OpenAI’s Codex

Data Circles Journal Club 9-14-22

Evaluating Large Language Models Trained on Code

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

OpenAI introduces **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 **GitHub Copilot**. On **HumanEval**, a new evaluation set we release to measure **functional correctness** for synthesizing programs from docstrings, our **model solves 28.8%** of the problems, while GPT-3 solves 0% and GPT-J solves 11.4%.

Intro & Related Work

Translate between NL (natural language) and PL (programming language) code

Tasks

- Explain code, generate documentation or function names (PL → NL)
- Code generation or search (NL → PL)

Examples

- CodeNN, CodeSum, code2seq - C#, SQL, Java
- CodeBERT - code search in 6 PL
- PyMT5 - Python

Evaluation Framework

HumanEval

- Handwritten evaluation set
- 164 programming problems with signature, docstring, body, unit tests
- Assess language comprehension, algorithms, simple math

HumanEval

```python
def incr_list(l: list):
    """Return list with elements incremented by 1.
    >>> incr_list([1, 2, 3])
    [2, 3, 4]
    >>> incr_list([5, 3, 5, 2, 3, 3, 9, 0, 123])
    [6, 4, 6, 3, 4, 4, 10, 1, 124]
    """
    return [i + 1 for i in l]
```

```python
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)
```

HumanEval

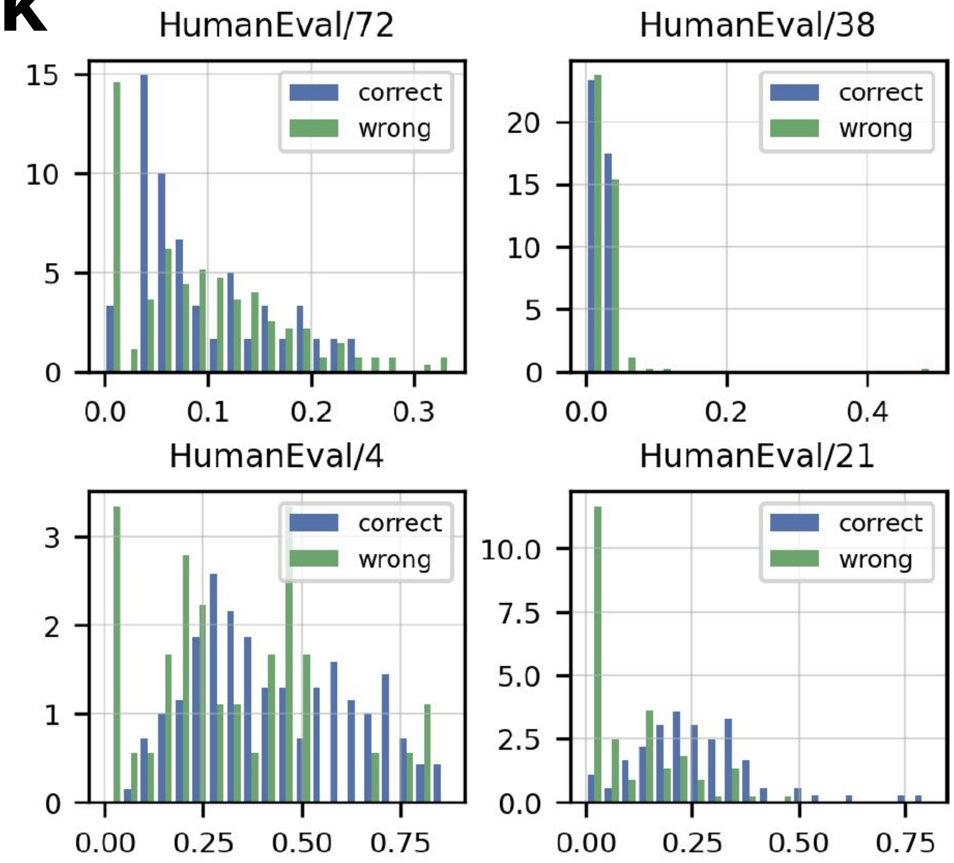
```python
def encode_cyclic(s: str):
    """
    returns encoded string by cycling groups of three characters.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group. Unless group has fewer elements than 3.
    groups = [(group[1:] + group[0]) if len(group) == 3 else group for group in groups]
    return "".join(groups)

def decode_cyclic(s: str):
    """
    takes as input string encoded with encode_cyclic function. Returns decoded string.
    """
    # split string to groups. Each of length 3.
    groups = [s[(3 * i):min((3 * i + 3), len(s))] for i in range((len(s) + 2) // 3)]
    # cycle elements in each group.
    groups = [(group[-1] + group[:-1]) if len(group) == 3 else group for group in groups]
    return "".join(groups)
```

Evaluation Framework

Functional Correctness

- **pass@k** metric
- Generate at least
  1 correct code passes unit test

BLEU score unreliable indicator of Functional Correctness



Code Fine-Tuning

Pre-trained GPT-3 12B params

Data Collection

- 54 million public GitHub repos
- 159 GB Python files filtered

Faster Convergence

Different Tokenizer

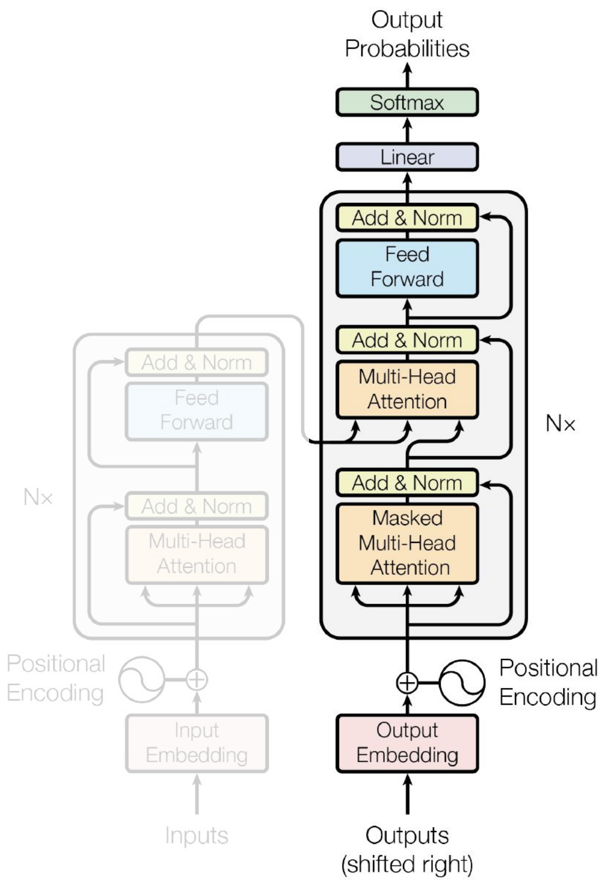
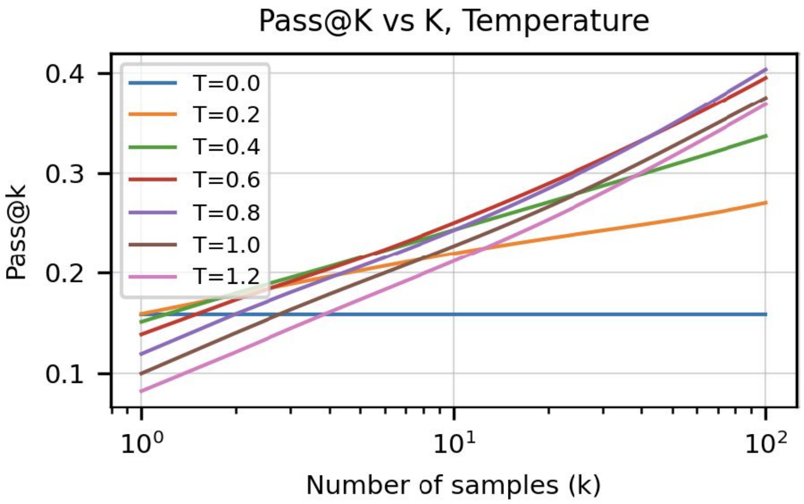


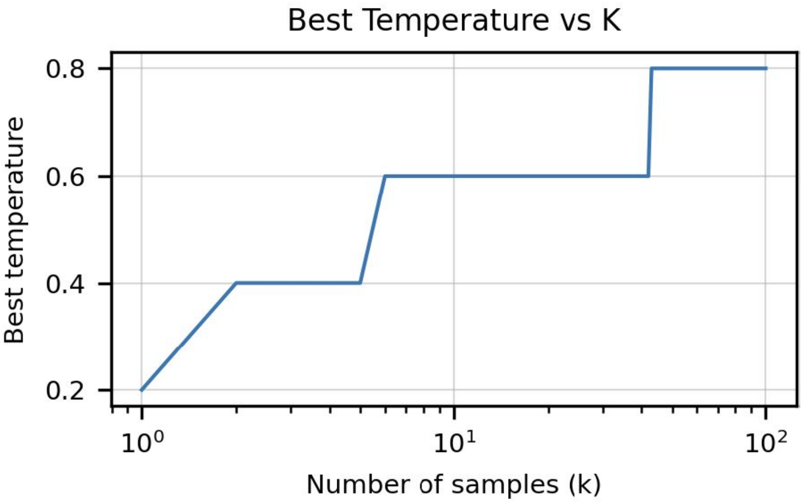
Figure 1: The Transformer - model architecture.

Samples & Temperature Parameters

k=1 T=0.2

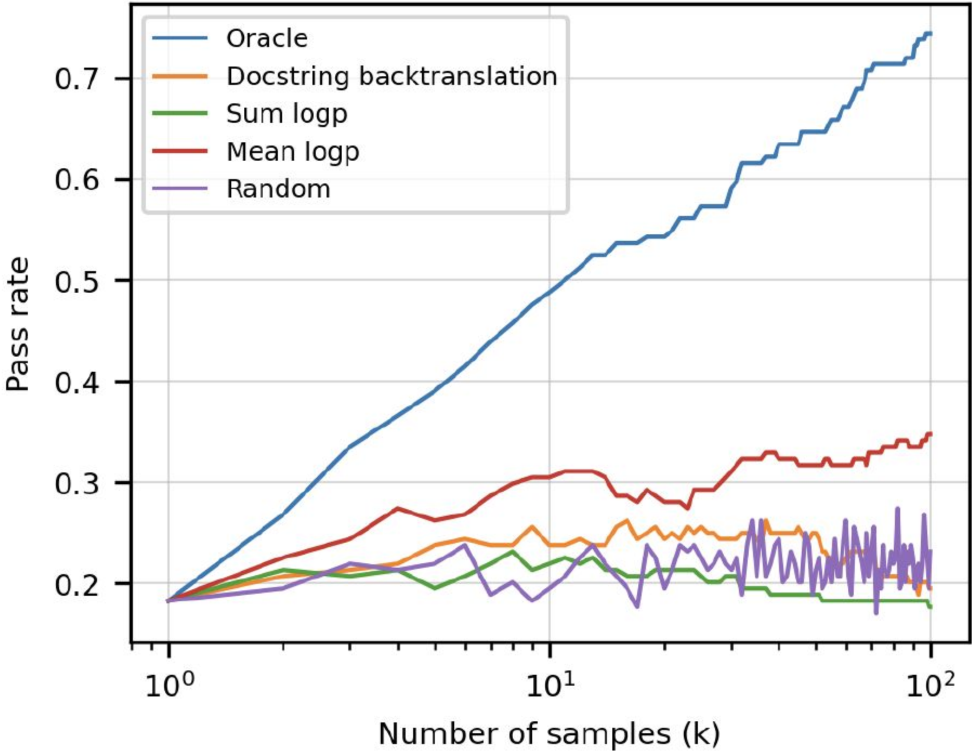
k=100 T=0.8





Ranking Heuristics

Find best from k samples without oracle?



Comparative Analysis on HumanEval

Open Source trained on The Pile

- GPT-Neo 2.7B
  Codex-85M
- GPT-J 6B
  Codex-300M

TabNine

	PASS@k $k=1$	$k=10$	$k=100$
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%

Evaluation on APPS Dataset

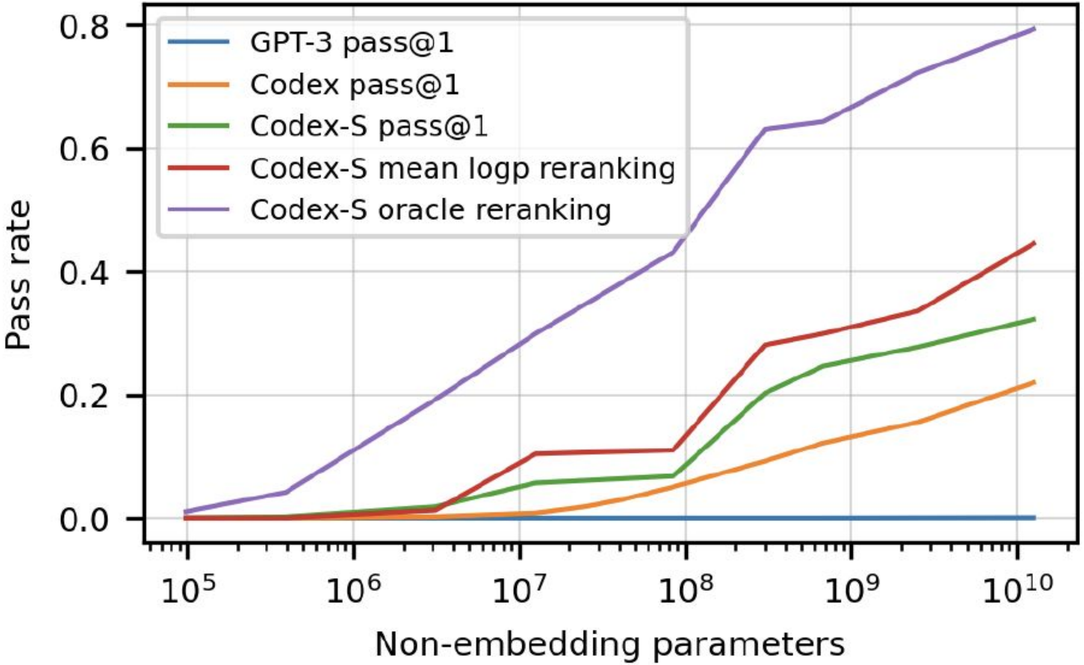
5000 train + 5000 test code challenge with 3 difficulty levels

	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%)

Supervised Fine-Tuning

Codex-**S**

- Competitive Programming
  +10K problems
- Continuous Integration
  +40K tracing tests
- Filtered Out
  Non-deterministic,
  Ambiguous, Difficult



Docstring generation

Codex-**D**

- Describe intent
- Concat signature, solution, docstring

Grade by hand, time-consuming

- 1,640 problems
- 10 samples each problem

Limitations, Impacts, Hazards

Over-reliance

Misalignment

Bias & Representation

Economic & Labor Markets

Security

Environmental

Legal

Risk Mitigation

Sandbox & Playground Demo

The Codex model series is a descendant of our GPT-3 series that's been trained on both natural language and billions of lines of code.
It's most capable in Python and proficient in over a dozen languages including JavaScript, Go, Perl, PHP, Ruby, Swift, TypeScript, SQL, and even Shell.

Sandbox & Playground Demo

You can use Codex for a variety of tasks including:

- Turn comments into code
- Complete your next line or function in context
- Bring knowledge to you,
  Finding a useful library or API call for an application
- Add comments
- Rewrite code for efficiency

Conclusion & Discussion

Experience with code generation?

Ideas of usage and applications?

Thoughts or concerns?