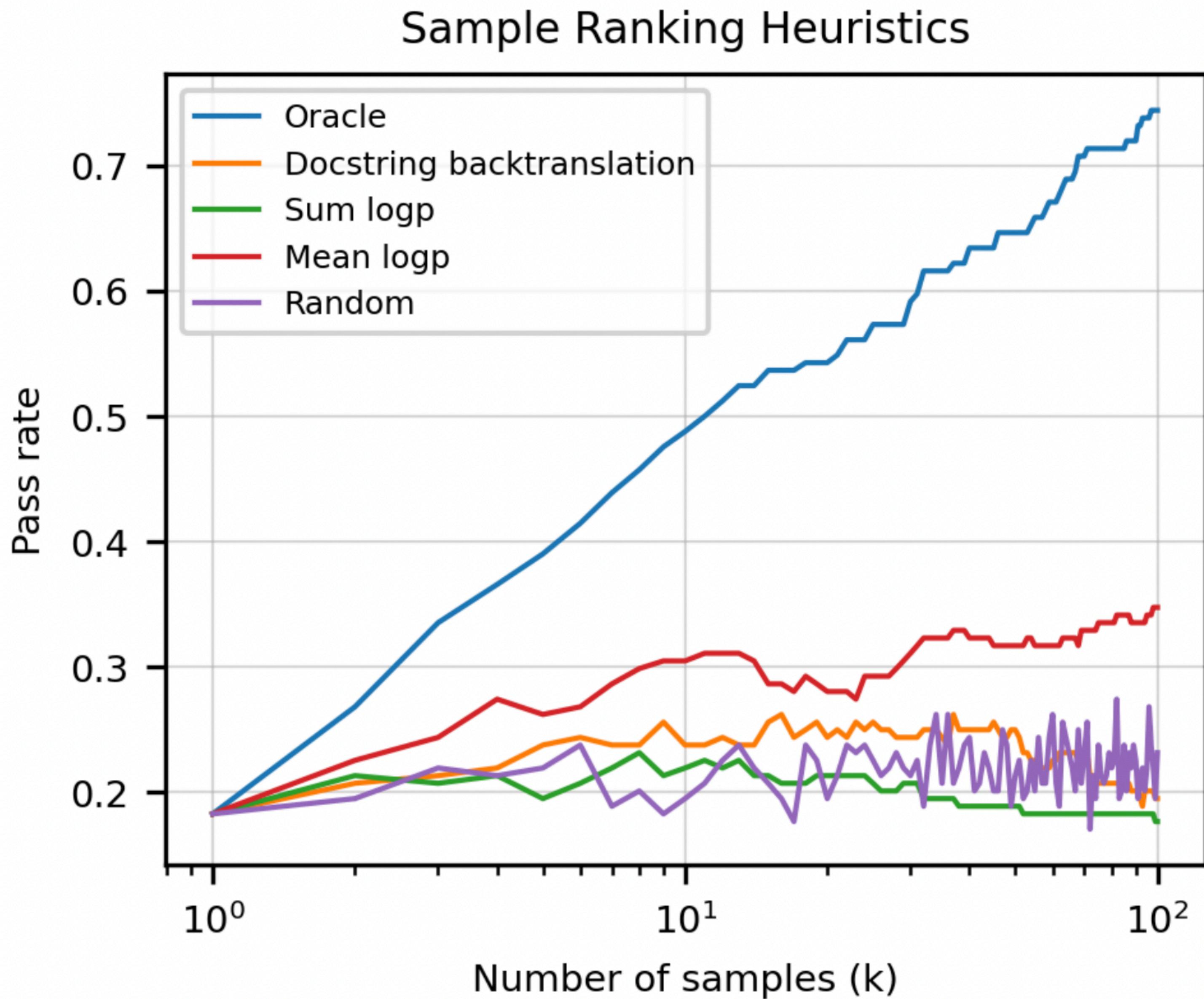
- Task: given n samples, select top-k most relevant

## Model-based approaches:

- sum log  $L(x_1, \dots, x_n) = \prod_i p(x_i | x_{1:t-1})$
- mean log  $L(x_1, \dots, x_n) = \frac{1}{n} \prod_i p(x_i | x_{1:t-1})$
- back-translation  $p(\text{ground truth docstring} | \text{generated sample})$

# How to select top-k samples?

- Mean logp is the best model-based strategy

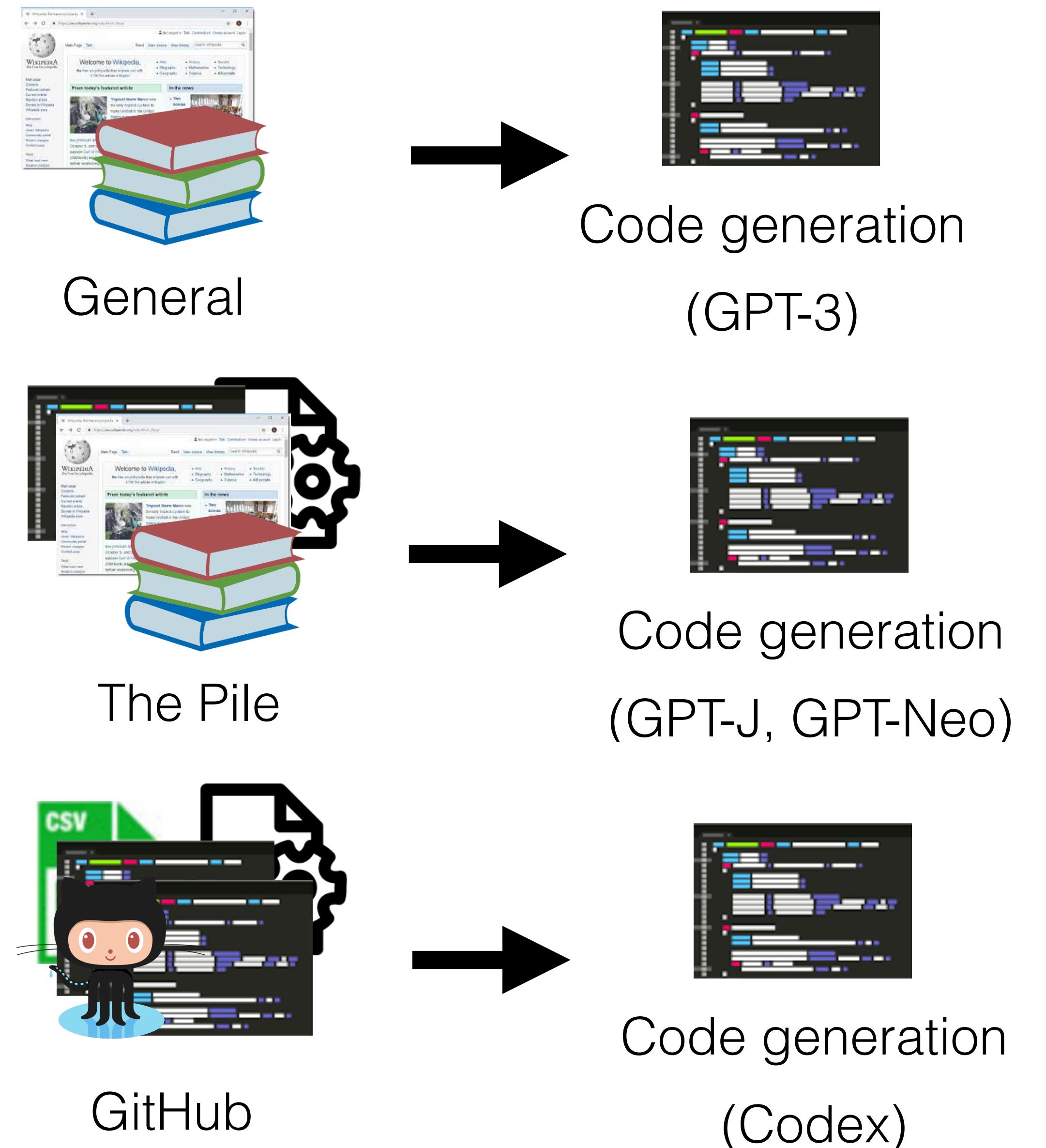


# Next, how to make models better

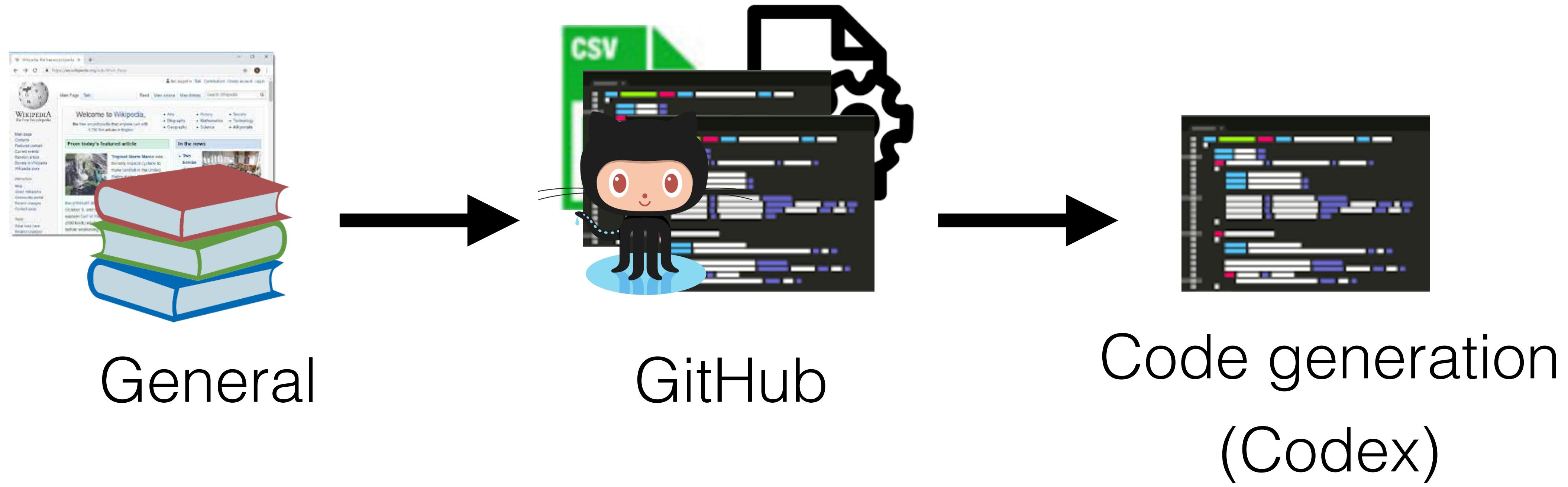
- Supervised fine-tuning
- Divide and conquer
- Write better tasks
- Give feedback

# Using GPT-3 weights for Codex?

- Already understand, which strategy works better: pre-training (or fine-tuning) on large code corpus e.g. GitHub

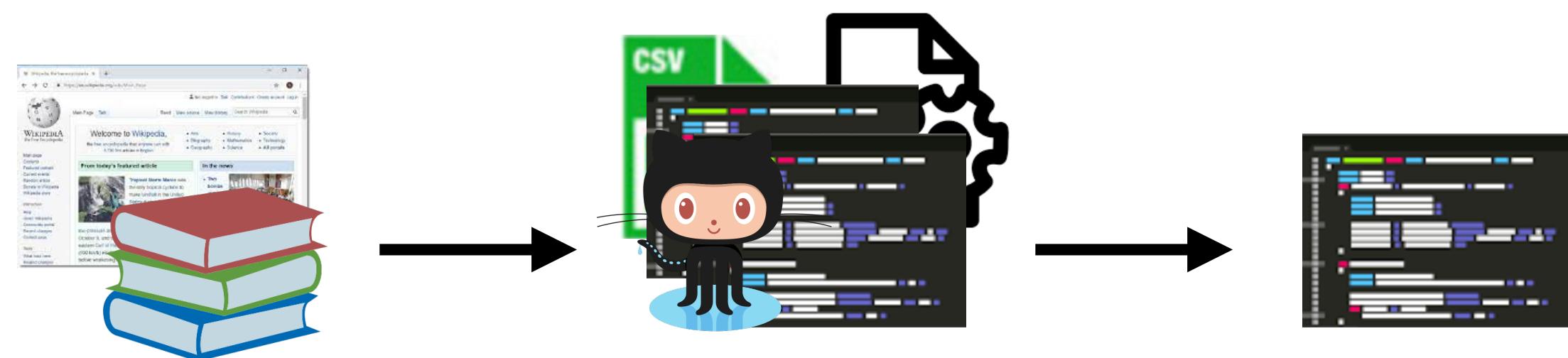


# Using GPT-3 weights for Codex?

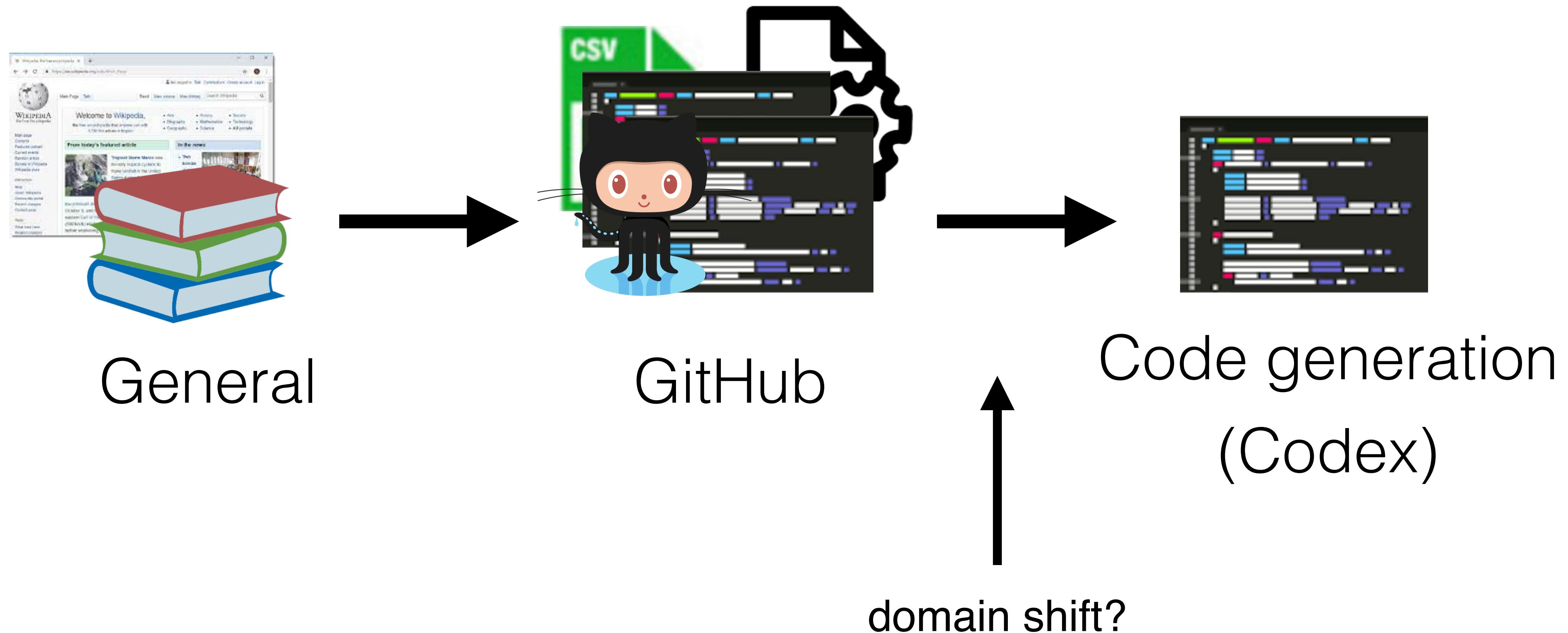


# Using GPT-3 weights for Codex?

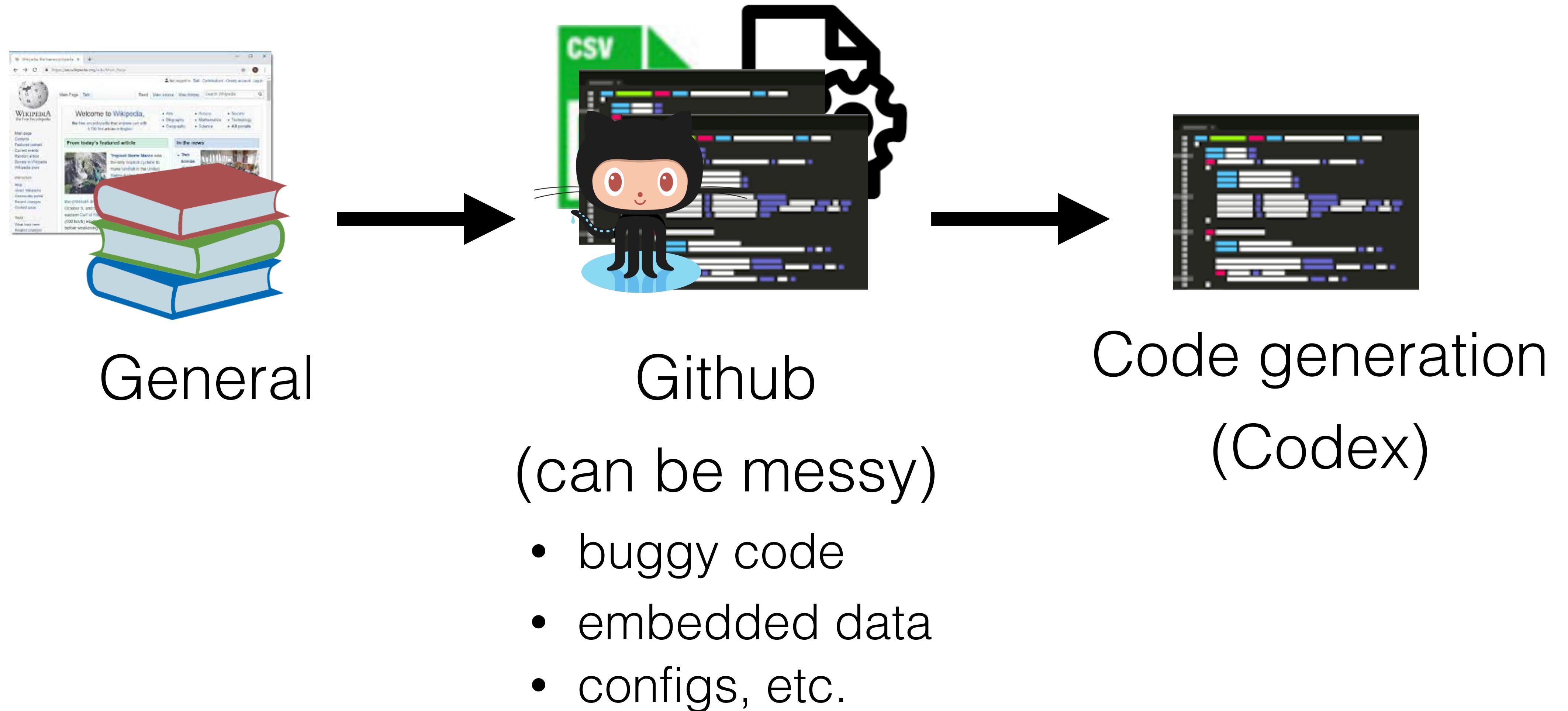
- Codex fine-trained from scratch ~ Codex fine-tuned from GPT-3 in terms of final quality
- Codex fine-trained from scratch < Codex fine-tuned from GPT-3 in terms of convergence speed
- Hence, Codex fine-tuned from GPT-3 was used in all experiments



# Getting better models by supervised fine-tuning



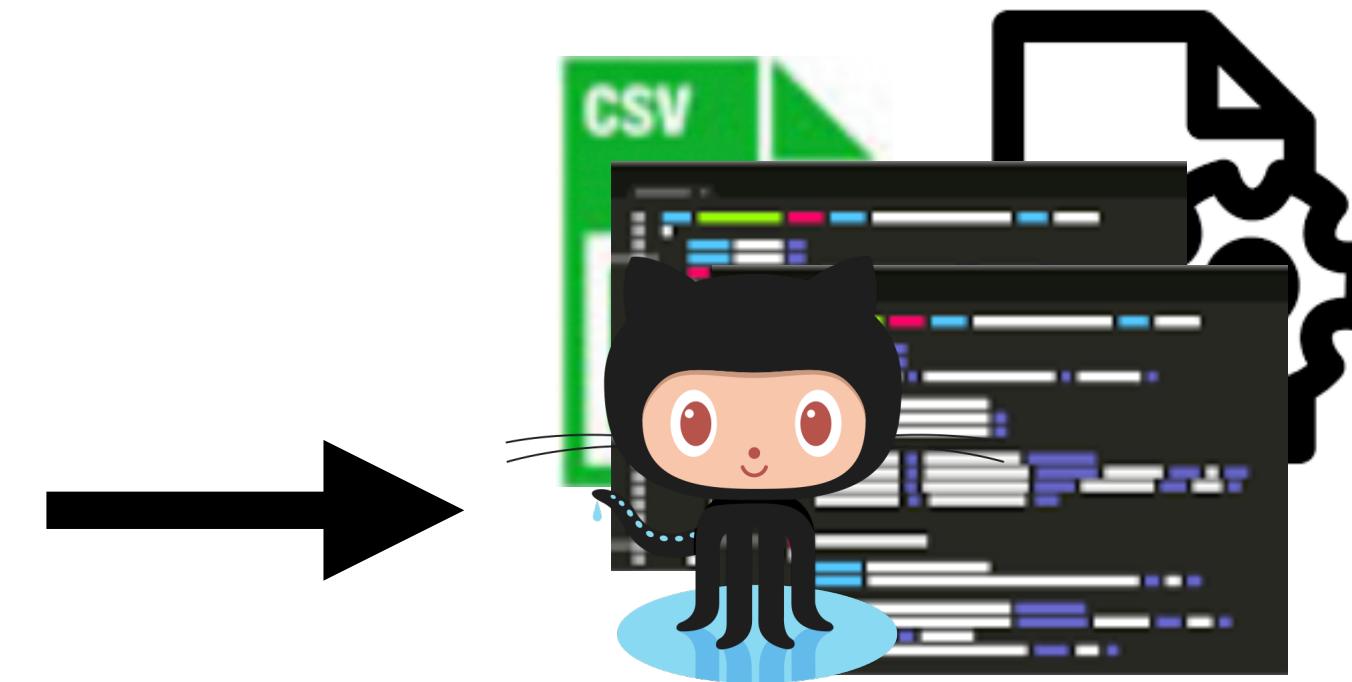
# Getting better models by supervised fine-tuning



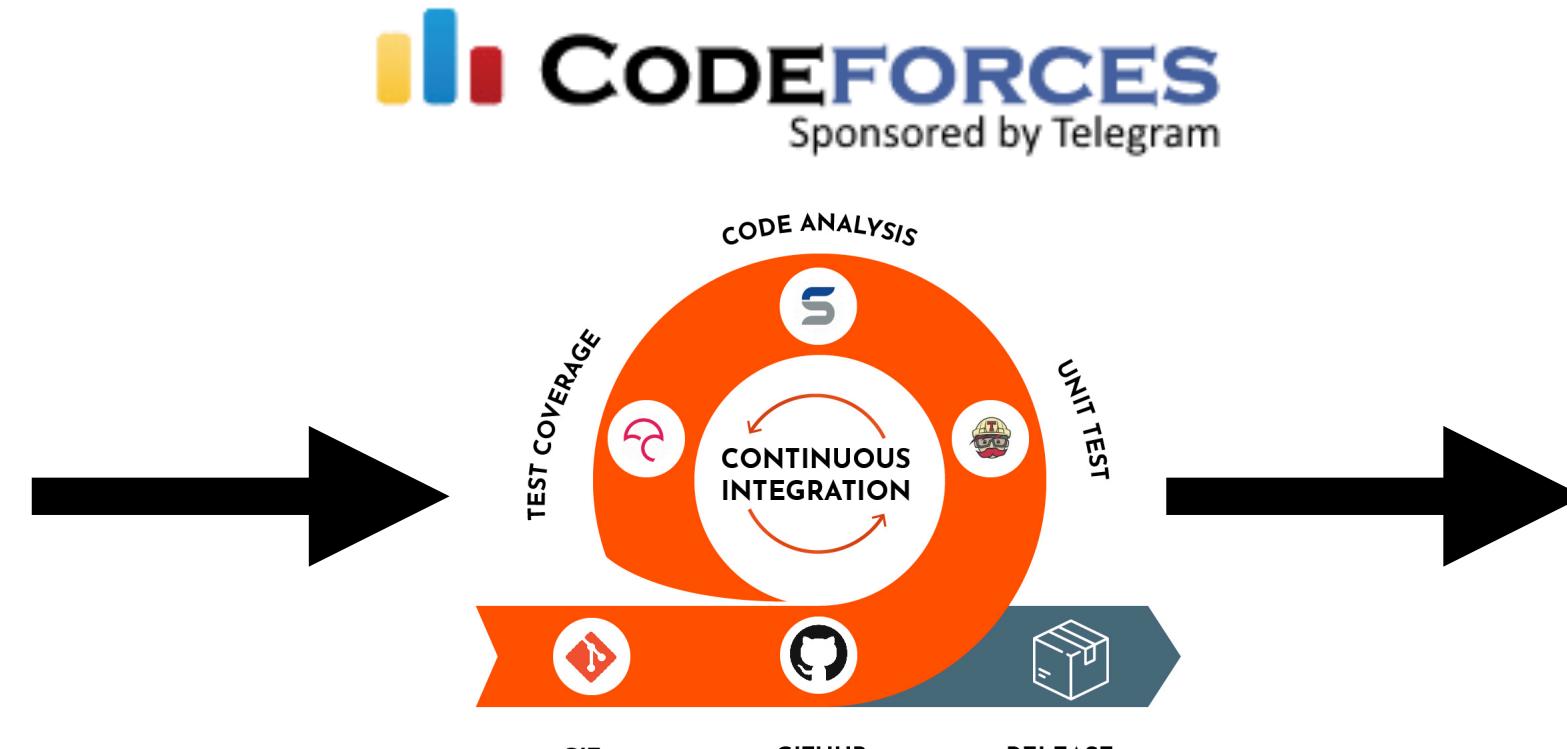
# Getting better models by supervised fine-tune



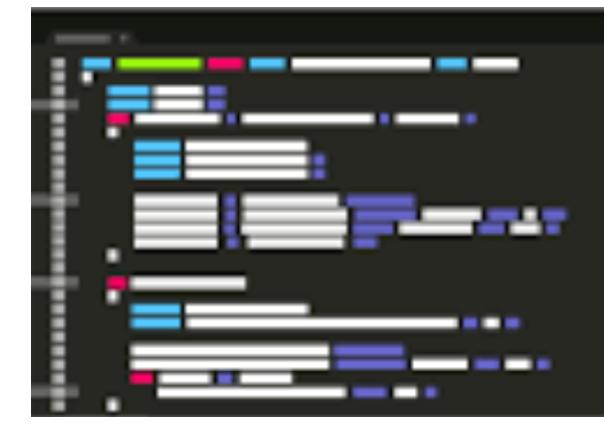
General



Github



Correct code

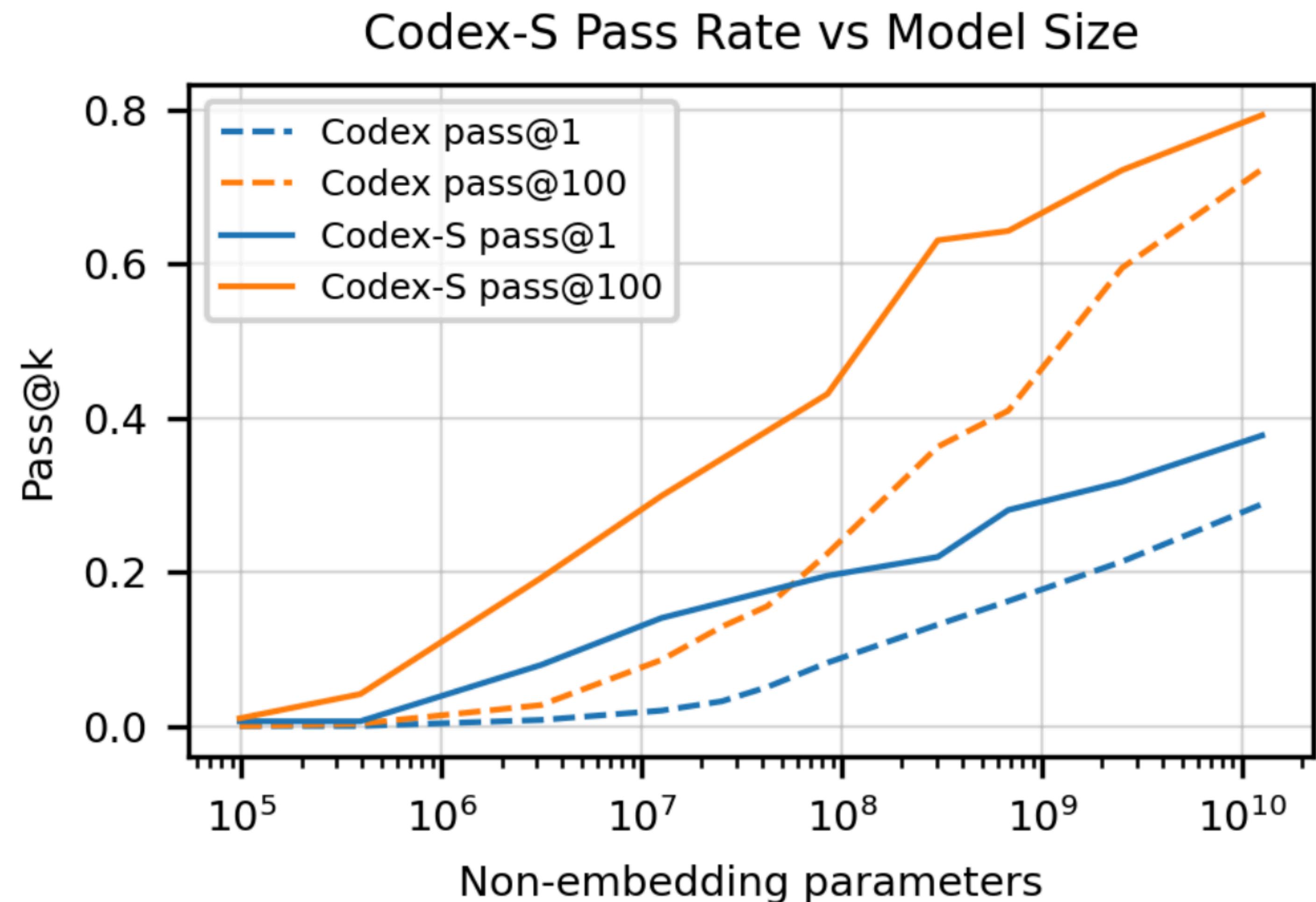


Code generation  
(Codex-S)

# Getting better models by supervised fine-tune

- Github data is messy: configs, data files, scripts, etc.

- Fine-tune on *correctly implemented functions* e.g. from programming contests and CI pipelines



# Getting better models by writing simpler tasks

- Experiments from Austin21:

	<b>Theme</b>	<b>Examples</b>
<b>Highest-performing problems</b>	Single operations	<p>Write a function to remove all whitespaces from a string.</p> <p>Write a python function to find the maximum of two numbers.</p>
	Common “coding interview” type questions	<p>Write a function to merge multiple sorted inputs into a single sorted iterator</p>

# Getting better models by writing simpler tasks

- Experiments from Austin21:

<b>Lowest-performing problems</b>	<p>Problems demanding multiple constraints or multiple sub-problems</p>	<p>Write a function to find the maximum difference between the number of 0s and number of 1s in any sub-string of the given binary string <i>(Sub-problems: count 0s and 1s, find difference, find max across all sub-strings)</i></p> <p>Write a function to find the longest palindromic subsequence in the given string <i>(Sub-problems: keep track of mirror-imaged letters, find palindromes, find longest one)</i></p>
-----------------------------------	---	---

# Getting better models by writing simpler tasks

1. “remove all instances of the letter e from the string”

```
s = s.replace("e", "")
```

...

3. “convert the string s to lowercase”

```
s = s.lower()
```

...

5. “removes all vowels from the string”

```
s = "".join(char for char in s if
            char not in "aeiouAEIOU")
```

...

# Getting better models by writing simpler tasks

```
def string_manipulation(s: str):  
    """
```

*This function takes a string as input, then returns  
the result of performing  
the following sequence of manipulations on that  
string:*

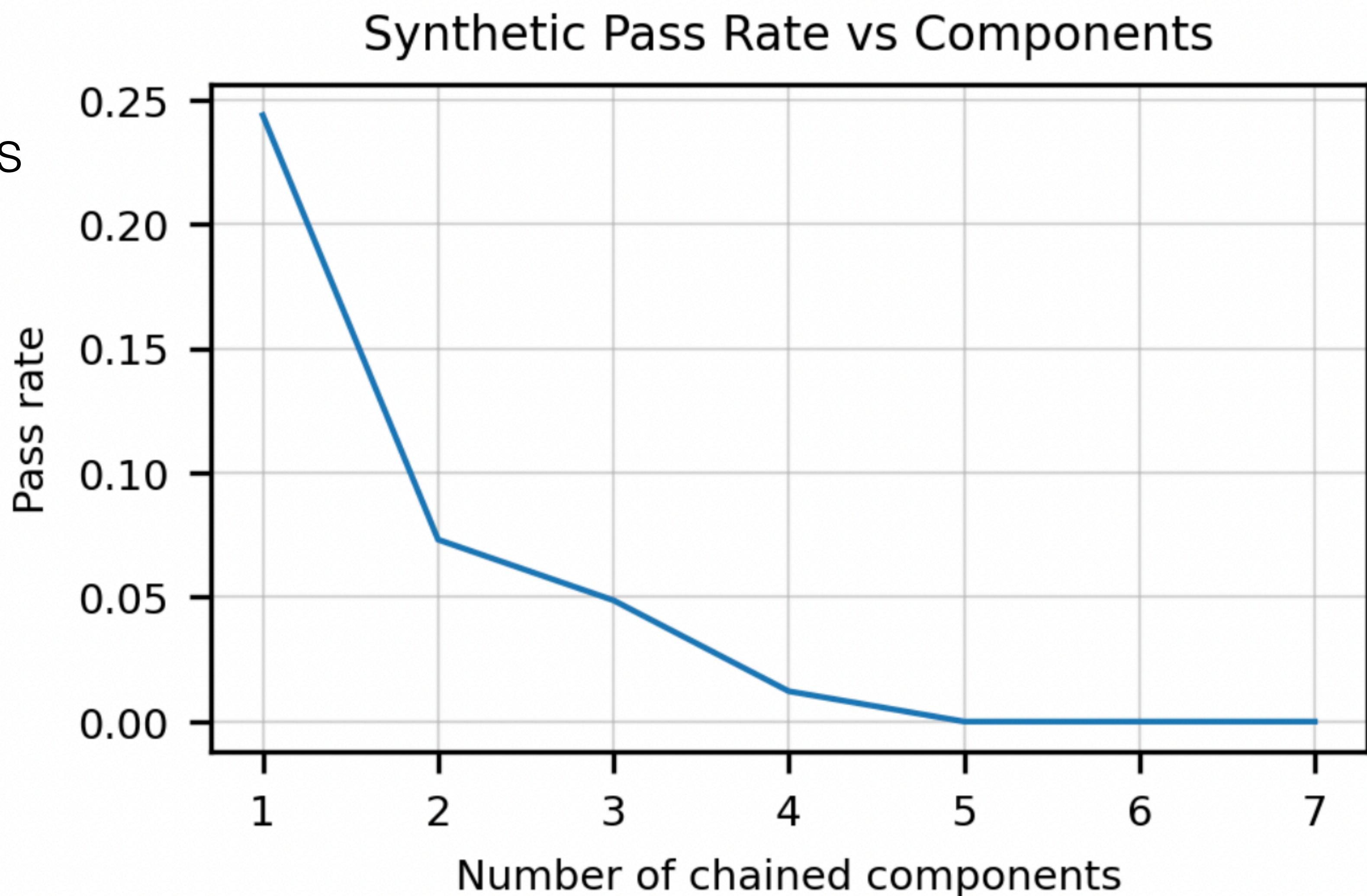
*-make every other character in the string uppercase  
-replace spaces with triple spaces*

```
"""
```

```
s = "".join(char.upper() if i % 2 == 0 else char  
            for i, char in enumerate(s))  
s = s.replace(" ", "   ")  
return s
```

# Getting better models by writing simpler tasks

- The quality drops exponentially with the number of sub-problems



# Getting better models by writing simpler tasks

- Divide and conquer approach

Write a function that does  
`<Task1>` and `<Tasks2>`

Write a function `f1` that does `<Task1>`

Write a function `f2` that does `<Task2>`

Write a function that combines `f1` and `f2`

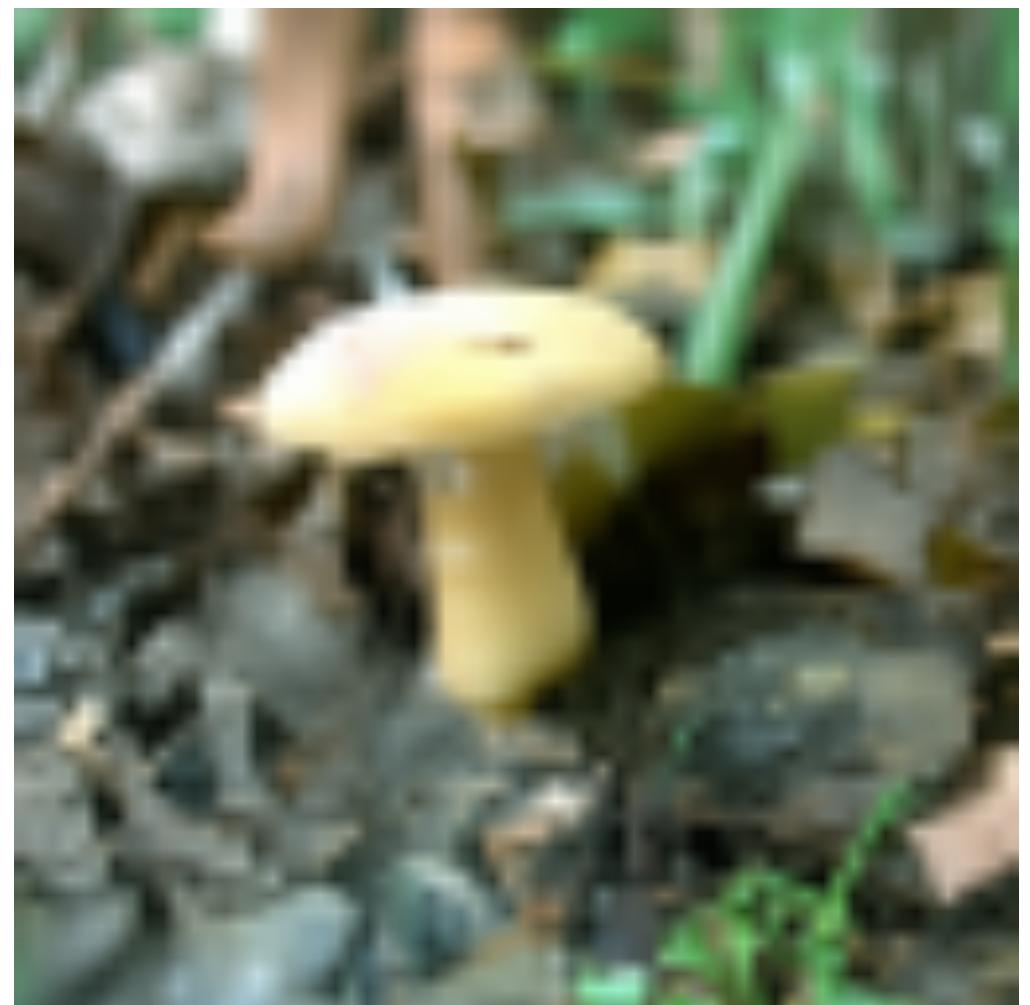
# Getting better models by writing better inputs



Aran Komatsuzaki  
@arankomatsuzaki

...

- manipulation of CLIP (text2image generation):



A mushroom



A 8K HUHD mushroom

When you generate images with VQGAN + CLIP, the image quality dramatically improves if you add "unreal engine" to your prompt.

People are now calling this "unreal engine trick" lol

e.g. "the angel of air. unreal engine"



12:02 AM · Jun 1, 2021 · Twitter Web App

# Getting better models by writing better inputs

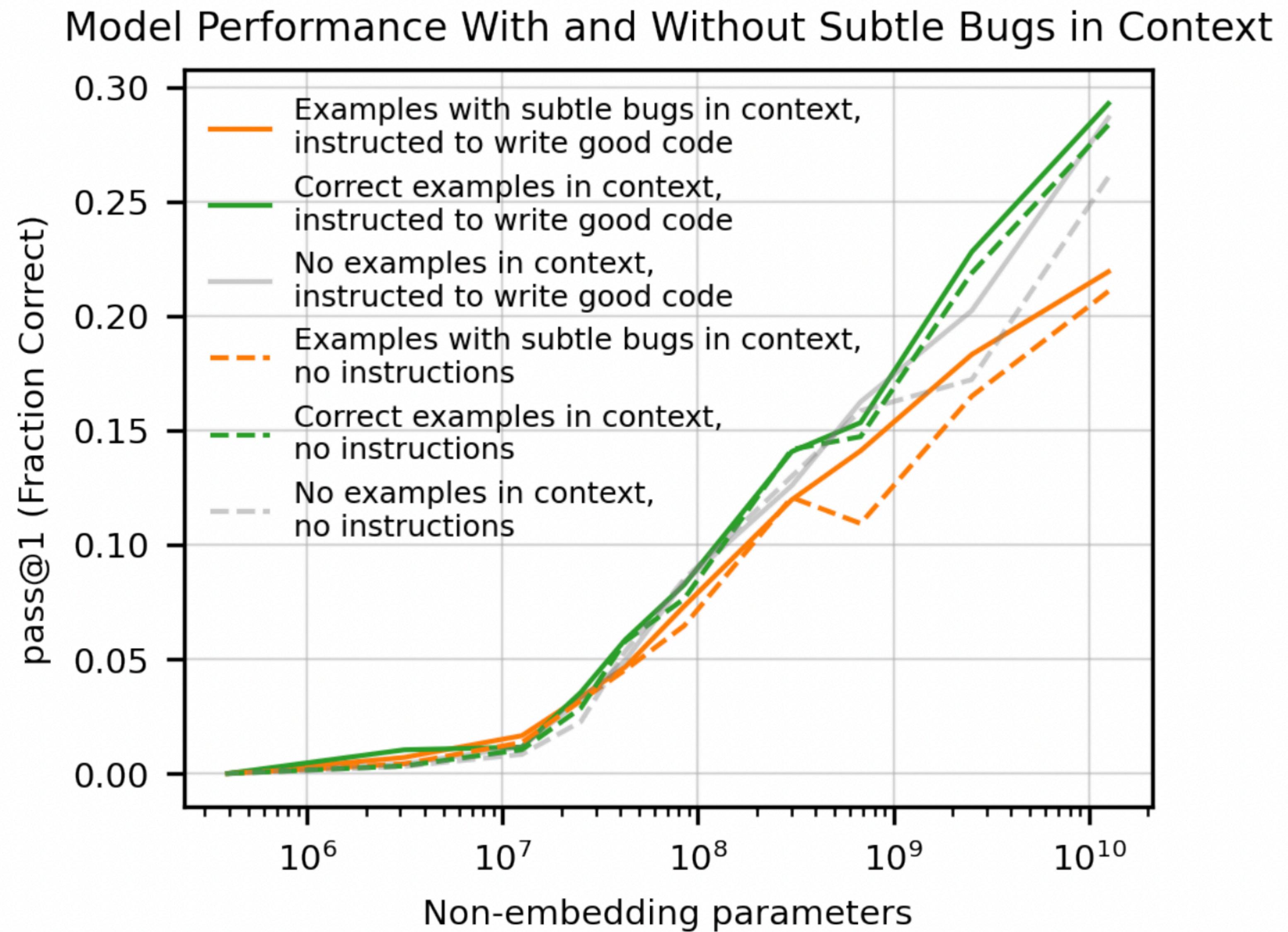
- manipulation of code:

```
def is_prime(a):  
  
    # write a function which checks if 'a' is a  
    prime number.
```

```
def is_prime(a):  
  
    # write a function which checks if 'a' is a  
    prime number. And also, please, write good  
    code.
```

# Getting better models by writing better inputs

- Adding some instructions to write good code can improve quality
- Bugs in input can reduce quality



# Getting better models by human feedback

user

I need to write a Python function called **snake\_to\_camel**. Here's a description: 'Write a function to convert a snake case string to camel case string'.

user

```
assert  
snake_to_camel('python_program') ==  
'PythonProgram'
```

user

Close, but you need to replace the underscore with an empty string.

(f) success (model passes tests)

(a) human provides a description

(b) model asks for few-shot examples

Can you give me an example?

model

(c) user provides an example (can be automatic)

```
def snake_to_camel(str):  
    str = str.title()  
    str = str.replace('_', '')  
    return str
```



model

(d) model attempts to solve the problem (but model fails tests)

(e) human provides feedback

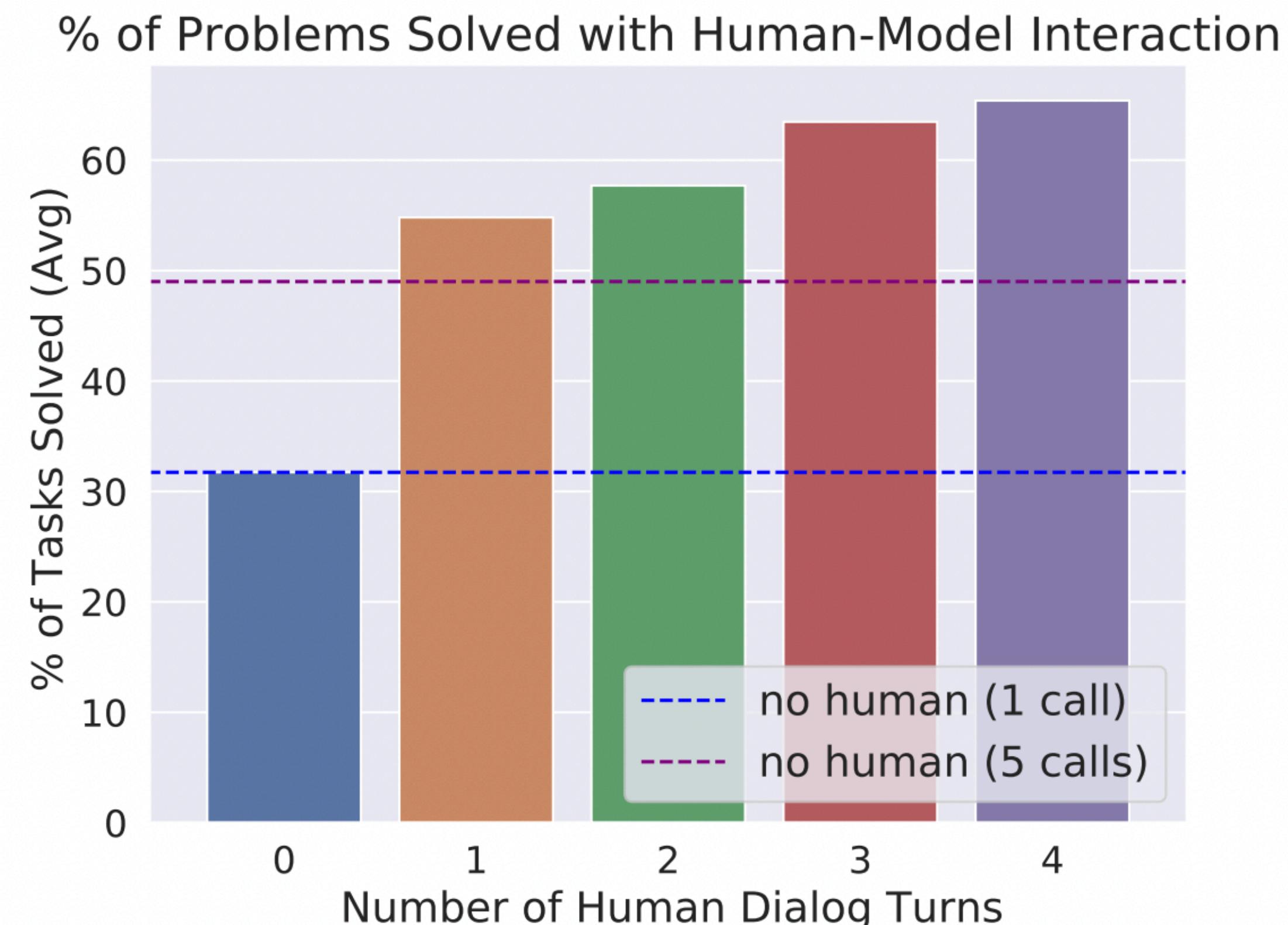
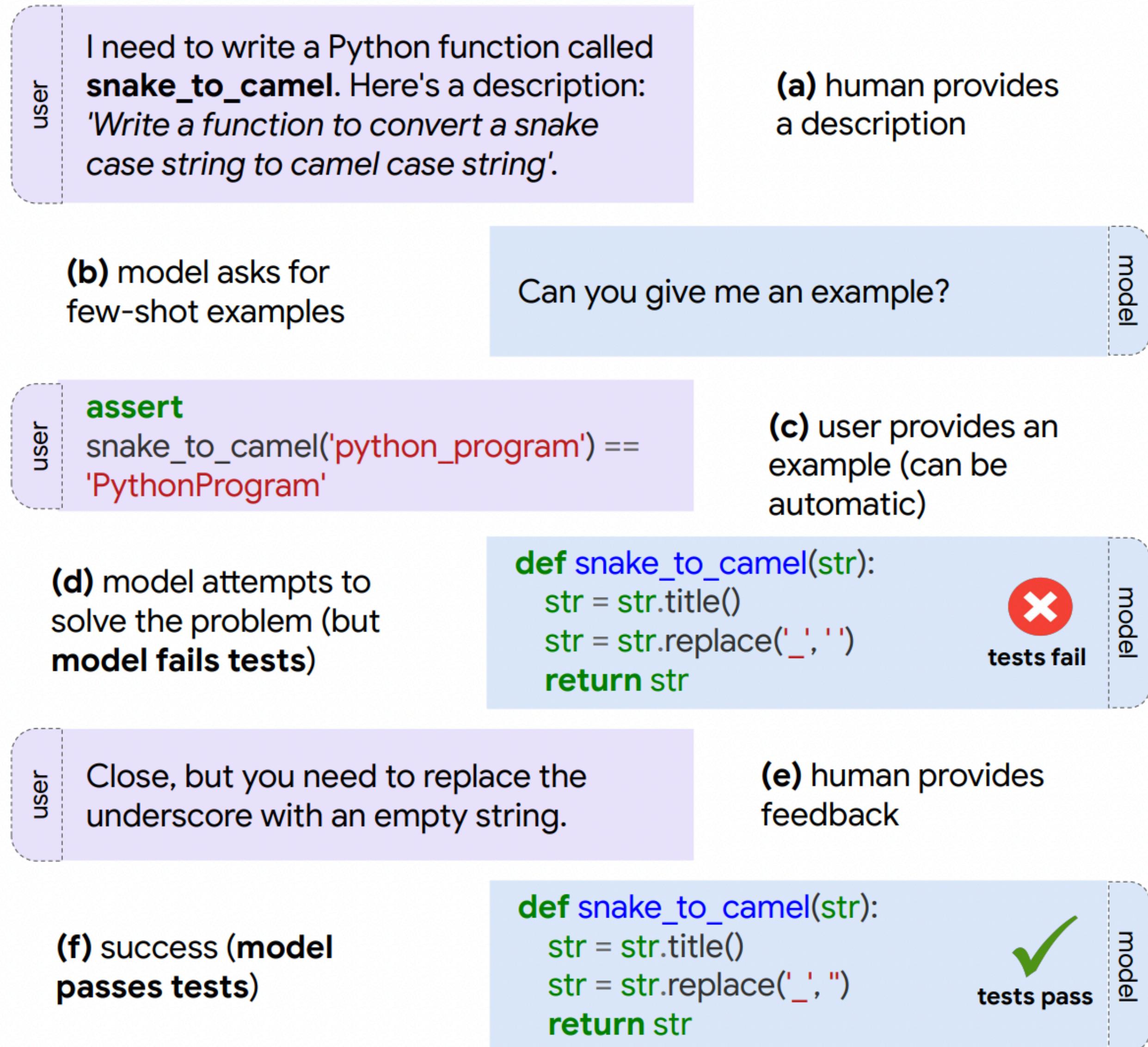
```
def snake_to_camel(str):  
    str = str.title()  
    str = str.replace('_', '')  
    return str
```



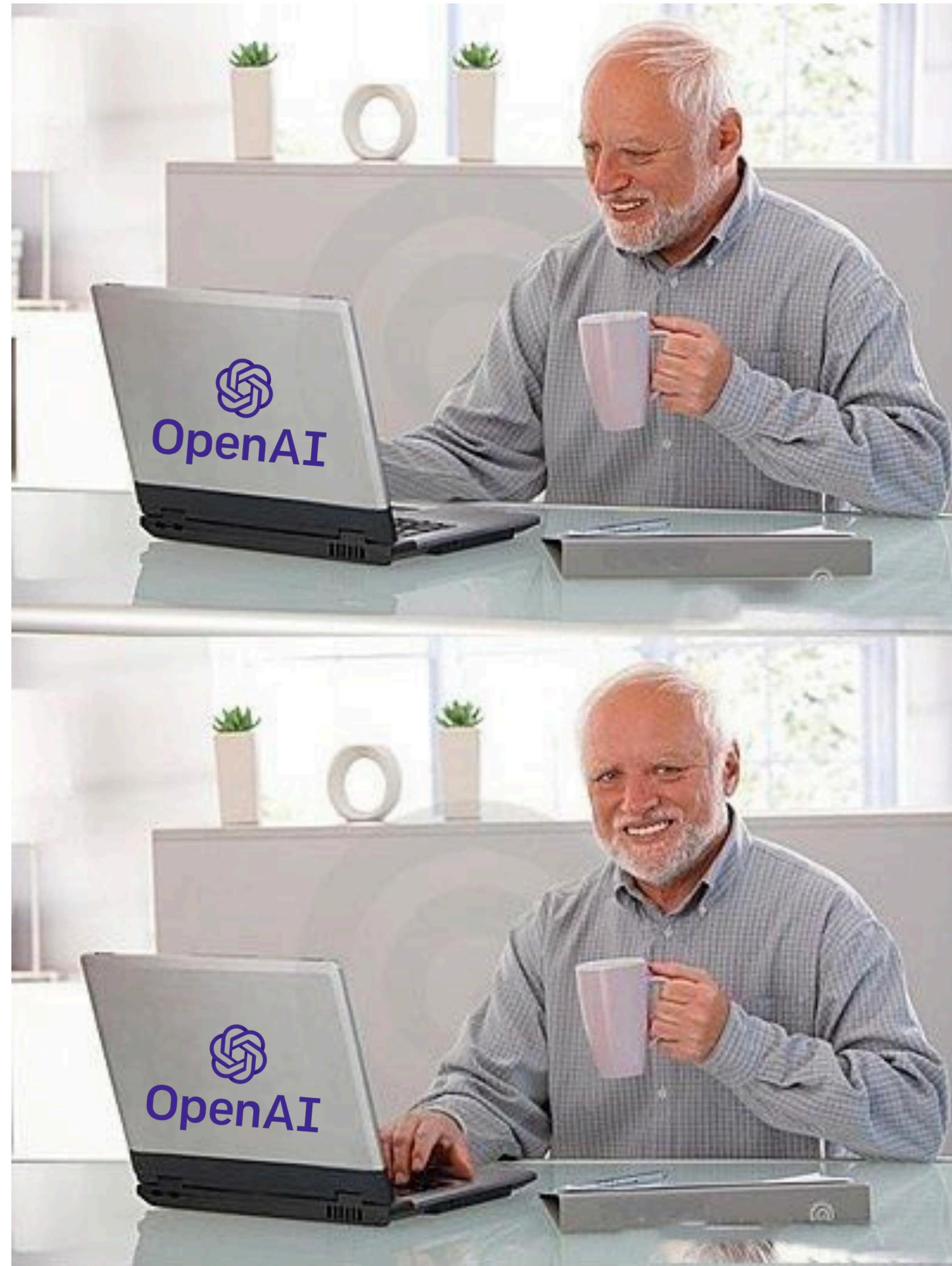
model

- Austin21: A developer can even try to interact with a model
- Requirement: Q/A data in pretraining

# Getting better models by human feedback



# Limitations of Codex



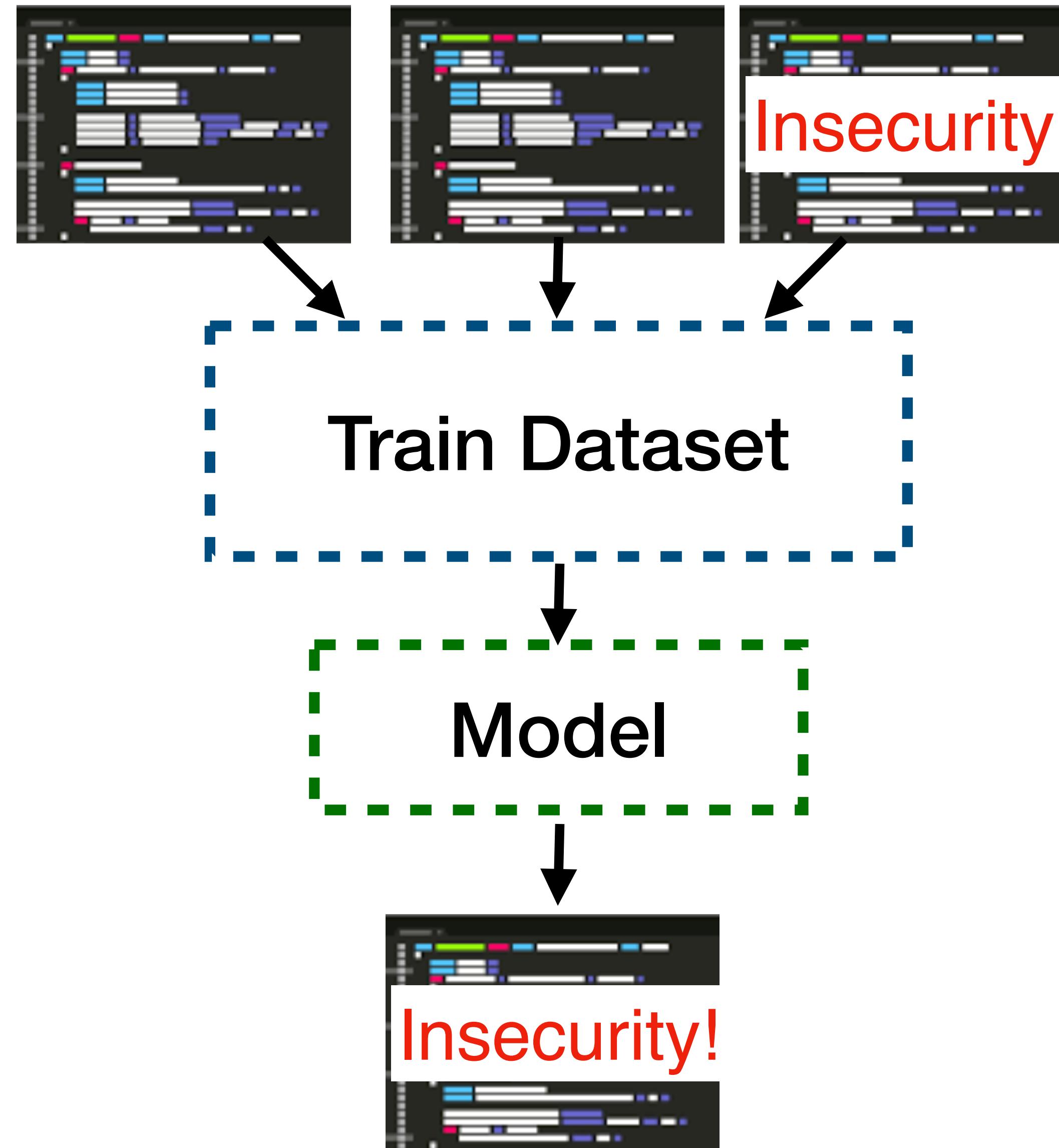
# Limitations of Codex

- Subtle bugs

```
def is_prime(d):
    # check if a number d is prime
    if d == 2 or d == 3 or d == 5
    or d == 7 or d == 11:
        return True
    else:
        return False
```

# Limitations of Codex

- Subtle bugs
- Insecure code



# Limitations of Codex

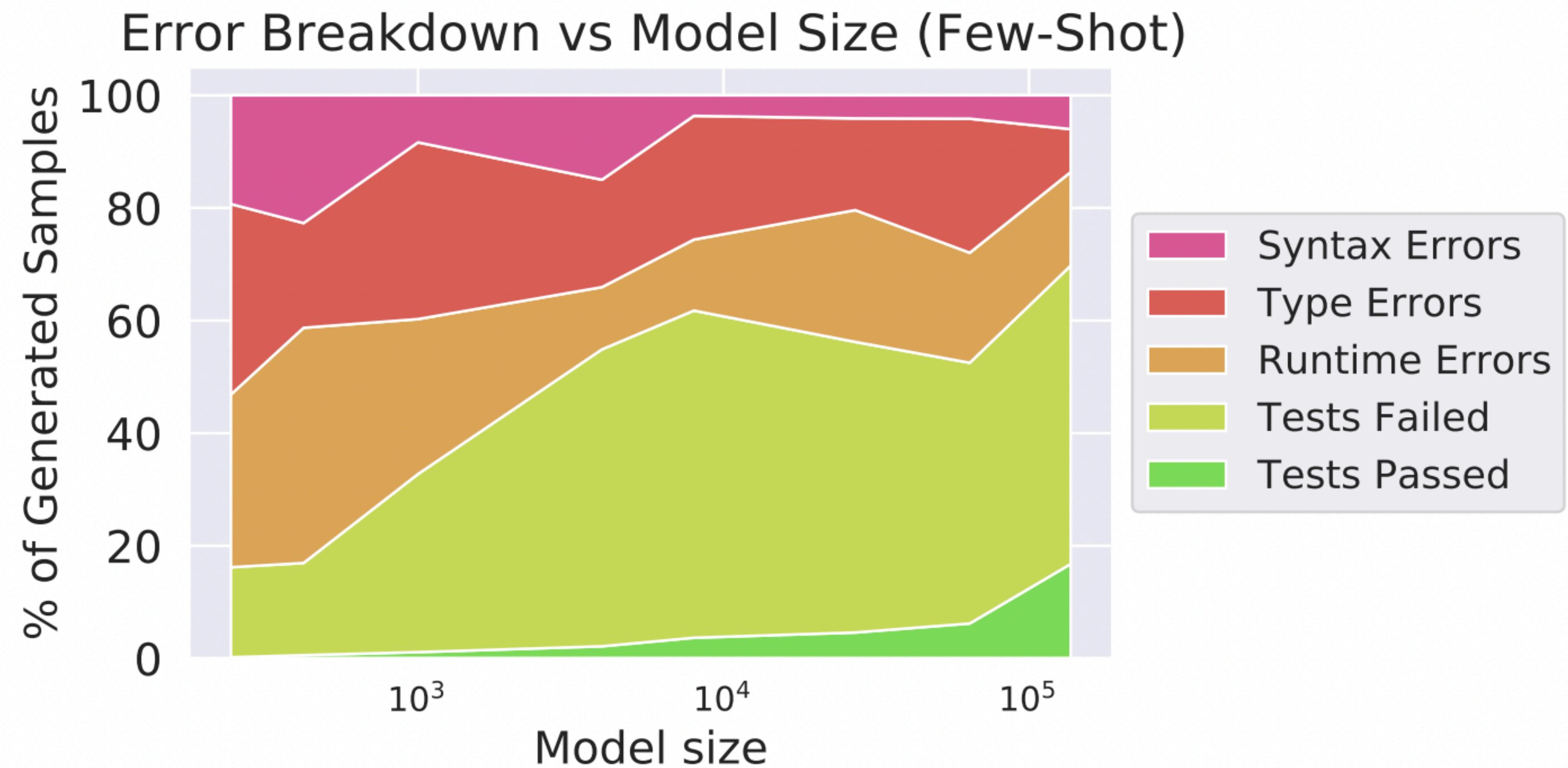
- Subtle bugs
- Insecure code
- Syntax errors
  - Some samples from the model may not compile/cause runtime errors:

```
def f(it:  
      pass
```

```
ar = []  
for it in ar:  
    ar.append(it)
```

# Limitations of Codex

- Subtle bugs
- Insecure code
- Syntax errors



Experiment from Austin21 et al.

# Limitations of Codex

- Subtle bugs
- Insecure code
- Syntax errors
- API diversity

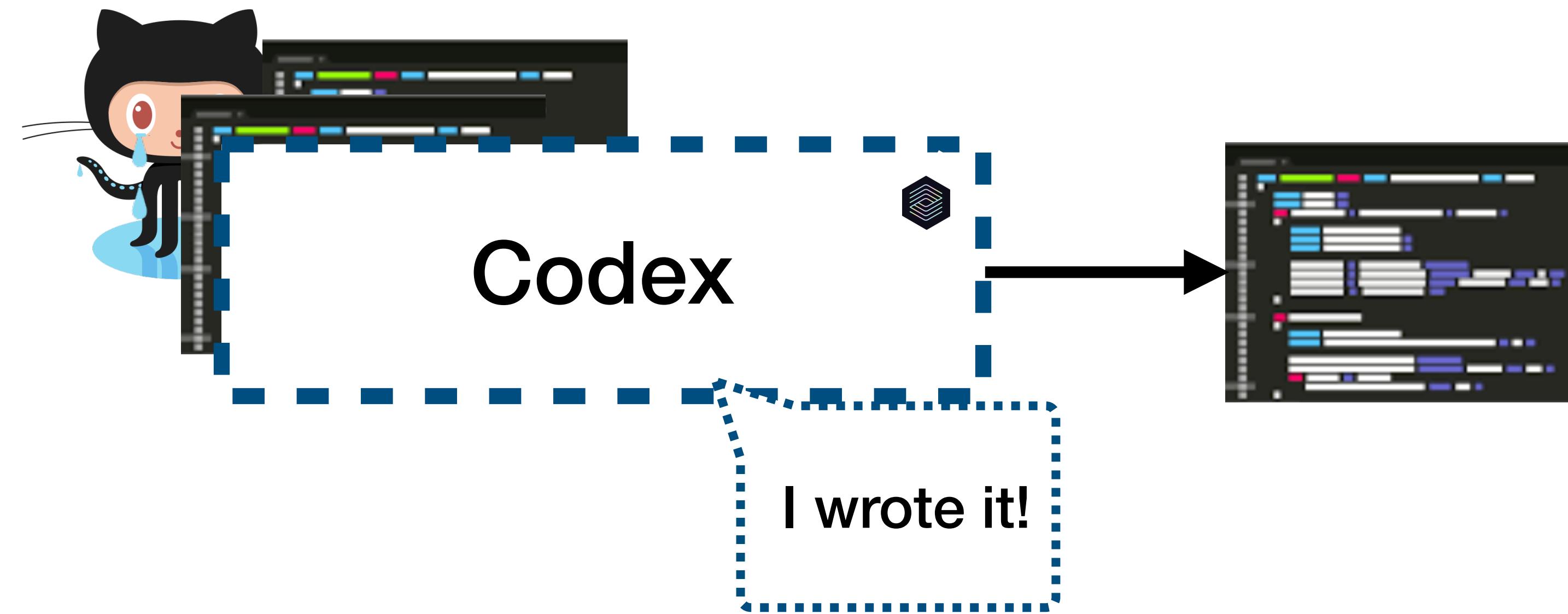
```
import torch
import tensorflow as tf
import jax
```



# Limitations of Codex

- Subtle bugs
- Insecure code
- Syntax errors
- API diversity
- Legal Issues

Fair use or copying?



# Conclusions;

- Functional correctness is better than BLEU for code generation tasks
- Pre-training on GitHub —> Fine-tuning on non-buggy code data
- Code generation can be improved with subtle ways, such as user feed-back or tuning the input
- GPT for code can be improved