Contextual Communication in Programming

Daniel Fried
Today’s Question

Are bigger models the solution for AI-assisted programming?
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]
Function pass rate on a Python synthesis dataset [Chen et al. 2021] by data & model scale:

[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]
Are bigger models the solution for AI-assisted programming?

YES
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: [Fried & Aghajanyan et al., 2022]
  - Partial Code: [Fried & Aghajanyan et al., 2022]
  - Tests & Execution: [Shi et al. 2022]

- **As Outputs**
  - [Fried & Aghajanyan et al., 2022]
  - Edits to Code: [Wallace et al., in progress]

- **Deictic (Pointing / Highlighting)**

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

As Inputs
- Natural Language: [Fried & Aghajanyan et al., 2022]
- Partial Code: [Fried & Aghajanyan et al., 2022]

As Outputs
- Tests & Execution: [Shi et al. 2022]
- Edits to Code: [Fried & Aghajanyan et al., 2022]
- Deictic (Pointing / Highlighting): [Wallace et al., in progress]

Modality Choice
- [Lin et al., 2022]
Neural Code Model 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  
[Donahue+ 2020, Aghajanyan+ 2022, ours, Bavarian+ 2022]
InCoder: Code Generation and Infilling

Training

Original Document

```
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
```

Masked Document

```
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:
        <MASK:0> in word_counts:
            word_counts[word] += 1
        else:
            word_counts[word] = 1
        return word_counts
```

[<NAME>*, <NAME>* et al., 2022]
InCoder: Code Generation and Infilling

- **Data**
  - 600K permissively-licensed repositories from GitHub & GitLab
  - StackOverflow: questions, answers, comments

- **Models**
  - Standard transformer LM
  - 1B model: ~1 week on 128 GPUs
  - 6B model: ~3 weeks on 240 GPUs
InCoder: Code Generation and 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, 'r') as file:
        for line in file:
            words.append(line.strip())
    return Counter(words)
```

Also usable as a left-to-right generation model with no apparent loss in performance [see also Bavarian et al. 2022]
Evaluation

- Zero-shot evaluation on realistic code infilling tasks
- Compare the model in three different modes to evaluate benefits of suffix context

<table>
<thead>
<tr>
<th>Baselines</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Left-to-Right Single</strong></td>
<td><strong>Causal Masking</strong></td>
</tr>
<tr>
<td>Doesn’t use suffix</td>
<td>Uses suffix when generating</td>
</tr>
<tr>
<td><strong>Left-to-Right Rerank</strong></td>
<td></td>
</tr>
<tr>
<td>Only uses suffix when reranking</td>
<td></td>
</tr>
</tbody>
</table>

**Ours**
- Uses suffix when generating
**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>
<tr>
<td>RoBERTa (Finetuned)</td>
<td>18.14</td>
</tr>
<tr>
<td>CodeBERT (Finetuned)</td>
<td>19.06</td>
</tr>
<tr>
<td>PLBART (Finetuned)</td>
<td>19.30</td>
</tr>
<tr>
<td>CodeT5 (Finetuned)</td>
<td>20.36</td>
</tr>
</tbody>
</table>
Evaluation

Type Inference

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>59.2</td>
</tr>
<tr>
<td>TypeWriter (Supervised)</td>
<td>48.3</td>
</tr>
</tbody>
</table>
Demo

Model Weights & Interactive Demo: [huggingface.co/facebook/incoder-1B](https://huggingface.co/facebook/incoder-1B)
Communicating with Multiple Modalities

As Inputs
- NL Instructions
  - [Fried & Aghajanyan et al., 2022]
- Partial Code
  - [Fried & Aghajanyan et al., 2022]
- Tests & Execution
  - [Shi et al. 2022]
- Edits to Code
  - [Fried & Aghajanyan et al., 2022]
- Deictic (Pointing / Highlighting)
  - [Wallace et al., in progress]

As Outputs
- [Lin et al., 2022]
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.""

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

Minimum Bayes Risk with Execution:

Cluster by Outputs

[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 ...
Thanks!

dfried@cs.cmu.edu
dpfried.github.io
## Scale and Performance

<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>
... and now
Details

- Remove duplicate files using exact match on alphanumeric token sequences
- Tokenization: retrain byte-level BPE, modified to allow merging across spaces
  - e.g. `import numpy as np` is a single token
    - 40% reduction in token count compared to GPT-2's tokenizer
      - Longer effective contexts & more efficient training
      - But, less compute spent in training may affect performance
- Meta-data conditioning and prediction
  - `<| file source=github stars=high filename=setup.py ext=.py |>
  - Attributes can appear at beginning of the file (conditioning) or end (prediction)
def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = []
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words

def count_words(filename):
    """Count the number of occurrences of each word in the file"""
    words = []
    with open(filename, 'r') as file:
        for line in file:
            line = line.lower().strip()
            for word in line.split():
                if word not in words:
                    words[word] = 0
                words[word] += 1
    return words

return words

return words