InCoder, SantaCoder, and StarCoder: Findings from Training Code LLMs

Daniel Fried, with many others from Meta AI and the BigCode project
Are bigger models the solution for AI-assisted programming?
On the Naturalness of Software

Abram Hindle, Earl Barr, Mark Gabel, Zhendong Su, Prem Devanbu

[ICSE 2012; Most Influential Paper 2022]

Natural languages like English are rich, complex, and powerful. We begin with the conjecture that most software is also natural, in the sense that it is created by humans at work, with all the attendant constraints and limitations—and thus, like natural language, it is also likely to be repetitive and predictable. We then proceed to ask whether a) code can be usefully modeled by statistical language models and b) such models can be leveraged to support software engineers.
Posing This Question in 2012...

On the Naturalness of Software

Abram Hindle, Earl Barr, Mark Gabel, Zhendong Su, Prem Devanbu

- n-gram models trained on ~25 million lines of code
- Substantial improvements to Eclipse’s auto-complete
- But, 3-4 orders of magnitude less data than modern neural models

[ICSE 2012; Most Influential Paper 2022]
... and now

AI pair programming is here.

75% more fulfilled

55% faster coding

46% code written

Keep flying with your favorite editor

Now available for all businesses
Function pass rate on a Python docstring-to-function task [HumanEval, Chen et al. 2021] by amount of Python data & model scale:

- 26% CodeGen-Mono
- 29% Codex-12B [Chen et al. 2021]
- 17% PaLM-Coder
- 34% StarCoder
- 6% PaLM-Mono
- 12% Codex-Mono
- 15% PaLM-Mono

[Compiled from Chen et al. 2021, Xu et al. 2021, Li et al. 2021, Fried et al. 2022, Nijkamp et al. 2022, Chowdhery et al. 2022, Li et al. 2023]
Are bigger models the solution for AI-assisted programming?

YES
Outline

- InCoder
  - Infilling and natural language data
- The Stack & SantaCoder
  - Data filtering and model improvement experiments
- StarCoder
  - More data: more languages, issues, commits, Jupyter...
  - Scale
Outline

- InCoder
  - Infilling and natural language data
- The Stack & SantaCoder
  - Data filtering and model improvement experiments
- StarCoder
  - More data: more languages, issues, commits, Jupyter...
  - Scale
LLM Training Objectives

```
def minimize_in_graph(build_loss_fn, num_steps=200, optimizer=None):
    """Minimize a loss function using gradient.
    Args:
        build_loss_fn: a function that returns a loss tensor for a mini-batch of examples.
        num_steps: number of gradient descent steps to perform.
        optimizer: an optimizer to use when minimizing the loss function. If None, will use Adam
    """
    optimizer = tf.compat.v1.train.AdamOptimizer(0.1) if optimizer is None else optimizer
    minimize_op = tf.compat.v1.while_loop(
        cond=lambda step: step < num_steps,
        body=train_loop_body,
        loop_vars=[tf.constant(0)], return_same_structure=True)[0]
    return minimize_op
```

"""Causal"" (L-to-R)
[e.g. GPT-*, Codex]

Masked Infilling
[e.g. BERT, CodeBERT]

"""Causal Masking"" / Fill-in-the-Middle (FIM)
[Donahue+ 2020, Aghajanyan+ 2022, ours, Bavarian+ 2022]
Causal Masking / FIM Objective

Training

Original Document

```python
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file."""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

[Donahue et al. 2020, Aghajanyan et al. 2022, Fried et al. 2022, Bavarian et al. 2022]
Evaluation: HumanEval Benchmark

Constructed by authors of Codex paper; programming puzzle/simple contest problems. Evaluated using unit tests.

```python
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    """
    Check if in given list of numbers, are any two numbers closer to each other
    than given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True
    """
    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True
    return False
```
Model Training

Training Data

- 600K permissively-licensed repositories from GitHub & GitLab. ~150GB total
- StackOverflow: questions, answers, comments. ~50GB

Models

- Standard transformer LM
- 1B model: ~1 week on 128 V100s
- 6B model: ~3 weeks on 240 V100s
Zero-Shot Software Tasks via Infilling

Zero-shot Inference
Docstring Generation

```python
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.
    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
    return word_counts
```

Multi-Region Infilling

```python
from collections import Counter

def word_count(file_name):
    """Count the number of occurrences of each word in the file."""
    words = []
    with open(file_name) as file:
        for line in file:
            words.append(line.strip())
    return Counter(words)
```
Zero-shot evaluation on several software development-inspired code infilling tasks (we’ll show two).

Compare the model in three different modes to evaluate benefits of suffix context:

- **Baselines**
  - Left-to-Right Single: Doesn’t use suffix
  - Left-to-Right Rerank: Only uses suffix when reranking

- **Ours**
  - Causal Masking: Uses suffix when generating
Fill in one or more lines of a function; evaluate with unit tests.

```python
from typing import List

def has_close_elements(numbers: List[float], threshold: float) -> bool:
    
    Check if in given list of numbers, are any two numbers closer to each other than given threshold.
    >>> has_close_elements([1.0, 2.0, 3.0], 0.5)
    False
    >>> has_close_elements([1.0, 2.8, 3.0, 4.0, 5.0, 2.0], 0.3)
    True

    for idx, elem in enumerate(numbers):
        for idx2, elem2 in enumerate(numbers):
            if idx != idx2:
                distance = abs(elem - elem2)
                if distance < threshold:
                    return True
    return False

```

<table>
<thead>
<tr>
<th>Method</th>
<th>Pass Rate</th>
<th>Exact Match</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-R single</td>
<td>24.9</td>
<td>15.8</td>
</tr>
<tr>
<td>L-R reranking</td>
<td>28.2</td>
<td>17.6</td>
</tr>
<tr>
<td>CM infilling</td>
<td>38.6</td>
<td>20.6</td>
</tr>
</tbody>
</table>

Constructed from HumanEval [Chen et al. 2021]
**Evaluation: Docstring Generation**

```python
def count_words(filename: str) -> Dict[str, int]:
    """
    Counts the number of occurrences of each word in the given file.
    :param filename: The name of the file to count.
    :return: A dictionary mapping words to the number of occurrences.
    """
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
        return word_counts
```

<table>
<thead>
<tr>
<th>Method</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours: L-R single</td>
<td>16.05</td>
</tr>
<tr>
<td>Ours: L-R reranking</td>
<td>17.14</td>
</tr>
<tr>
<td>Ours: Causal-masked infilling</td>
<td>18.27</td>
</tr>
</tbody>
</table>

[CodeXGlue, Lu et al. 2021]
### Evaluation: Return Type Prediction

#### Type Inference

```python
def count_words(filename: str) -> Dict[str, int]:
    """Count the number of occurrences of each word in the file.""
    with open(filename, 'r') as f:
        word_counts = {}
        for line in f:
            for word in line.split():
                if word in word_counts:
                    word_counts[word] += 1
                else:
                    word_counts[word] = 1
        return word_counts
```

<table>
<thead>
<tr>
<th>Method</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours: Left-to-right single</td>
<td>30.8</td>
</tr>
<tr>
<td>Ours: Left-to-right reranking</td>
<td>33.3</td>
</tr>
<tr>
<td>Ours: Causal-masked infilling</td>
<td><strong>59.2</strong></td>
</tr>
<tr>
<td>TypeWriter (Supervised)</td>
<td>48.3</td>
</tr>
</tbody>
</table>

[TypeWriter OSS, Pradel et al. 2020]
### Ablations

- StackOverflow data improves performance
- Comparable performance from infilling and non-infilling models

<table>
<thead>
<tr>
<th>#</th>
<th>Size (B)</th>
<th>Obj.</th>
<th>Training Data</th>
<th>Data Size</th>
<th>Train Tokens</th>
<th>HumanEval Pass@1</th>
<th>MBPP Pass@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>6.7</td>
<td>CM</td>
<td>multi lang + SO</td>
<td>204 GB</td>
<td>52 B</td>
<td>15</td>
<td>19.4</td>
</tr>
<tr>
<td>2)</td>
<td>1.3</td>
<td>CM</td>
<td>multi lang + SO</td>
<td>204 GB</td>
<td>52 B</td>
<td>8</td>
<td>10.9</td>
</tr>
<tr>
<td>3)</td>
<td>1.3</td>
<td>LM</td>
<td>multi lang + SO</td>
<td>204 GB</td>
<td>52 B</td>
<td>6</td>
<td>8.9</td>
</tr>
<tr>
<td>4)</td>
<td>1.3</td>
<td>LM</td>
<td>Python + SO</td>
<td>104 GB</td>
<td>25 B</td>
<td>9</td>
<td>9.8</td>
</tr>
<tr>
<td>5)</td>
<td>1.3</td>
<td>LM</td>
<td>Python</td>
<td>49 GB</td>
<td>11 B</td>
<td>5</td>
<td>6.1</td>
</tr>
</tbody>
</table>
Demo

Demo: huggingface.co/spaces/facebook/incoder-demo
Outline

- InCoder
  - Infilling and natural language data

- The Stack & SantaCoder
  - Experiments with data filtering and model improvements

- StarCoder
  - More data: more languages, issues, commits, Jupyter...
  - Scale
The Stack: Dataset

GH Archive

query

220 M repos names

near-deduplication

2.9 TB of data

Raw dataset

git clone

137 M repos
52 B files
102 TB of data

selecting file extensions

license filtering

6.4 TB of data

69 TB of data

[Kocetkov et al. 2022]
The Stack: Dataset

Permissive license distribution of licenses used to filter the dataset:

MIT (67.7%) | Apache-2.0 (19.1%) | BSD-3-Clause (3.9%) | Unlicense (2.0%) | CC0-1.0 (1.5%) | BSD-2-Clause (1.2%) | CC-BY-4.0 (1.1%) | CC-BY-3.0 (0.7%) | 0BSD (0.4%) | RSA-MD (0.3%) | WTFPL (0.2%) | MIT-0 (0.2%) | Others (166) (2.2%)

[Kocetkov et al. 2022]
The Stack: Python Models

- Possible to approximate Codex-12B performance with permissively licensed data? Train 350M models on Python

- **Deduplication always improves performance** (https://huggingface.co/blog/dedup)

- License filtering hurts, but there’s enough data available to match Chen et al. 2021

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Filtering</th>
<th>pass@1</th>
<th>pass@10</th>
<th>pass@100</th>
<th>Python Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codex (300M)</td>
<td>Exact-dedup?</td>
<td>13.17</td>
<td>20.17</td>
<td>36.27</td>
<td>180 GB</td>
</tr>
<tr>
<td>CodeGen (350M)</td>
<td>unknown</td>
<td>12.76</td>
<td>23.11</td>
<td>35.19</td>
<td></td>
</tr>
<tr>
<td>Python all-license</td>
<td>None</td>
<td>13.11</td>
<td>21.77</td>
<td>36.67</td>
<td>740 GB</td>
</tr>
<tr>
<td></td>
<td>Near-dedup</td>
<td>17.34</td>
<td>27.64</td>
<td>45.52</td>
<td></td>
</tr>
<tr>
<td>Python permissive-license</td>
<td>None</td>
<td>10.99</td>
<td>15.94</td>
<td>27.21</td>
<td>191 GB</td>
</tr>
<tr>
<td></td>
<td>Near-dedup</td>
<td>12.89</td>
<td>22.26</td>
<td>36.01</td>
<td>80 GB</td>
</tr>
</tbody>
</table>
SantaCoder: Overview

- Preparation for a big run: explorations at 1B scale
- Data: The Stack
- Tokenizer: BPE following GPT-2 recipe; use a digit splitter
- Ablations
  - Multi-query attention and infilling (FIM, Bavarian et al. 2022)
  - Data filtering
Multi-Query Attention

- Designed to reduce memory-bandwidth cost to speed up inference

![Diagram of Multi-Query Attention](image)

- Scaled dot-product attention
- Shared key, value projection parameters across heads

Shazeer, 2019
SantaCoder: Model Ablations

- Infilling (FIM) and MQA “for cheap”

<table>
<thead>
<tr>
<th>Language</th>
<th>Attention</th>
<th>FIM</th>
<th>HumanEval</th>
<th>MBPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>Multi Query Attention</td>
<td>✓</td>
<td>0.35</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>Multi Head Attention</td>
<td>✓</td>
<td>0.36</td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>Multi Query Attention</td>
<td>✕</td>
<td>0.37</td>
<td>0.55</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Multi Query Attention</td>
<td>✓</td>
<td>0.33</td>
<td>0.64</td>
</tr>
<tr>
<td></td>
<td>Multi Head Attention</td>
<td>✓</td>
<td>0.37</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>Multi Query Attention</td>
<td>✕</td>
<td>0.37</td>
<td>0.65</td>
</tr>
<tr>
<td>Python</td>
<td>Multi Query Attention</td>
<td>✓</td>
<td>0.36</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>Multi Head Attention</td>
<td>✓</td>
<td>0.38</td>
<td>0.70</td>
</tr>
<tr>
<td></td>
<td>Multi Query Attention</td>
<td>✕</td>
<td>0.39</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Table 5: Pass@100 results for the architecture ablations on HumanEval and MBPP.
SantaCoder: Data Filtering Ablations

- Remove repos with < 5 stars
  - Hurts substantially!

- Remove files with low (or very high) comment-to-code ratio
  ~ Mixed effects

- More aggressive near-duplicate filtering
  + Very slight improvements

- Remove files with low character-to-token ratios
  + Very slight improvements
Outline

- InCoder
  - Infilling and natural language data
- The Stack & SantaCoder
  - Experiments with data filtering
- StarCoder
  - More data: more languages, issues, commits, Jupyter...
  - Scale
A Large-Scale Multilingual Model

We follow the natural distribution and sample data from 86 languages proportionally to their volume. **800GB total.** Lots of natural language (~20%)!
GitHub Data

Issues (discussion threads)

```xml
<issue_start>title + USERID: comment
<issue_comment>USERID: comment ...
<issue_closed> <eos>
```

Model input:

```xml
<issue_start><issue_comment>username_0: I am trying to train a
text classifier based on BERT using 'transformers' but I
get the error: 'RuntimeError: CUDA Out of memory'.
<issue_comment>
```

Model output:

```xml
username_1: Hi,

This is not an issue with transformers. You should look at your
GPU usage and see if you can free up some space before
training starts (or use smaller batch sizes).
```

Commits and Commit Messages

Model input:

```xml
<commit_before>def fibonacci(n):
<commit_msg>add type hints to
function
<commit_after>def
```

Model output:

```xml
def fibonacci(n: int) -> list[int]:
```
Jupyter Notebooks

Model input:

```python
<jupyter_text>Let’s test our ‘is_prime‘ function:<jupyter_code>
    print(is_prime(3))
print(is_prime(4))
print(is_prime(29))
print(is_prime(33))<jupyter_output>
```

Model output:

```
True
False
True
False
```

Model input:

```python
<jupyter_code>numbers = [1, 9, 8, 3, 27]
print([n*2 for n in numbers])<jupyter_output>
```

Model output:

```
[2, 18, 16, 6, 54]
```
Flash Attention

→ up to 4x speedup over standard attention
→ scale sequence length up to 8192 tokens.

[Dao et al. 2022]
Models

- **StarCoderBase**
  - 15.5B parameters, trained on 1T tokens (~3 epochs)
    - This is much smaller than Chinchilla optimal, but we were aiming for inference efficiency
    - Multiple epochs didn’t seem to hurt
  - ~1 month on 512 80GB A100s
  - Megatron-LM with BF16 and FlashAttention

- **StarCoder**
  - Continued training on 35B tokens of Python (two epochs)
Translations of the HumanEval benchmark into other programming languages.

Together, StarCoderBase and StarCoder outperform OpenAI’s code-cushman-001 on HumanEval in 12 languages.

Surprisingly, StarCoder outperforms StarCoderBase on 9 languages in addition to Python.

<table>
<thead>
<tr>
<th>Language</th>
<th>code-cushman-001</th>
<th>StarCoder</th>
<th>StarCoderBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpp</td>
<td>30.59</td>
<td><strong>31.55</strong></td>
<td>30.56</td>
</tr>
<tr>
<td>c-sharp</td>
<td><strong>22.06</strong></td>
<td>21.01</td>
<td>20.56</td>
</tr>
<tr>
<td>d</td>
<td>6.73</td>
<td><strong>13.57</strong></td>
<td>10.01</td>
</tr>
<tr>
<td>go</td>
<td>19.68</td>
<td>17.61</td>
<td><strong>21.47</strong></td>
</tr>
<tr>
<td>java</td>
<td><strong>31.90</strong></td>
<td>30.22</td>
<td>28.53</td>
</tr>
<tr>
<td>julia</td>
<td>1.54</td>
<td><strong>23.02</strong></td>
<td>21.09</td>
</tr>
<tr>
<td>javascript</td>
<td>31.27</td>
<td>30.79</td>
<td><strong>31.70</strong></td>
</tr>
<tr>
<td>lua</td>
<td>26.24</td>
<td>23.89</td>
<td><strong>26.61</strong></td>
</tr>
<tr>
<td>php</td>
<td><strong>28.94</strong></td>
<td>26.08</td>
<td>26.75</td>
</tr>
<tr>
<td>perl</td>
<td><strong>19.29</strong></td>
<td>17.34</td>
<td>16.32</td>
</tr>
<tr>
<td>python</td>
<td>30.71</td>
<td><strong>33.57</strong></td>
<td>30.35</td>
</tr>
<tr>
<td>r</td>
<td>10.99</td>
<td><strong>15.50</strong></td>
<td>10.18</td>
</tr>
<tr>
<td>ruby</td>
<td><strong>28.63</strong></td>
<td>1.24</td>
<td>17.25</td>
</tr>
<tr>
<td>racket</td>
<td>7.05</td>
<td>0.07</td>
<td><strong>11.77</strong></td>
</tr>
<tr>
<td>rust</td>
<td><strong>25.22</strong></td>
<td>21.84</td>
<td>24.46</td>
</tr>
<tr>
<td>scala</td>
<td>27.62</td>
<td>27.61</td>
<td><strong>28.79</strong></td>
</tr>
<tr>
<td>bash</td>
<td><strong>11.74</strong></td>
<td>10.46</td>
<td>11.02</td>
</tr>
<tr>
<td>swift</td>
<td>22.12</td>
<td><strong>22.74</strong></td>
<td>16.74</td>
</tr>
<tr>
<td>typescript</td>
<td>31.26</td>
<td><strong>32.29</strong></td>
<td>32.15</td>
</tr>
</tbody>
</table>
StarCoderBase: Performance Over Training
StarCoderBase: Performance By Data

- How correlated is code completion performance for a language with the amount of data available for a language?
- Train model for 200B tokens (on all languages). Evaluate on all languages, getting a dot for each language. Observe a strong correlation.
- Continue training, evaluate again at 400B tokens. The correlation remains strong, and line shifts upward.
Here is a sample dataframe:

df = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})

I'd like to add inverses of each existing column to the dataframe and name them based on existing column names with a prefix, e.g. inv_A is an inverse of column A and so on.

The resulting dataframe should look like so:

result = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6], "inv_A": [1/1, 1/2, 1/3], "inv_B": [1/4, 1/5, 1/6]})

Obviously there are redundant methods like doing this in a loop, but there should exist much more pythonic ways of doing it ... [omitted for brevity]

```
result = df.join(df.apply(lambda x: 1/x).add_prefix("inv_"))
```

<table>
<thead>
<tr>
<th>Format</th>
<th>Model</th>
<th>Matplotlib</th>
<th>NumPy</th>
<th>Pandas</th>
<th>PyTorch</th>
<th>SciPy</th>
<th>Scikit-Learn</th>
<th>TensorFlow</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Completion</td>
<td>InCoder-6B</td>
<td>28.3</td>
<td>4.4</td>
<td>3.1</td>
<td>4.4</td>
<td>2.8</td>
<td>2.8</td>
<td>3.8</td>
<td>7.4</td>
</tr>
<tr>
<td>Completion</td>
<td>CodeGen-16B-Mono</td>
<td>31.7</td>
<td>10.9</td>
<td>3.4</td>
<td>7.0</td>
<td>9.0</td>
<td>10.8</td>
<td>15.2</td>
<td>11.7</td>
</tr>
<tr>
<td>Completion</td>
<td>code-cushman-001</td>
<td>40.7</td>
<td>21.8</td>
<td>7.9</td>
<td>12.4</td>
<td>11.3</td>
<td>18.0</td>
<td>12.2</td>
<td>18.1</td>
</tr>
<tr>
<td>Completion</td>
<td>StarCoderBase</td>
<td>47.0</td>
<td>27.1</td>
<td>10.1</td>
<td>19.5</td>
<td>21.7</td>
<td>27.0</td>
<td>20.5</td>
<td>23.8</td>
</tr>
<tr>
<td>Completion</td>
<td>StarCoder</td>
<td><strong>51.7</strong></td>
<td><strong>29.7</strong></td>
<td><strong>11.4</strong></td>
<td><strong>21.4</strong></td>
<td><strong>20.2</strong></td>
<td><strong>29.5</strong></td>
<td><strong>24.5</strong></td>
<td><strong>26.0</strong></td>
</tr>
</tbody>
</table>
Evaluating Infilling

<table>
<thead>
<tr>
<th>Model</th>
<th>Java</th>
<th>JavaScript</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>InCoder-6B</td>
<td>0.49</td>
<td>0.51</td>
<td>0.31</td>
</tr>
<tr>
<td>SantaCoder</td>
<td>0.62</td>
<td>0.60</td>
<td>0.44</td>
</tr>
<tr>
<td>StarCoder</td>
<td>0.73</td>
<td>0.74</td>
<td>0.62</td>
</tr>
</tbody>
</table>

Single-line code completion for three languages (SantaCoder/InCoder benchmarks)

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>InCoder-6B</td>
<td>18.27</td>
</tr>
<tr>
<td>SantaCoder</td>
<td>19.74</td>
</tr>
<tr>
<td>StarCoderBase</td>
<td>21.38</td>
</tr>
<tr>
<td>StarCoder</td>
<td>21.99</td>
</tr>
</tbody>
</table>

Python return-type prediction (InCoder/TypeWriter benchmarks)

<table>
<thead>
<tr>
<th>Model</th>
<th>Non-None F1</th>
<th>All F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>InCoder-6B</td>
<td>59.1</td>
<td>46.8</td>
</tr>
<tr>
<td>SantaCoder</td>
<td>66.9</td>
<td>78.5</td>
</tr>
<tr>
<td>StarCoderBase</td>
<td>77.4</td>
<td>86.6</td>
</tr>
<tr>
<td>StarCoder</td>
<td>77.1</td>
<td>86.4</td>
</tr>
</tbody>
</table>

TypeScript type inference (TypeWeaver benchmarks)

<table>
<thead>
<tr>
<th>Packages type check</th>
<th>✓</th>
<th>Total</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>InCoder</td>
<td>30</td>
<td>128</td>
<td>23.4</td>
</tr>
<tr>
<td>StarCoderBase</td>
<td>49</td>
<td>128</td>
<td>38.3</td>
</tr>
</tbody>
</table>

Python docstring generation (CodeXGLUE / InCoder benchmark)
Derived test data from GPL repositories on GitHub. GPL was excluded from training data.

Demonstrates StarCoder can benefit from information within long files or repositories.

Longer contexts provides noticeable decreases in perplexity.
Non-Trivial Natural Language Abilities

- Surprisingly reasonable performance on some natural language reasoning tasks
- CodeGen < StarCoderBase < LLaMA

**Problem:** Beth bakes 4, 2 dozen batches of cookies in a week. If these cookies are shared amongst 16 people equally, how many cookies does each person consume?

**Solution:** Beth bakes 4 2 dozen batches of cookies for a total of $4 \times 2 = 8$ dozen cookies. There are 12 cookies in a dozen and she makes 8 dozen cookies for a total of $12 \times 8 = 96$ cookies. She splits the 96 cookies equally amongst 16 people so they each eat $96 / 16 = 6$ cookies.

**Final Answer:** 6

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>GSM8K CoT</th>
<th>+maj1@100</th>
<th>GSM8K PAL</th>
<th>+maj1@40</th>
</tr>
</thead>
<tbody>
<tr>
<td>StarCoderBase</td>
<td>15.5B</td>
<td>8.4</td>
<td>—</td>
<td>21.5</td>
<td>31.2</td>
</tr>
<tr>
<td>CodeGen-Multi</td>
<td>16B</td>
<td>3.18</td>
<td>—</td>
<td>8.6</td>
<td>15.2</td>
</tr>
<tr>
<td>CodeGen-Mono</td>
<td>16B</td>
<td>2.6</td>
<td>—</td>
<td>13.1</td>
<td>22.4</td>
</tr>
<tr>
<td>7B</td>
<td>11.0</td>
<td>18.1</td>
<td>10.5</td>
<td>16.8</td>
<td></td>
</tr>
<tr>
<td>13B</td>
<td>17.8</td>
<td>29.3</td>
<td>16.9</td>
<td>28.5</td>
<td></td>
</tr>
<tr>
<td>LLaMA</td>
<td>33B</td>
<td>35.6</td>
<td>38.7</td>
<td>50.3</td>
<td></td>
</tr>
<tr>
<td>65B</td>
<td>50.9</td>
<td>69.7</td>
<td>—</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>
## Reasoning Tasks in HELM

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>Open Access</th>
<th>Synth. Reason. (AS)</th>
<th>Synth. Reason. (NL)</th>
<th>bAbI</th>
<th>Dyck</th>
<th>GSM8K</th>
<th>MATH</th>
<th>MATH (CoT)</th>
<th>LSAT</th>
<th>Legal Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>code-davinci-002</td>
<td>175B</td>
<td></td>
<td><strong>54.0</strong></td>
<td>68.4</td>
<td><strong>68.6</strong></td>
<td>80.5</td>
<td><strong>56.8</strong></td>
<td><strong>41.0</strong></td>
<td>43.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>text-davinci-003</td>
<td>175B</td>
<td></td>
<td>50.2</td>
<td><strong>73.4</strong></td>
<td>65.3</td>
<td>75.1</td>
<td>50.6</td>
<td>39.0</td>
<td><strong>44.9</strong></td>
<td><strong>23.3</strong></td>
<td>62.2</td>
</tr>
<tr>
<td>Luminous Supreme</td>
<td>70B</td>
<td></td>
<td>31.2</td>
<td>—</td>
<td>50.4</td>
<td>72.9</td>
<td>11.2</td>
<td>14.9</td>
<td>5.7</td>
<td>21.2</td>
<td>53.0</td>
</tr>
<tr>
<td>StarCoderBase</td>
<td>15.5B</td>
<td>✓</td>
<td>44.0</td>
<td>21.0</td>
<td>50.4</td>
<td><strong>85.4</strong></td>
<td>8.4</td>
<td>15.1</td>
<td>7.0</td>
<td>19.0</td>
<td>53.2</td>
</tr>
<tr>
<td>Cohere Command Beta</td>
<td>52.4B</td>
<td>✓</td>
<td>24.3</td>
<td>24.5</td>
<td>47.3</td>
<td>42.1</td>
<td>13.8</td>
<td>13.3</td>
<td>7.5</td>
<td>22.9</td>
<td>60.6</td>
</tr>
<tr>
<td>J1-Jumbo v1</td>
<td>178B</td>
<td></td>
<td>26.3</td>
<td>17.4</td>
<td>54.3</td>
<td>44.5</td>
<td>5.4</td>
<td>8.9</td>
<td>3.3</td>
<td>23.2</td>
<td>48.4</td>
</tr>
<tr>
<td>J1-Grande v2 beta</td>
<td>17B</td>
<td></td>
<td>28.6</td>
<td>13.9</td>
<td>47.0</td>
<td>61.7</td>
<td>9.6</td>
<td>12.7</td>
<td>6.8</td>
<td>19.1</td>
<td>56.2</td>
</tr>
<tr>
<td>code-cushman-001</td>
<td>12B</td>
<td></td>
<td>34.1</td>
<td>16.4</td>
<td>48.1</td>
<td>45.1</td>
<td>4.9</td>
<td>9.9</td>
<td>7.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>175B</td>
<td>✓</td>
<td>22.5</td>
<td>24.8</td>
<td>50.7</td>
<td>49.4</td>
<td>4.0</td>
<td>6.5</td>
<td>2.6</td>
<td>22.0</td>
<td>53.2</td>
</tr>
<tr>
<td>GPT-NeoX</td>
<td>20B</td>
<td>✓</td>
<td>20.4</td>
<td>16.7</td>
<td>46.8</td>
<td>74.7</td>
<td>5.3</td>
<td>14.1</td>
<td>7.1</td>
<td>19.1</td>
<td>51.5</td>
</tr>
<tr>
<td>BLOOM</td>
<td>176B</td>
<td>✓</td>
<td>30.4</td>
<td>19.7</td>
<td>44.7</td>
<td>54.5</td>
<td>9.5</td>
<td>4.3</td>
<td>5.5</td>
<td>20.9</td>
<td>54.3</td>
</tr>
<tr>
<td>GLM</td>
<td>130B</td>
<td>✓</td>
<td>25.2</td>
<td>25.4</td>
<td>44.3</td>
<td>54.9</td>
<td>6.1</td>
<td>0</td>
<td>5.9</td>
<td>19.3</td>
<td>45.1</td>
</tr>
<tr>
<td>UL2</td>
<td>20B</td>
<td>✓</td>
<td>20.5</td>
<td>21.7</td>
<td>50.1</td>
<td>14.0</td>
<td>2.4</td>
<td>0</td>
<td>0</td>
<td>20.7</td>
<td>50.6</td>
</tr>
<tr>
<td>OPT</td>
<td>66B</td>
<td>✓</td>
<td>19.3</td>
<td>21.3</td>
<td>40.8</td>
<td>47.1</td>
<td>1.8</td>
<td>4.8</td>
<td>2.9</td>
<td>17.5</td>
<td>52.7</td>
</tr>
<tr>
<td>YaLM</td>
<td>100B</td>
<td>✓</td>
<td>5.6</td>
<td>6.1</td>
<td>34.6</td>
<td>63.3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.3</td>
<td>48.4</td>
</tr>
<tr>
<td>T5</td>
<td>11B</td>
<td>✓</td>
<td>19.6</td>
<td>10.1</td>
<td>41.2</td>
<td>34.7</td>
<td>2.3</td>
<td>0</td>
<td>0</td>
<td>15.9</td>
<td>55.8</td>
</tr>
</tbody>
</table>
Chat Models via Continued Training

- StarChat-Beta: Continued training on Falcon English Web, the Pile, and OpenAssistant Dataset

Create a simple HTML website for a fictional company called "TechCoAI". Please make an HTML file that adheres to the following specifications:

- The webpage should have a title that says "TechCoAI - Welcome".
- The background color of the webpage should be light blue (#E6F1F6).
- The website should have a navigation menu with three links: "Home", "About", and "Contact", these links should be displayed in dark blue.
- Include a paragraph of text that describes TechCoAI's mission and vision: "The mission of TechCoAI is to build the next generation of AI models with responsible practices."
- Finally, create a footer with the copyright information.
- The footer should display the copyright symbol (©), the current year 2023, and "TechCoAI. All rights reserved."

Remember to follow these instructions precisely and ensure that the HTML code is properly structured and formatted.

Welcome To Tech Co Ai

We are building the Next Generation Of AI Models With Responsible Practices.

© 2023 TechCoAI | All Rights Reserved

https://huggingface.co/spaces/HuggingFaceH4/starchat-playground
Programming as Communication

We begin with the conjecture that most software... is created by humans at work, with all the attendant constraints and limitations communicating with the compiler, other developers, and themselves, and thus, like natural language, it is also likely to be repetitive and predictable. Writing software is a form of contextual and interactive communication. We then proceed to ask whether a) code can be usefully modeled by statistical language models and b) such models can be leveraged to support software engineers.
Communicating with Multiple Modalities

- **As Inputs**
  - Natural Language: InCoder, SantaCoder, Starcoder
  - Partial Code: InCoder, SantaCoder, Starcoder
  - Tests & Execution: MBR-Exec [Shi et al. 2022]

- **As Outputs**
  - InCoder, SantaCoder, Starcoder

- **Deictic (Pointing / Highlighting)**: [Wallace et al., in progress]

Modality Choice
[Lin et al., 2022]
Communicating with Multiple Modalities

As Inputs
- Natural Language: InCoder, SantaCoder, Starcoder
- Partial Code: InCoder, SantaCoder, Starcoder
- Tests & Execution: MBR-Exec [Shi et al. 2022]
- Edits to Code: [Wallace et al., in progress]
- Deictic (Pointing / Highlighting):

As Outputs
- Natural Language: InCoder, SantaCoder, Starcoder
- Partial Code: InCoder, SantaCoder, Starcoder
- Tests & Execution: MBR-Exec [Shi et al. 2022]
- Edits to Code: [Wallace et al., in progress]
- Deictic (Pointing / Highlighting):

Modality Choice [Lin et al., 2022]
Using Test Inputs

**Description:**

```python
def longest(strings: List[str]) -> Optional[str]:
    """ Out of list of strings, return the longest one. 
    Return the first one in case of multiple strings of 
    the same length. Return None if the list is empty."
```

**Test Inputs:**

- `longest([]) == ___`
- `longest(['x', 'y', 'z']) == ___`
- `longest(['x', 'yyy', 'zzzz', 'www', 'kkkk', 'abc']) == ___`

**Minimum Bayes Risk with Execution:**

[Shi et al. 2022. See also AlphaCode, Li et al. 2022]
Other Features of Communication

- **Communicative cost**
  - Copilot outputs can be hard to understand [Vaithilingam et al. 2022]
  - Would a user rather type a comment or edit code?

- **Resolving uncertainty**
  - Disambiguate by prompting with test inputs [Zhong et al. 2022]
  - How to convey uncertainty to the user & build trust?

- **Adaptation**
  - Acceleration vs exploration modes for using Copilot [Barke et al. 2022]
  - API preferences, functional vs imperative, design patterns, documentation style ...
Collaborators

Armen Aghajanyan
Jessy Lin
Sida Wang
Eric Wallace
Freda Shi
Ruiqi Zhong
Scott Yih
Luke Zettlemoyer
Mike Lewis
Raymond Li
Louba Ben Allal
Denis Kocetkov
Arjun Guha
Leandro von Werra
Harm de Vries

and ~60 others from the BigCode project!