Language Models of Code

Arseniy Andreyev, Jiatong Yu
Codex

Introduction
Evaluation
Methodology
Experiments
Discussions

InCoder

CodeGen

Codex for NLP
Outline

Codex

**Introduction**
- Evaluation
- Methodology
- Experiments
- Discussions

**InCoder**

**CodeGen**

**Codex for NLP**

2022.11
"Hey, GitHub!"

2022.4
InCoder: Code Infilling and Synthesis

2022.3
CodeGen: Multi-turn Program Synthesis

2021.7
Codex: Evaluating Large Language Models Trained on Code

2020
CodeBERT, PyMT5, TransCoder, ContraCode...
Brief overview
Brief overview

**Task:** generate code using LLMs
Given natural-language prompt (docstring), output the code that implements it
Task: generate code using LLMs
Given natural-language prompt (docstring), output the code that implements it

Model: Codex - a GPT-3, fine-tuned on code
up to 12B params
Brief overview

**Task:** generate code using LLMs
Given natural-language prompt (docstring), output the code that implements it

**Model:** Codex - a GPT-3, fine-tuned on code
up to 12B params

**Training data:** 160GB of Python code
Brief overview

**Task:** generate code using LLMs
Given natural-language prompt (docstring), output the code that implements it

**Model:** Codex - a GPT-3, fine-tuned on code
up to 12B params

**Training data:** 160GB of Python code

**Evaluation:** HumanEval
a novel dataset with 164 programming problems created by the authors
generate k samples from the model to see if at least one sample passes all the unit tests
Brief overview

**Task:** generate code using LLMs
Given natural-language prompt (docstring), output the code that implements it

**Model:** Codex - a GPT-3, fine-tuned on code
up to 12B params

**Training data:** 160GB of Python code

**Evaluation:** HumanEval
a novel dataset with 164 programming problems created by the authors
generate k samples from the model to see if at least one sample passes all the unit tests

**Result:** Codex-12B “solves” 72.3% of the problems (given 100 samples)
GPT-3 solves 0%, GPT-J solves 27.7%
if using only one sample (with lowest perplexity) we get 28.8% for Codex, 11.6% for GPT-J
Some examples
Some examples

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]) => 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)
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, 32]) => 0
    ***
    return sum(lst[i] for i in range(0, len(lst)) if i % 2 == 0 and lst[i] % 2 == 1)

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[i:] + group[:i]) 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)
Some examples

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]) => 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)

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)
Real-life examples

```python
import time
s_t = time.time()
if self.use_amp:
    self.scaler.scale(loss).backward()
elif self.use_apex:
    with amp.scale_loss(loss, self.optimizer) as scaled_loss:
        scaled_loss.backward()
elif self.deepspeed:
    # loss gets scaled under gradient_accumulation_steps in deepspeed
    loss = self.deepspeed.backward(loss)
else:
    loss.backward()
e_t = time.time()
print(f"Backward time: ", e_t - s_t)
```

```python
from collections import defaultdict
asins = set()
cnt_cats = defaultdict(int)
cnt_atts = defaultdict(int)
cnt_catatts = defaultdict(lambda: defaultdict(int))
for goal in env.server.goals:
    if goal['asin'] not in asins:
        asins.add(goal['asin'])
        cnt_cats[goal['category']][goal['attributes']] += 1
    for att in goal['attributes']:
        cnt_atts[att] += 1
        cnt_catatts[goal['category']][att] += 1
```
Codex

Introduction

Evaluation

Methodology

Experiments

Discussions

InCoder

CodeGen

Codex for NLP
HumanEval

164 hand-written problems

hand-writing to avoid repeating the problems in the training data (“training data leakage”)

Evaluates language comprehension, reasoning, algorithms and simple math

HE was used in later papers (CodeGen, InCoder)
HumanEval

164 hand-written problems

hand-writing to avoid repeating the problems in the training data ("training data leakage")

Evaluates language comprehension, reasoning, algorithms and simple math

HE was used in later papers (CodeGen, InCoder)

"Check if two words have the same characters."

"Return median of elements in the list l."

"sum_to_n is a function that sums numbers from 1 to n."

"Given a non-empty list of integers lst, add the even elements that are at odd indices."

"Return true if a given number is prime, and false otherwise."

"Return n-th Fibonacci number."
HumanEval: Format

Format:
• function signature
• docstring with examples
• unit-tests

```python
def vowels_count(s):
    """Write a function vowels_count which takes a string as input and returns
    the number of vowels in the string.
    Vowels in this case are 'a', 'e', 'i', 'o', 'u'.
    Here, 'y' is also a vowel, but only when it is at the end of the given word.
    """
    Example:
    >>> vowels_count("abcde")
    2
    >>> vowels_count("ACEDY")
    3
    """
```

```python
def check(candidate):
    # Check some simple cases
    assert candidate("abcde") == 2, "Test 1" assert candidate("Alone") == 3, "Test 2"
    assert candidate("key") == 2, "Test 3" assert candidate("bye") == 1, "Test 4" assert
    candidate("keY") == 2, "Test 5" assert
    candidate("bYe") == 1, "Test 6" assert
    candidate("ACEDY") == 3, "Test 7"
    # Check some edge cases that are easy to work out by hand.
    assert True, "This prints if this assert fails 2 (also good for debugging!)"
```
### HumanEval: Metric

**Functional correctness:**
- Whether the generated code implements the correct function
- I.e. passes all unit tests
- This is the way humans evaluate correctness of the code

BLEU score doesn’t work
- optimized for the semantics of text
HumanEval: Metric

*Functional correctness*: Whether the generated code implements the correct function
I.e. passes all unit tests
This is the way humans evaluate correctness of the code

BLEU score doesn’t work
optimized for the semantics of text

Reference code
```
def f(a, b):
    c = a - b
    return c
```

Equivalent code
```
def f(a, b):
    summ = 0
    summ += a
    summ -= b
    return summ
```
BLEU = 66

Non-equivalent code
```
def f(a, b):
    c = a + b
    return c
```
BLEU = 81
BLEU Score

BLEU score doesn’t work:
• Algorithmic difference
• Variable name
• Operation orders

Optimizing for BLEU score is *not* equivalent to optimizing functional correctness
Q1: For evaluating code generation, why is functional correctness better than match-based metrics (e.g., BLEU)?

Existing match-based metrics are designed for comparing natural languages, which is not inherently applicable to code. In particular, when evaluating code, the aspect that matters is its correct behavior. One can use unit tests to check this correctness (with large likelihood).

“Perhaps the most convincing reason to evaluate functional correctness is that it is used by human developers to judge code.”

In a sense, this makes evaluation of code generation more “precise” than evaluation of text generation.

Recent research (Ren et al.) showed that BLEU score doesn’t capture the semantic features specific to code. Aforementioned experiment result corroborate that BLEU score and the correctness of generated code are not equivalent.
HumanEval: Sampling & pass@k

Given a prompt, generate $k$ samples
a sample is generated until a stop sequence is encountered

pass@k:
having $k$ generations per problem, a problem is “solved” if at least one generation passes all unit tests
total fraction of problems “solved” is reported
HumanEval: pass@k

Estimating pass@k:

- Naively: high variance for small k

Instead:

- Generate $n$ samples ($n \geq k$)
- Use the following unbiased estimator

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

where $c$ is the #(correct samples)
HumanEval: Potential Shortcomings

Small dataset
   large variance when comparing different models

Most of the tasks are “short” - could be solved in less than 10 lines of code

Data leakage
   the solutions to the problems might already be present in the training data
   e.g.: primality

```python
### COMPLETION 4 (CORRECT): ###
if n < 2: return False
if n == 2: return True
if n%2 == 0: return False
return not any(n%k == 0 for k in range(3,int(n **0.5)+1,2))
```
Model + training data

Model: GPT-3
sizes: 12M - 12B

Data: Python code from Github - 159GB

Method: fine-tune on code
use next token prediction

no difference between fine-tuning GPT-3 and training from scratch on code
yet, faster convergence when fine-tuning
Codex-S

Motivation: potential distribution mismatch between GitHub files and HumanEval / APPS problems

Solution: fine-tune Codex on correct standalone functions
Codex-S: Training Data

**Competitive Programming** (10,000 problems)
- Self-contained
- Unit test coverage
- Problem descriptions as docstrings

**Continuous Integration** (40,000 functions)

“Developers regularly merge code changes into a central repository, after which automated builds and tests are run.”
- Open source
- Tracing test functions
Codex-S: Filtering

Concern:
- Low-quality docstring
- Stateful functions

Solution:
Use Codex-12B to generate 100 samples per problem, discard the problem if no generation passes
Codex-S: Tuning

Training examples assembled into the same format as in \textit{pass@k} evaluation

```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]) => 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)
```

Objective: minimize negative log-likelihood of reference solution, mask out loss for prompt
Motivation: useful for safety concerns

Training Dataset: same as Codex-S dataset

Training Objective: Minimize negative log-likelihood of docstring

Evaluation: manually evaluate 10 samples per problem, creating pass@1 and pass@10

<table>
<thead>
<tr>
<th>Model</th>
<th>Pass@1</th>
<th>Pass@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codex-S-12B</td>
<td>32.2%</td>
<td>59.5%</td>
</tr>
<tr>
<td>Codex-D-12B</td>
<td>20.3%</td>
<td>46.5%</td>
</tr>
</tbody>
</table>
Docstring Generation

Examples of generated docstrings:

- "I just found this function online"
- "This test is not correctly written and it's not my solution."
Codex

Introduction
Evaluation
Methodology
Experiments
Discussions

InCoder

CodeGen

Codex for NLP
**Generation Example**

**Codex-12B**

**Prompt**

```python
def is_prime(n):
    """Return true if a given number is prime, and false otherwise."
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

```python
>>> is_prime(6)
False
>>> is_prime(101)
True
>>> is_prime(11)
True
>>> is_prime(13441)
True
>>> is_prime(61)
True
>>> is_prime(4)
False
>>> is_prime(1)
False
"""
```

**Generated samples**

### COMPLETION 1 (WRONG)

```python
for i in range(2, n):
    if n % i == 0:
        check if n is divisible by 1
        Fails for n = 1 (returns True)
return True
```

### COMPLETION 2 (WRONG)

```python
for i in range(2, n):
    if n % i == 0:
        return False
return True
```

### COMPLETION 3 (WRONG)

```python
# TODO: implement this function
pass
```

Fails for all unit test cases

### COMPLETION 4 (CORRECT)

```python
if n < 2: return False
if n == 2: return True
if n%2 == 0: return False
even numbers not prime
return not any(n%k == 0 for k in range(3,int(n**0.5)+1,2))
check if divisible by odd number up to $\sqrt{n}$
```

### COMPLETION 5 (WRONG)

```python
def helper(n):
    # check if an odd number is multiple of any integer up to the square root of the number
    if this is true, n is not prime
    for i in range(2, int(n**0.5 + 1)):
        if n % i == 0:
            return False
    return True

# if helper returns true, then n is prime, else n
# is not prime
if helper(n):
    return True
else:
    return False
```

Fails for n = 1 (returns True)
Sampling Temperature

Temperature experiment on Codex-679M

larger $k$ benefits from higher temperature (i.e. higher diversity)
Sampling Temperature

Pass Rate vs Model Size

- pass@1 ($T^* = 0.2$)
- pass@100 ($T^* = 0.8$)
Sampling Temperature

Codex-S prefers *higher temperature* compared to Codex
Q2: Why does temperature term matter for code generation and what is the relationship between temperature and $k$? What ranking heuristic works best?

**Temperature**

Temperature controls the variance of generated samples, and higher $k$ prefers higher temperature. Since pass@$k$ rewards only whether the model generated any correct result given $k$ samples, higher temperature generates more diverse solutions and more likely one of them would pass.

Codex-S prefers higher temperature than Codex. This could be explained as Codex-S is fine-tuned on data with matching distribution, so it's more "concentrated" than Codex.

**Ranking Heuristic**

Mean log probability works best.
**Sampling Heuristics**

**Oracle**: best sample chosen as one that passes the unit tests

**Back Translation**: best sample maximizes $P(\text{ground truth docstring} \mid \text{generated sample})$

**Mean log probability**

**Sum log probability**

**Random**

*Codex-12B, temp = 0.8*
Sampling Heuristics

**Oracle**: best sample chosen as one that passes the unit tests

**Back Translation**: best sample maximizes $P(\text{ground truth docstring} \mid \text{generated sample})$

Mean log probability
Sum log probability
Random

*Codex-12B, temp = 0.8*
Sampling Heuristics

**Oracle**: best sample chosen as one that passes the unit tests

**Back Translation**: best sample maximizes $P(\text{ground truth docstring} \mid \text{generated sample})$

**Mean log probability**

**Sum log probability**

**Random**

*Codex-12B, $temp = 0.8$*
Sampling Heuristics

**Oracle**: best sample chosen as one that passes the unit tests

**Back Translation**: best sample maximizes $P(\text{ground truth docstring} \mid \text{generated sample})$

**Mean log probability**

**Sum log probability**

**Random**

*Codex-12B, temp = 0.8*
Sampling Heuristics

**Oracle**: best sample chosen as one that passes the unit tests

**Back Translation**: best sample maximizes $P(\text{ground truth docstring | generated sample})$

Mean log probability

Sum log probability

**Random**

*Codex-12B, temp = 0.8*
Q2: Why does temperature term matter for code generation and what is the relationship between temperature and $k$? What ranking heuristic works best?

**Temperature**

Temperature controls the variance of generated samples, and higher $k$ prefers higher temperature. Since pass@$k$ rewards only whether the model generated any correct result given $k$ samples, higher temperature generates more diverse solutions and more likely one of them would pass.

Codex-S prefers higher temperature than Codex. This could be explained as Codex-S is fine-tuned on data with matching distribution, so it's more "concentrated" than Codex.

**Ranking Heuristic**

Mean log probability works best.

**Better decoding heuristics?**
Results

Codex-S outperforms Codex on HumanEval
Results

Codex-S outperforms Codex on HumanEval
Code Model Comparison

**The Pile**: 8% GitHub code, along with natural language data

GPT-J and GPT-Neo: similar architecture

TabNine: Code Autocomplete as a service

---

**Two models in the same vein as Codex:**

<table>
<thead>
<tr>
<th>Model</th>
<th>GPT-Neo (Black et al., 2021)</th>
<th>GPT-J-6B (Wang et al., 2021)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GPT-Neo 125M</td>
<td>0.75%</td>
</tr>
<tr>
<td></td>
<td>GPT-Neo 1.3B</td>
<td>4.79%</td>
</tr>
<tr>
<td></td>
<td>GPT-Neo 2.7B</td>
<td>6.41%</td>
</tr>
<tr>
<td></td>
<td>GPT-J 6B</td>
<td>11.62%</td>
</tr>
<tr>
<td>TabNine</td>
<td>2.58%</td>
<td>4.35%</td>
</tr>
</tbody>
</table>

Both are trained on **The Pile** (8% of which is sourced from GitHub)

GPT-J-6B appears to produce **qualitatively reasonable code** (Woolf, 2021)

### Temperatures

- **GPT-Neo**: 0.2, 0.4, 0.8
- **GPT-J-6B**: 0.2, 0.8
- **Tabnine**: 0.4, 0.8

---

**Codex-12B** goes considerably beyond the performance of prior models

---

x20 fewer parameters than GPT-J-6B
**APPs**

**Sources:** coding websites such as Codeforces, Kattis, etc.

**Difficulty Level:**
1. Introductory
2. Interview
3. Competition

**Distribution:**
5000 training set and 5000 test set

Train set:
52% Introductory, 40% Interview, 8% Competition

Test set:
20% Introductory, 60% Interview, 20% Competition

---

**Example:**

```
Input: [3,0,6,1,4]  
Output: 3
```

---

**Figure 1:** An example “interview”-level problem from APPS (left) along with possible generated code (middle) and two example test cases we use to evaluate the generated code (right). Our evaluation framework has test cases and 10,000 code generation problems of varying difficulty levels.
APPS

*Raw Pass@k*: calculated as before

*Filtered Pass@k*: filter out cases that doesn’t pass the 3 samples in problem description
Problem
Given is a directed graph $G$ with $N$ vertices and $M$ edges. The vertices are numbered 1 to $N$, and the $i$-th edge is directed from Vertex $A_i$ to Vertex $B_i$. It is guaranteed that the graph contains no self-loops or multiple edges. Determine whether there exists an induced subgraph (see Notes) of $G$ such that the in-degree and out-degree of every vertex are both 1. If the answer is yes, show one such subgraph. Here the null graph is not considered as a subgraph.
## APPS

<table>
<thead>
<tr>
<th>APPS dataset</th>
<th>INTRODUCTORY</th>
<th>INTERVIEW</th>
<th>COMPETITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-Neo 2.7B RAW PASS@1</td>
<td>3.90%</td>
<td>0.57%</td>
<td>0.00%</td>
</tr>
<tr>
<td>GPT-Neo 2.7B RAW PASS@5</td>
<td>5.50%</td>
<td>0.80%</td>
<td>0.00%</td>
</tr>
<tr>
<td>1-shot Codex RAW PASS@1</td>
<td>4.14% (4.33%)</td>
<td>0.14% (0.30%)</td>
<td>0.02% (0.03%)</td>
</tr>
<tr>
<td>1-shot Codex RAW PASS@5</td>
<td>9.65% (10.05%)</td>
<td>0.51% (1.02%)</td>
<td>0.09% (0.16%)</td>
</tr>
<tr>
<td>1-shot Codex RAW PASS@100</td>
<td>20.20% (21.57%)</td>
<td>2.04% (3.99%)</td>
<td>1.05% (1.73%)</td>
</tr>
<tr>
<td>1-shot Codex RAW PASS@1000</td>
<td>25.02% (27.77%)</td>
<td>3.70% (7.94%)</td>
<td>3.23% (5.85%)</td>
</tr>
<tr>
<td>1-shot Codex FILTERED PASS@1</td>
<td>22.78% (25.10%)</td>
<td>2.64% (5.78%)</td>
<td>3.04% (5.25%)</td>
</tr>
<tr>
<td>1-shot Codex FILTERED PASS@5</td>
<td>24.52% (27.15%)</td>
<td>3.23% (7.13%)</td>
<td>3.08% (5.53%)</td>
</tr>
</tbody>
</table>

**Note:** passing timeouts in (parentheses)  
**Temperature 0.6 used for sampling all k in pass@k**
Outline

Codex
Introduction
Evaluation
Code Fine-Tuning
Experiments
Supervised Fine-Tuning
Docstring Generation
Discussions

InCoder

CodeGen

2022.11
"Hey, GitHub!"

2022.4
InCoder: Code Infilling and Synthesis

2022.3
CodeGen: Multi-turn Program Synthesis

2021.7
Codex: Evaluating Large Language Models Trained on Code

2020
CodeBERT, PyMT5, TransCoder, ContraCode...
Limitations

Degradation with length of instruction

Experiment: Compose 13 building blocks of <description, function> pair

Chained Building Blocks:
- Concatenate one-line descriptions into docstring

```
1. "remove all instances of the letter e from the string"
   s = s.replace("e", ")

2. "replace all spaces with exclamation points in the string"
   s = s.replace(" ", "!")

3. "convert the string s to lowercase"
   s = s.lower()

4. "remove the first and last two characters of the string"
   s = s[2:-2]

5. "removes all vowels from the string"
   s = ".join(char for char in s if char not in "aeiouAEIOU")
```
Limitations

Degradation with length of instruction

![Graph showing synthetic pass rate vs components (Codex 12B)]
Hazard Analysis

**Over-Reliance**: Codex may generate incorrect code that looks fine to novice programmers

**Misalignment**: Training distribution misalign with the intention of programmers

**Bias**: Biased prompting setups lead to biased generations

---

A model is **intent misaligned** if outputs B, in a scenario where the user prefers output A and the model is both:

1. capable of outputting A
2. capable of distinguishing situations where the user prefers A or B
Outline

Codex

CodeGen

InCoder

Codex for NLP
CodeGen

Open-source alternative to Codex
16B params

Otherwise, very similar to Codex - albeit with a different exact dataset

Slightly outperforms Codex for all k on pass@k on HumanEval

Multi-turn evaluation provides a performance boost
Similar to chain of thought

<table>
<thead>
<tr>
<th>Model</th>
<th>pass@k [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>k = 1</td>
</tr>
<tr>
<td>GPT-NEO 350M</td>
<td>0.85</td>
</tr>
<tr>
<td>GPT-NEO 2.7B</td>
<td>6.41</td>
</tr>
<tr>
<td>GPT-J 6B</td>
<td>11.62</td>
</tr>
<tr>
<td>CODEX 300M</td>
<td>13.17</td>
</tr>
<tr>
<td>CODEX 2.5B</td>
<td>21.36</td>
</tr>
<tr>
<td>CODEX 12B</td>
<td>28.81</td>
</tr>
<tr>
<td>CODEGEN-NL 350M</td>
<td>2.12</td>
</tr>
<tr>
<td>CODEGEN-NL 2.7B</td>
<td>6.70</td>
</tr>
<tr>
<td>CODEGEN-NL 6.1B</td>
<td>10.43</td>
</tr>
<tr>
<td>CODEGEN-NL 16.1B</td>
<td>14.24</td>
</tr>
<tr>
<td>CODEGEN-MULT 350M</td>
<td>6.67</td>
</tr>
<tr>
<td>CODEGEN-MULT 2.7B</td>
<td>14.51</td>
</tr>
<tr>
<td>CODEGEN-MULT 6.1B</td>
<td>18.16</td>
</tr>
<tr>
<td>CODEGEN-MULT 16.1B</td>
<td>18.32</td>
</tr>
<tr>
<td>CODEGEN-MONO 350M</td>
<td>12.76</td>
</tr>
<tr>
<td>CODEGEN-MONO 2.7B</td>
<td>23.70</td>
</tr>
<tr>
<td>CODEGEN-MONO 6.1B</td>
<td>26.13</td>
</tr>
<tr>
<td>CODEGEN-MONO 16.1B</td>
<td><strong>29.28</strong></td>
</tr>
</tbody>
</table>
AlphaCode

Goal: solving competition problems

Similar approach, except:
- 41B parameters
- encoder-decoder model
- larger sampling

Avoiding data-leakage of HumanEval
time-separated

Performed on par with a “median” human competitor
Outline

Codex
CodeGen
InCoder
Codex for NLP
**Motivation**: supports both left-to-right generation and infilling arbitrary blocks of code
Causal Masking

**Training**

Given document size $s$, select

- Number of masks: $n \sim \text{Clamp} \left( \text{Poisson}(1), 1, 16 \right)$
- Length of each mask: $m \sim \left( \text{Uniform}(0, s), \text{Uniform}(0, s) \right)$
Causal Masking

**Training**

Given chosen spans, remove corresponding blocks with *mask sentinel token*
Move blocks of code to the end of file, each separated by `<EOM>`
Causal Masking

Training

Maximize $\log P([\text{left}; \langle \text{Mask}, 0 \rangle; \text{right}; \langle \text{Mask} 0 \rangle; \text{SPAN}; \langle \text{EOM} \rangle])$
Causal Masking

**Inferencing**

Prompt generation through feeding the corresponding *mask sentinel token*

Left-to-right generation: Prompt generation with no right context

Infilling:

\[
[A; \text{<Mask:0>}; C; \text{<Mask:1>}; E; \text{<Mask:2>}] \rightarrow [A; \text{<Mask:0>}; C; \text{<Mask:1>}; \text{E}; \text{<Mask:2>}; \text{<Mask:0>}; B; \text{<EOM>}; \text{<Mask:1>}; \text{D}; \text{<EOM>}]\]
## Left-to-Right

<table>
<thead>
<tr>
<th>Model</th>
<th>Size (B)</th>
<th>Python Code (GB)</th>
<th>Other Code (GB)</th>
<th>Other (GB)</th>
<th>Code License</th>
<th>Infill?</th>
<th>HE @1</th>
<th>HE @10</th>
<th>HE @100</th>
<th>MBPP @1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Released</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CodeParrot [61]</td>
<td>1.5</td>
<td>50</td>
<td>None</td>
<td>None</td>
<td>—</td>
<td>—</td>
<td>4.0</td>
<td>8.7</td>
<td>17.9</td>
<td>—</td>
</tr>
<tr>
<td>PolyCoder [68]</td>
<td>2.7</td>
<td>16</td>
<td>238</td>
<td>None</td>
<td>—</td>
<td>—</td>
<td>5.6</td>
<td>9.8</td>
<td>17.7</td>
<td>—</td>
</tr>
<tr>
<td>GPT-J [63, 18]</td>
<td>6</td>
<td>6</td>
<td>90</td>
<td>730</td>
<td>—</td>
<td>—</td>
<td>11.6</td>
<td>15.7</td>
<td>27.7</td>
<td>—</td>
</tr>
<tr>
<td>INCODER-6.7B</td>
<td>6.7</td>
<td>52</td>
<td>107</td>
<td>57</td>
<td>Permissive</td>
<td>✓</td>
<td>15.2</td>
<td>27.8</td>
<td>47.0</td>
<td>19.4</td>
</tr>
<tr>
<td>GPT-NeoX [14]</td>
<td>20</td>
<td>6</td>
<td>90</td>
<td>730</td>
<td>—</td>
<td>—</td>
<td>15.4</td>
<td>25.6</td>
<td>41.2</td>
<td>—</td>
</tr>
<tr>
<td>CodeGen-Multi [46]</td>
<td>6.1</td>
<td>62</td>
<td>375</td>
<td>1200</td>
<td>—</td>
<td>—</td>
<td>18.2</td>
<td>28.7</td>
<td>44.9</td>
<td>—</td>
</tr>
<tr>
<td>CodeGen-Mono [46]</td>
<td>6.1</td>
<td>279</td>
<td>375</td>
<td>1200</td>
<td>—</td>
<td>—</td>
<td>26.1</td>
<td>42.3</td>
<td>65.8</td>
<td>—</td>
</tr>
<tr>
<td>CodeGen-Mono [46]</td>
<td>16.1</td>
<td>279</td>
<td>375</td>
<td>1200</td>
<td>—</td>
<td>—</td>
<td>29.3</td>
<td>49.9</td>
<td>75.0</td>
<td>—</td>
</tr>
<tr>
<td><strong>Unreleased</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LaMDA [10, 60, 21]</td>
<td>137</td>
<td>None</td>
<td>None</td>
<td>???</td>
<td>—</td>
<td>—</td>
<td>14.0</td>
<td>—</td>
<td>47.3</td>
<td>14.8</td>
</tr>
<tr>
<td>AlphaCode [44]</td>
<td>1.1</td>
<td>54</td>
<td>660</td>
<td>None</td>
<td>—</td>
<td>—</td>
<td>17.1</td>
<td>28.2</td>
<td>45.3</td>
<td>—</td>
</tr>
<tr>
<td>Codex-12B [18]</td>
<td>12</td>
<td>180</td>
<td>None</td>
<td>&gt;570</td>
<td>—</td>
<td>—</td>
<td>28.8</td>
<td>46.8</td>
<td>72.3</td>
<td>—</td>
</tr>
<tr>
<td>PaLM-Coder [21]</td>
<td>540</td>
<td>~20</td>
<td>~200</td>
<td>~4000</td>
<td>Permissive</td>
<td>—</td>
<td>36.0</td>
<td>—</td>
<td>88.4</td>
<td>47.0</td>
</tr>
</tbody>
</table>
Codex
CodeGen
InCoder
CodeGen
Codex for NLP
Code Model on NLP Tasks

**Highlight**: Code Models (e.g. Codex) can outperform LLMs (e.g. GPT3) in NLP tasks that involve structural and logical analysis.
Codex-NLP: Semantic Parsing

Task: convert an utterance $u$ to a semantic meaning representation $m$, viewed either as a generation or classification problem

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Natural language</th>
<th>Canonical utterance</th>
<th>Meaning representation</th>
</tr>
</thead>
</table>
| SMCalFlow   | Schedule Hide and Seek in the mall for Saturday night | create event called "Hide and Seek" starting next Saturday night at "mall" | \{(Yield :output (CreateCommitEventWrapper :event (CreatePreflightEventWrapper :constraint (Constraint[Event] :subject (?= #(String "Hide and Seek") :start (DateTimeConstraint :constraint (Night) :date (NextDOW :dow #(DayOfWeek "SATURDAY") :location (?= #(LocationKeyphrase "mall")))) (call listValue (call superlative (call getProperty (call singleton en.meeting) (string !type)) (string min) (call ensureNumericProperty (string end_time)))))

| Overnight Cal. | which meeting has the earliest end time | meeting that has the smallest end time | |

59
Task: convert an utterance $u$ to a semantic meaning representation $m$, viewed either as a generation or classification problem

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy Overnight Cal.</th>
<th>SMCalFlow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Davinci</td>
<td>0.81</td>
<td>0.340</td>
</tr>
<tr>
<td>Curie</td>
<td>0.66</td>
<td>0.260</td>
</tr>
<tr>
<td>Davinci Codex</td>
<td>0.86</td>
<td>0.355</td>
</tr>
<tr>
<td>Cushman Codex</td>
<td>0.87</td>
<td>0.320</td>
</tr>
</tbody>
</table>
Codex-NLP: Reasoning

Task: Given a event/goal T, generate a commonsense reasoning represented as a graph

(a) The script $G$

(b) $G$ converted to Python code $G_c$ using our approach
Codex-NLP: Reasoning

Task: Given an event/goal T, generate a commonsense reasoning represented as a graph

<table>
<thead>
<tr>
<th></th>
<th>PID</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAVINCI</td>
<td>r1</td>
<td>72.7</td>
<td>50.9</td>
<td>59.8</td>
</tr>
<tr>
<td></td>
<td>r2</td>
<td>75.9</td>
<td>45.6</td>
<td>57.0</td>
</tr>
<tr>
<td></td>
<td>r3</td>
<td>73.8</td>
<td>42.4</td>
<td>53.9</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>74.2±1.3</td>
<td>46.3±3.5</td>
<td>56.9±2.4</td>
</tr>
<tr>
<td>CODEX-002</td>
<td>r1</td>
<td>69.0</td>
<td>54.5</td>
<td>60.9</td>
</tr>
<tr>
<td></td>
<td>r2</td>
<td>84.6</td>
<td>48.1</td>
<td>61.3</td>
</tr>
<tr>
<td></td>
<td>r3</td>
<td>77.6</td>
<td>55.7</td>
<td>66.2</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>77.1±6.4</td>
<td><strong>52.8±3.3</strong></td>
<td><strong>62.8±2.4</strong></td>
</tr>
</tbody>
</table>
Codex-NLP: Proof Synthesis

Task: Given a set of facts and a hypothesis, construct a proof of how the facts conclude at the hypothesis

<table>
<thead>
<tr>
<th>Method</th>
<th>Leaves</th>
<th></th>
<th></th>
<th>Steps</th>
<th></th>
<th></th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F1</td>
<td>AllCorrect</td>
<td>F1</td>
<td>AllCorrect</td>
<td>F1</td>
<td>AllCorrect</td>
</tr>
<tr>
<td>EntailmentWriter</td>
<td>86.2</td>
<td>43.9</td>
<td></td>
<td>40.6</td>
<td>28.3</td>
<td></td>
<td>67.1</td>
</tr>
<tr>
<td>EntailmentWriter (T5-11B)</td>
<td>89.4</td>
<td>52.9</td>
<td></td>
<td>46.6</td>
<td>35.3</td>
<td></td>
<td>69.1</td>
</tr>
<tr>
<td>NLProofS (ours)</td>
<td>89.4 ± 0.8</td>
<td>56.0 ± 0.7</td>
<td></td>
<td>50.4 ± 1.9</td>
<td>38.4 ± 1.3</td>
<td></td>
<td>71.9 ± 1.4</td>
</tr>
<tr>
<td>GPT-3 (Brown et al., 2020)</td>
<td>64.2 ± 2.3</td>
<td>15.3 ± 1.9</td>
<td></td>
<td>17.6 ± 0.6</td>
<td>12.3 ± 1.4</td>
<td></td>
<td>53.6 ± 1.4</td>
</tr>
<tr>
<td>Codex (Chen et al., 2021)</td>
<td>68.9 ± 3.7</td>
<td>19.8 ± 3.2</td>
<td></td>
<td>21.4 ± 3.0</td>
<td>14.6 ± 1.7</td>
<td></td>
<td>55.6 ± 2.2</td>
</tr>
</tbody>
</table>

Table B: Validation results of proof generation on EntailmentBank (Dalvi et al., 2021). Results of GPT-3 and Codex are based on prompting with 7 in-context examples randomly sampled from the training data.
Q3. As a programmer, what types of features would you like to use from a code model (describe the use cases in detail)? Do you think current code models already achieve that, or what improvements need to be done in the future?
Q3. As a programmer, what types of features would you like to use from a code model (describe the use cases in detail)? Do you think current code models already achieve that, or what improvements need to be done in the future?

Features

- Autocomplete:
  Given comment and some written code to start with, the model should be able to complete a block of code that matches the intention of function, variable names, and be able to call (if needed) other functions / imported libraries

- Docstring Formatting
  Given function signature and code, infill missing components in the function docstring to include variable type, description, return type, etc.

- Code Search
  Given a description of intention of function, search a codebase if such function is already implemented

......
Q3. As a programmer, what types of features would you like to use from a code model (describe the use cases in detail)? Do you think current code models already achieve that, or what improvements need to be done in the future?

Discussion

Code model that can fill in blanks in code or docstring would be better than left-to-right only generation for many reasons (e.g. to call functions declared both above and below a target function). HumanEval focuses on standalone functions, which may create a potential misalignment between the model’s trained goal and intention of programmers.
References

(apart from the ones on the website):

AlphaCode: https://arxiv.org/abs/2203.07814
Natural Language Proofs: https://arxiv.org/abs/2205.12443