Code Pretraining

Daniel Fried
11-891: Neural Code Generation
https://cmu-codegen.github.io/s2024/

With slides from Greg Durrett, Nikitha Rao, and Zora Wang
Train the model to generate language/code, then use -- without updating the model -- on other generation tasks.

def count_lines(__filename
def count_words(__filename
Pre-train and Fine-Tune

First train on one task, then train on another

```python
def count_lines__ (Model) → filename
```

- **Generation**
  ```python
def count_lines__ (Model) → Python
```
- **Classification**

---

**Model Initialization**
Objectives: Autoregressive Language Modeling

\[ P(X) = \prod_{i=1}^{\vert X \vert} P(x_i \mid x_1, \ldots, x_{i-1}) \]

Outputs: count_, lines, (, filename

Inputs: def, count_, lines, (, Used mostly for generation/prompting
Objectives: Masked Language Modeling

\[ P(X) \neq \prod_{i=1}^{\mid X \mid} P(x_i \mid x \neq i) \]

Outputs:

Inputs:

Used mostly for representation learning
Unidirectional vs Bidirectional Transformers

**Unidirectional**
Each token has info about previous.

```python
def count_lines(filename)
```

**Bidirectional**
Each token has info about all others.

```python
def count_lines([MASK])
```
Objectives: Sequence-to-Sequence

\[ P(Y|X) = \prod_{i=1}^{\lfloor Y \rfloor} P(y_i|X, y_1, \ldots, y_{i-1}) \]

Bidirectional Transformer (Encoder)

Unidirectional Transformer (Decoder)

"Count the lines in the

def count_lines (filename

Used mostly for translation tasks, with fine-tuning.
**Which Objective?**

**Autoregressive language modeling**
\[ P(X) = \prod_{i=1}^{\vert X \vert} P(x_i|x_1, \ldots, x_{i-1}) \]
used more for prompting/text generation

**Masked language modeling**
\[ P(X) \neq \prod_{i=1}^{\vert X \vert} P(x_i|x_{\neq i}) \]
used more for pre-training + fine-tuning

**Seq-to-seq de-noising**
\[ P(Y|X) = \prod_{i=1}^{\vert Y \vert} P(y_i|X, y_1, \ldots, y_{i-1}) \]
used for pre-train + fine-tune on generation tasks
Autoregressive Generation

\[ P(X) = \prod_{i=1}^{\left| X \right|} P(x_i | x_1, \ldots, x_{i-1}) \]
OpenAI GPT/GPT2

- Very large language models using the Transformer architecture
- Straightforward unidirectional decoder language model, trained on raw text
- GPT2: trained on 40GB of text

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Layers</th>
<th>$d_{model}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>117M</td>
<td>12</td>
<td>768</td>
</tr>
<tr>
<td>345M</td>
<td>24</td>
<td>1024</td>
</tr>
<tr>
<td>762M</td>
<td>36</td>
<td>1280</td>
</tr>
<tr>
<td>1542M</td>
<td>48</td>
<td>1600</td>
</tr>
</tbody>
</table>

GPT-2

- By far the largest of these models trained when it came out in March 2019
- Because it's a language model, we can generate from it

Radford et al. (2019)
Pushing the Limits: GPT-3

- 175B parameter model: 96 layers, 96 heads, 12k-dim vectors

- Trained on Microsoft Azure, estimated to cost roughly $10M
Autoregressive Language Modeling for Code

- Typically trained on lots of code from GitHub, often mixed with text.

- Codex (Chen et al. 2021): OpenAI continues to train GPT-3 12B on 160GB of Python data from GitHub.

- All GPT 3.5 models are trained on mixtures of code and text. [https://platform.openai.com/docs/model-index-for-researchers](https://platform.openai.com/docs/model-index-for-researchers)

- Many open-source models since then follow this recipe (PolyCoder,CodeGen, StarCoder).
Codex: “HumanEval” Benchmark

- Evaluation: test case execution
- 164 hand-written examples
- Why human-written?
  - "It is important for these tasks to be hand-written, since our models are trained on a large fraction of GitHub, which already contains solutions to problems from a variety of sources."
- Optimizing BLEU != Improving Functional Correctness
Sampling-Based Evaluation

- Sampling more candidate functions dramatically increases chance of correctness
- pass@k: sample $k$ candidate functions; see if any pass
- Many ways of combining/using multiple candidates to help improve code correctness --- more in a future lecture!
Codex: Scaling Laws

**Codex Loss Scaling**

- Test loss vs Non-embedding parameters
- Equation: \( N \left(5.925 \times 0.07\right)^{-0.13} \)

**Pass Rate vs Model Size**

- Graphs showing pass rates at different model sizes for different values of \( T^* \)
- Pass@1 \( T^*=0.2 \)
- Pass@100 \( T^*=0.8 \)
Models Generate Good and Bad Code!

Figure 12. When the prompt includes subtle bugs, Codex tends to produce worse code than it is capable of. This persists when the prompt also includes instructions to write correct code. This gap increases with model size.
Masked Language Modeling

\[ P(X) \neq \prod_{i=1}^{\lfloor X \rfloor} P(x_i | x \neq i) \]

used more for pre-training + fine-tuning
CodeBERT: Masked Language Modeling Objective

Mask 15% of the tokens, randomly, and try to predict these masked tokens.

\[
\mathcal{L}_{\text{MLM}}(\theta) = \sum_{i \in m^w \cup m^c} -\log p_{D_1}(x_i | w^\text{masked}, c^\text{masked})
\]

---

**Code**

```
[CLS]          def        count_

...               [SEP]

Count the

Docstring
```
CodeBERT: Replaced Token Detection Objective

Rather than masked tokens, use tokens replaced by (weaker) LMs, and distinguish original tokens from replaced tokens.
CodeBERT: Pre-Training

- 125M parameter bidirectional encoder Transformer
- Train on 2M documented functions (text & code) and 6M undocumented functions (code only) from GitHub (CodeSearchNet)

<table>
<thead>
<tr>
<th>TRAINING DATA</th>
<th>bimodal DATA</th>
<th>unimodal CODES</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO</td>
<td>319,256</td>
<td>726,768</td>
</tr>
<tr>
<td>JAVA</td>
<td>500,754</td>
<td>1,569,889</td>
</tr>
<tr>
<td>JAVASCRIPT</td>
<td>143,252</td>
<td>1,857,835</td>
</tr>
<tr>
<td>PHP</td>
<td>662,907</td>
<td>977,821</td>
</tr>
<tr>
<td>PYTHON</td>
<td>458,219</td>
<td>1,156,085</td>
</tr>
<tr>
<td>RUBY</td>
<td>52,905</td>
<td>164,048</td>
</tr>
<tr>
<td>ALL</td>
<td>2,137,293</td>
<td>6,452,446</td>
</tr>
</tbody>
</table>
CodeBERT: Finetuning

Parts of the task network are initialized with CodeBERT parameters.

Classification Tasks

- Input tokens: `[CLS] text/code [SEP] code [SEP]`
- CodeBERT
- FFNN + Softmax
- 0, 1 - Category distribution

Supported tasks:
- code search
- code clone detection

Generation Tasks

- Input code
- CodeBERT as Encoder
- Decoder
- Output code

Supported tasks:
- code repair
- code translation
# CodeXGLUE Benchmark

Collection of tasks, largely with natural data mined from GitHub

<table>
<thead>
<tr>
<th>Category</th>
<th>Task</th>
<th>Dataset Name</th>
<th>Language</th>
<th>Train/Dev/Test Size</th>
<th>Baselines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clone Detection</td>
<td>BigCloneBench [71]</td>
<td>Java</td>
<td>900K/416K/416K</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POJ-104 [52]</td>
<td>C/C++</td>
<td>32K/8K/12K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defect Detection</td>
<td>Devign [99]</td>
<td>C</td>
<td>21K/2.7K/2.7K</td>
<td></td>
<td>CodeBERT</td>
</tr>
<tr>
<td>Code-Code</td>
<td>Cloze Test</td>
<td>CT-all</td>
<td>Python,Java,PHP,</td>
<td>~/-/176K</td>
<td>CodeBERT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CT-max/min [18]</td>
<td>JavaScript,Ruby,Go</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Completion</td>
<td>PY150 [62]</td>
<td>Python</td>
<td>100K/5K/50K</td>
<td></td>
<td>CodeGPT</td>
</tr>
<tr>
<td></td>
<td>Github Java Corpus[4]</td>
<td>Java</td>
<td>13K/7K/8K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Repair</td>
<td>Bugs2Fix [75]</td>
<td>Java</td>
<td>98K/12K/12K</td>
<td></td>
<td>Encoder-Decoder</td>
</tr>
<tr>
<td>Code Translation</td>
<td>CodeTrans</td>
<td>Java-C#</td>
<td>10K/0.5K/1K</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CodeSearchNet [35], WebQueryTest</td>
<td>Python</td>
<td>251K/9.6K/1K</td>
<td></td>
</tr>
<tr>
<td>Text-to-Code Generation</td>
<td>CONCODE [38]</td>
<td>Java</td>
<td>100K/2K/2K</td>
<td></td>
<td>CodeGPT</td>
</tr>
<tr>
<td>Text-Text</td>
<td>Documentation Translation</td>
<td>Microsoft Docs</td>
<td>English-Latvian/Danish</td>
<td>156K/4K/4K</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/Norwegian/Chinese</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CodeBERT: Results

- Joint training on code and documentation > code alone
- Initializing with a text-only model (RoBERTa) helps

<table>
<thead>
<tr>
<th>MODEL</th>
<th>RUBY</th>
<th>JAVASCRIPT</th>
<th>GO</th>
<th>PYTHON</th>
<th>JAVA</th>
<th>PHP</th>
<th>MA-AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoBERTA</td>
<td>0.6245</td>
<td>0.6060</td>
<td>0.8204</td>
<td>0.8087</td>
<td>0.6659</td>
<td>0.6576</td>
<td>0.6972</td>
</tr>
<tr>
<td>PT w/ Code Only (init=s)</td>
<td>0.5712</td>
<td>0.5557</td>
<td>0.7929</td>
<td>0.7855</td>
<td>0.6567</td>
<td>0.6172</td>
<td>0.6632</td>
</tr>
<tr>
<td>PT w/ Code Only (init=R)</td>
<td>0.6612</td>
<td>0.6402</td>
<td>0.8191</td>
<td>0.8438</td>
<td>0.7213</td>
<td>0.6706</td>
<td>0.7260</td>
</tr>
<tr>
<td>CodeBERT (MLM, init=s)</td>
<td>0.5695</td>
<td>0.6029</td>
<td>0.8304</td>
<td>0.8261</td>
<td>0.7142</td>
<td>0.6556</td>
<td>0.6998</td>
</tr>
<tr>
<td>CodeBERT (MLM, init=R)</td>
<td>0.6898</td>
<td>0.6997</td>
<td>0.8383</td>
<td>0.8647</td>
<td>0.7476</td>
<td>0.6893</td>
<td>0.7549</td>
</tr>
<tr>
<td>CodeBERT (RTD, init=R)</td>
<td>0.6414</td>
<td>0.6512</td>
<td>0.8285</td>
<td>0.8263</td>
<td>0.7150</td>
<td>0.6774</td>
<td>0.7233</td>
</tr>
<tr>
<td>CodeBERT (MLM+RTD, init=R)</td>
<td><strong>0.6926</strong></td>
<td><strong>0.7059</strong></td>
<td><strong>0.8400</strong></td>
<td><strong>0.8685</strong></td>
<td><strong>0.7484</strong></td>
<td><strong>0.7062</strong></td>
<td><strong>0.7603</strong></td>
</tr>
</tbody>
</table>

Results for function/documentation matching (code retrieval)
CodeBERT: Results

- Joint training on code and documentation > code alone
- Initializing with a text-only model (RoBERTa) helps

<table>
<thead>
<tr>
<th>MODEL</th>
<th>RUBY</th>
<th>JAVASCRIPT</th>
<th>GO</th>
<th>PYTHON</th>
<th>JAVA</th>
<th>PHP</th>
<th>OVERALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ2SEQ</td>
<td>9.64</td>
<td>10.21</td>
<td>13.98</td>
<td>15.93</td>
<td>15.09</td>
<td>21.08</td>
<td>14.32</td>
</tr>
<tr>
<td>TRANSFORMER</td>
<td>11.18</td>
<td>11.59</td>
<td>16.38</td>
<td>15.81</td>
<td>16.26</td>
<td>22.12</td>
<td>15.56</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>11.17</td>
<td>11.90</td>
<td>17.72</td>
<td>18.14</td>
<td>16.47</td>
<td>24.02</td>
<td>16.57</td>
</tr>
<tr>
<td>PRE-TRAIN W/ CODE ONLY</td>
<td>11.91</td>
<td>13.99</td>
<td>17.78</td>
<td>18.58</td>
<td>17.50</td>
<td>24.34</td>
<td>17.35</td>
</tr>
<tr>
<td>CodeBERT (RTD)</td>
<td>11.42</td>
<td>13.27</td>
<td>17.53</td>
<td>18.29</td>
<td>17.35</td>
<td>24.10</td>
<td>17.00</td>
</tr>
<tr>
<td>CodeBERT (MLM)</td>
<td>11.57</td>
<td>14.41</td>
<td>17.78</td>
<td>18.77</td>
<td>17.38</td>
<td>24.85</td>
<td>17.46</td>
</tr>
<tr>
<td>CodeBERT (RTD+MLM)</td>
<td><strong>12.16</strong></td>
<td><strong>14.90</strong></td>
<td><strong>18.07</strong></td>
<td><strong>19.06</strong></td>
<td><strong>17.65</strong></td>
<td><strong>25.16</strong></td>
<td><strong>17.83</strong></td>
</tr>
</tbody>
</table>

Results for function-to-docstring generation
CodeBERT: Masked Prediction Probing

Figure 3: Case study on python language. Masked tokens in NL (in blue) and PL (in yellow) are separately applied. Predicted probabilities of RoBERTa and CodeBERT are given.
Filling-in-the-Middle
```
def minimize_in_graph(build_loss_fn, num_steps=200, optimizer=None):
    
    def build_loss_fn(...)
    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)

Masked Infilling

“Causal Masking” / Fill-in-the-Middle (FIM)

[e.g. GPT-*, Codex]

[e.g. BERT, CodeBERT]

[Donahue+ 2020, Aghajanyan+ 2022, ours, Bavarian+ 2022]
Causal Masking / FIM Objective

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

[Donahue et al. 2020, Aghajanyan et al. 2022, Fried et al. 2022, Bavarian et al. 2022]
InCoder: Model Training

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

- **Models**
  - Unidirectional, decoder-only Transformer
  - 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
Evaluation: Function Completion

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]
Function completion

Single-Line Infilling

Fraction of Lines in Right Context

Pass Rate

- CM Infilling
- L-R Single
- L-R Reranking

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

[TypeWriter OSS, Pradel et al. 2020]
**Evaluation**

**Variable Name Prediction**

```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_count = {}
        for line in f:
            for word in line.split():
                if word in word_count:
                    word_count[word] += 1
                else:
                    word_count[word] = 1
        return word_count
```

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left-to-right single</td>
<td>18.4</td>
</tr>
<tr>
<td>Left-to-right reranking</td>
<td>23.5</td>
</tr>
<tr>
<td>Causal-masked infilling</td>
<td>30.6</td>
</tr>
</tbody>
</table>
### Ablations

- StackOverflow data improves performance
- Roughly comparable performance from infilling and non-infilling models (but see Ben Allal et al. 2022 and Nijkamp et al. 2023)

<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>
Other Infilling Code Models

Efficient Training of Language Models to Fill in the Middle

Mohammad Bavarian
Heewoo Jun
Nikolas Tezak
John Schulman
Christine McLeavey
Jerry Tworek
Mark Chen

OpenAI

SANTA CODER: DON’T REACH FOR THE STARS! 🌟

Loubna Ben Allal*
Hugging Face
Raymond Li*
ServiceNow Research
Denis Kocetkov*
ServiceNow Research

STAR CODER: MAY THE SOURCE BE WITH YOU!

Raymond Li² Loubna Ben Allal¹ Yangtian Zi¹ Niklas Muennighoff¹ Denis Kocetkov² Chenghao Mou³ Marc Marone⁸ Christopher Akiki⁹,¹⁰ Jia Li⁵ Jenny Chim¹¹ Qian Liu¹³

CODE LLAMA: OPEN FOUNDATION MODELS FOR CODE

Baptiste Rozière¹, Jonas Gehring¹, Fabian Gloeckle¹*, Sten Sootla¹, Itai Gat, Xiaoping Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattaflori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, Gabriel Synnaeve¹

CODGEN2: LESSONS FOR TRAINING LLMs ON PROGRAMMING AND NATURAL LANGUAGES

Erik Nijkamp*, Hiroaki Hayashi*, Caiming Xiong, Silvio Savarese, Yingbo Zhou
Demo:

```python
def <infill>
    """Count the number of occurrences of each word in the file."""
    <infill>
```
Encoder-Decoder LMs

\[ P(Y|X) = \prod_{i=1}^{\frac{|Y|}{2}} P(y_i|X, y_1, \ldots, y_{i-1}) \]

used for pre-train + fine-tune on generation tasks
How do we pre-train seq2seq models?

- LMs $P(x)$: trained unidirectionally
- Masked LMs: trained bidirectionally but with masking
- How can we pre-train a model for $P(y|x)$?
- Well, why was BERT effective?
  - Predicting a mask requires some kind of text “understanding”.
- What would it take to do the same for sequence prediction?
- Requirements: (1) should use unlabeled data; (2) should force a model to attend from $y$ back to $x$
Several possible strategies for corrupting a sequence are explored in the BART paper.

- Token Masking
- Sentence Permutation
- Document Rotation
- Token Deletion
- Text Infilling

Infilling is longer spans than masking.

Lewis et al. (2019)
BART

- **Model & Objective**: Sequence-to-sequence Transformer trained on this data: permute/make/delete tokens, then predict full sequence autoregressively

- **Data**: Same as RoBERTa; 160 GB of text

Lewis et al. (2019)
BERT vs. BART

- BERT: only parameters are an encoder, trained with masked language modeling objective. Cannot generate text or do seq2seq tasks

- BART: both an encoder and a decoder. Can also use just the encoder wherever we would use BERT

Lewis et al. (2019)
T5: Text-to-Text Transfer Transformer

- **Objective**: similar denoising scheme to BART (they were released within a week of each other in fall 2019).
- Lower computational cost compared to BART: predicts fewer tokens.

---

Original text:

Thank you for inviting me to your party last week.

Inputs:

Thank you <X> me to your party <Y> week.

Targets:

<X> for inviting <Y> last <Z>
CodeT5: Objectives

Pre-train like T5 (seq-to-seq denoising/masked span prediction), but add identifier-specific objectives to learn code semantics.

Like T5

- **Masked Input**
  
  ```python
  def binarySearch(arr, left, right, x):
      mid = (left + right) // 2
      if arr[mid] == x:
          return mid
  
  MASK0 binary search MASK1 right // 2
  MASK2 [ mid ]
  ```

- **Output**
  
  ```python
  MASK0 binary search MASK1 right // 2
  MASK2 [ mid ]
  ```

(c) Masked Identifier Prediction

(b) Identifier Tagging

0 1 0 1 0 0 1 0 0 1

If arr [ mid ] == x : return mid

(d) Bimodal Dual Generation

- **Bimodal Input**
  
  ```python
  def binarySearch(arr, left, right, x):
      mid = (left + right) // 2
      if arr[mid] == x:
          return mid
  
  # recursive binary search
  ```

Figure 2: Pre-training tasks of CodeT5. We first alternately train span prediction, identifier prediction, and identifier tagging on both unimodal and bimodal data, and then leverage the bimodal data for dual generation training.
Pre-train on CodeSearchNet (6 PLs) + BigQuery (C & C#); 8.4M instances
  ▶ 60M and 220M parameter models, trained for 5 & 12 days on 16 GPUs.
  ▶ Couldn’t initialize with T5, because T5’s tokenizer doesn’t preserve code-specific symbols like { and }. Train own tokenizer (more in a future class!)
Then, optionally do **multi-task fine-tuning**: train on multiple seq-to-seq tasks from CodeXGLUE simultaneously (translation, refinement, summarization, ...).
CodeT5: Analysis

- All components of the objective help. MSP: masked span prediction. IT: identifier tagging. MIP: masked identifier prediction

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeT5</td>
<td>20.04</td>
<td>41.39</td>
<td>19.06</td>
<td>63.40</td>
</tr>
<tr>
<td>-MSP</td>
<td>18.93</td>
<td>37.44</td>
<td>15.92</td>
<td>64.02</td>
</tr>
<tr>
<td>-IT</td>
<td>19.73</td>
<td>39.21</td>
<td>18.65</td>
<td>63.29</td>
</tr>
<tr>
<td>-MIP</td>
<td>19.81</td>
<td>38.25</td>
<td>18.32</td>
<td>62.92</td>
</tr>
</tbody>
</table>
CodeT5: Analysis

- Multi-task fine-tuning sometimes helps and sometimes hurts, with some effects from task similarity.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Java to C#</th>
<th>C# to Java</th>
<th>Refine Small</th>
<th>Refine Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BLEU</td>
<td>EM</td>
<td>BLEU</td>
<td>EM</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>79.92</td>
<td>59.00</td>
<td>72.14</td>
<td>58.80</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td>80.58</td>
<td>59.40</td>
<td>72.64</td>
<td>58.80</td>
</tr>
<tr>
<td>PLBART</td>
<td>83.02</td>
<td>64.60</td>
<td>78.35</td>
<td>65.00</td>
</tr>
<tr>
<td>CodeT5-small</td>
<td>82.98</td>
<td>64.10</td>
<td>79.10</td>
<td>65.60</td>
</tr>
<tr>
<td>+dual-gen</td>
<td>82.24</td>
<td>63.20</td>
<td>78.10</td>
<td>63.40</td>
</tr>
<tr>
<td>+multi-task</td>
<td>83.49</td>
<td>64.30</td>
<td>78.56</td>
<td>65.40</td>
</tr>
<tr>
<td>CodeT5-base</td>
<td><strong>84.03</strong></td>
<td><strong>65.90</strong></td>
<td><strong>79.87</strong></td>
<td><strong>66.90</strong></td>
</tr>
<tr>
<td>+dual-gen</td>
<td>81.84</td>
<td>62.00</td>
<td>77.83</td>
<td>63.20</td>
</tr>
<tr>
<td>+multi-task</td>
<td>82.31</td>
<td>63.40</td>
<td>78.01</td>
<td>64.00</td>
</tr>
</tbody>
</table>

Code translation and refinement results.
Hybrid Models
Specializations of past approaches:

- For **translation**: T5-like (seq-to-seq denoising) generally best
- For **generating new content**: GPT-like (unidirectional decoder-only) generally best
- For **doc-level embeddings**: BERT-like (MLM bidirectional encoder) generally best

CodeT5+: use a seq-to-seq model but train it with a progression of objectives, and pre-trained initializations

Wang et al. (2023)
CodeT5+: Overview

Generation Tasks
- Text-To-Code Generation, Math Programming, Code Summarization, Code Completion

Understanding Tasks
- Text-To-Code Retrieval, Code Defect Detection, Code Clone Detection

Retrieval-augmented Generation

Unimodal Code Data
- 1st pretraining
- Encoder-Decoder LM
- Span Denoising

Bimodal Code-Text Data
- 2nd pretraining
- Encoder-Decoder LM
- Causal LM

Encoder-only LM
- activate different modules

Decoder-only LM
- Contrastive Loss

Encoder-Decoder LM
- Matching Loss

zero-shot/finetuning/instruction-tuning
CodeT5+: Supports downstream tasks

- **Unimodal Code Data**
  - 1st pretraining
  - Encoder-Decoder LM

- **Bimodal Code-Text Data**
  - 2nd pretraining
  - Encoder-Decoder LM

- **Generation Tasks**
  - Text-To-Code Generation
  - Math Programming
  - Code Summarization
  - Code Completion

- **Understanding Tasks**
  - Text-To-Code Retrieval
  - Code Defect Detection
  - Code Clone Detection

- **Encoder-only LM**
- **Decoder-only LM**
- **Encoder-Decoder LM**

- **Span Denoising**
- **Causal LM**
- **Contrastive Loss**
- **Matching Loss**

- zero-shot/finetuning/instruction-tuning
CodeT5+: Can operate in different modes

- Unimodal Code Data
  - 1st pretraining
- Encoder-Decoder LM
- Bimodal Code-Text Data
  - 2nd pretraining
- Encoder-Decoder LM
- activate different modules
- Encoder-only LM
- Decoder-only LM
- Encoder-Decoder LM
- Generation Tasks
  - Text-To-Code Generation, Math Programming, Code Summarization, Code Completion
- Understanding Tasks
  - Text-To-Code Retrieval, Code Defect Detection, Code Clone Detection
- Retrieval-augmented Generation
- Span Denoising
- Causal LM
- Contrastive Loss
- Matching Loss
- zero-shot/finetuning/instruction-tuning
CodeT5+: Uses several pre-training tasks

- **Unimodal Code Data**
  - Encoder-Decoder LM
  - 1st pretraining

- **Bimodal Code-Text Data**
  - Encoder-Decoder LM
  - 2nd pretraining
  - activate different modules

- **Generation Tasks**
  - Text-To-Code Generation
  - Math Programming
  - Code Summarization
  - Code Completion

- **Understanding Tasks**
  - Text-To-Code Retrieval
  - Code Defect Detection
  - Code Clone Detection

- **Retrieval-augmented Generation**

- **Encoder-only LM**

- **Decoder-only LM**

- **Encoder-Decoder LM**

- **Span Denoising**

- **Causal LM**

- **Contrastive Loss**

- **Matching Loss**
CodeT5+: Has two pre-training stages

Unimodal Code Data
1st pretraining
Encoder-Decoder LM

Bimodal Code-Text Data
2nd pretraining
Encoder-Decoder LM

Encoder-only LM
Decoder-only LM
Encoder-Decoder LM

Span Denoising
Causal LM
Contrastive Loss
Matching Loss

Generation Tasks
Text-To-Code Generation,
Math Programming,
Code Summarization,
Code Completion

Understanding Tasks
Text-To-Code Retrieval,
Code Defect Detection,
Code Clone Detection

Retrieval-augmented
Generation

zero-shot/
finetuning/instruction-
tuning
Goal: Train model to recover code contexts at different scales

Data: Code from GitHub

Tasks:

- Span Denoising (15% masked tokens)
- Causal LM
  - Partial programs
  - Complete programs
Stage 2: Code and text pre-training

Goal: Train model for cross-modal understanding and generation

Data: CodeSearchNet (Docstring & Code)

Tasks:
- Contrastive Learning (align feature space of code and text representation)
- Text-Code Matching (predict if semantics match)
- Text-Code Causal LM (text-to-code and code-to-text generation)
**Code T5+: Architecture**

Encoder: encodes contextual representations from either complete, partial or span-masked code/text sequences.

Decoder: generates different outputs based on pretraining task.

*S1: Stage1 and S2: Stage2*
Code T5+: Compute-Efficient Training

- Shallow encoder and deep decoder, initialized with pretrained weights of a decoder code model (CodeGen, Nijkamp et al. 2023)

- Only encoder and cross attention layers are trainable

- Decoder weights are frozen
## CodeT5+: Results

HumanEval code generation: slightly outperforms the CodeGen models it is initialized with.

<table>
<thead>
<tr>
<th>Model</th>
<th>Model size</th>
<th>pass@1</th>
<th>pass@10</th>
<th>pass@100</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Closed-source models</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Codex</td>
<td>2.5B</td>
<td>21.4</td>
<td>35.4</td>
<td>59.5</td>
</tr>
<tr>
<td>Codex</td>
<td>12B</td>
<td>28.8</td>
<td>46.8</td>
<td>72.3</td>
</tr>
<tr>
<td>code-cushman-001</td>
<td>-</td>
<td>33.5</td>
<td>54.3</td>
<td>77.4</td>
</tr>
<tr>
<td>code-davinci-002</td>
<td>-</td>
<td>47.0</td>
<td>74.9</td>
<td>92.1</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>-</td>
<td>48.1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Open-source models</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CodeGen-mono</td>
<td>2B</td>
<td>23.7</td>
<td>36.6</td>
<td>57.0</td>
</tr>
<tr>
<td>CodeGen-mono</td>
<td>6B</td>
<td>26.1</td>
<td>42.3</td>
<td>65.8</td>
</tr>
<tr>
<td>CodeGen-mono</td>
<td>16B</td>
<td>29.3</td>
<td>49.9</td>
<td>75.0</td>
</tr>
<tr>
<td>CodeT5+</td>
<td>220M</td>
<td>12.0</td>
<td>20.7</td>
<td>31.6</td>
</tr>
<tr>
<td>CodeT5+</td>
<td>770M</td>
<td>15.5</td>
<td>27.2</td>
<td>42.7</td>
</tr>
<tr>
<td>CodeT5+</td>
<td>2B</td>
<td>24.2</td>
<td>38.2</td>
<td>57.8</td>
</tr>
<tr>
<td>CodeT5+</td>
<td>6B</td>
<td>28.0</td>
<td>47.2</td>
<td>69.8</td>
</tr>
<tr>
<td>CodeT5+</td>
<td>16B</td>
<td>30.9</td>
<td>51.6</td>
<td>76.7</td>
</tr>
</tbody>
</table>
Code retrieval: outperforms CodeT5 and CodeBERT

Table 6: **Text-to-Code Retrieval results (MRR) on CodeXGLUE**: CodeT5+ achieves consistent performance gains over the original CodeT5 models across all 3 retrieval benchmarks in 7 programming languages. Overall, our models demonstrate remarkable performance, outperforming many strong encoder-based models pretrained with contrastive loss such as SYNCOBERT and UniXcoder.

<table>
<thead>
<tr>
<th>Model</th>
<th>Ruby</th>
<th>JS</th>
<th>Go</th>
<th>Python</th>
<th>Java</th>
<th>PHP</th>
<th>Overall</th>
<th>CosQA</th>
<th>AdvTest</th>
</tr>
</thead>
<tbody>
<tr>
<td>CodeBERT 125M</td>
<td>67.9</td>
<td>62.0</td>
<td>88.2</td>
<td>67.2</td>
<td>67.6</td>
<td>62.8</td>
<td>69.3</td>
<td>65.7</td>
<td>27.2</td>
</tr>
<tr>
<td>GraphCodeBERT 125M</td>
<td>70.3</td>
<td>64.4</td>
<td>89.7</td>
<td>69.2</td>
<td>69.1</td>
<td>64.9</td>
<td>71.3</td>
<td>68.4</td>
<td>35.2</td>
</tr>
<tr>
<td>SYNCOBERT 125M</td>
<td>72.2</td>
<td>67.7</td>
<td>91.3</td>
<td>72.4</td>
<td>72.3</td>
<td>67.8</td>
<td>74.0</td>
<td>-</td>
<td>38.3</td>
</tr>
<tr>
<td>UniXcoder 125M</td>
<td>74.0</td>
<td>68.4</td>
<td>91.5</td>
<td>72.0</td>
<td>72.6</td>
<td>67.6</td>
<td>74.4</td>
<td>70.1</td>
<td>41.3</td>
</tr>
<tr>
<td>CodeGen-multi 350M</td>
<td>66.0</td>
<td>62.2</td>
<td>90.0</td>
<td>68.6</td>
<td>70.1</td>
<td>63.9</td>
<td>70.1</td>
<td>64.8</td>
<td>34.8</td>
</tr>
<tr>
<td>PLBART 140M</td>
<td>67.5</td>
<td>61.6</td>
<td>88.7</td>
<td>66.3</td>
<td>66.3</td>
<td>61.1</td>
<td>68.6</td>
<td>65.0</td>
<td>34.7</td>
</tr>
<tr>
<td>CodeT5 220M</td>
<td>71.9</td>
<td>65.5</td>
<td>88.8</td>
<td>69.8</td>
<td>68.6</td>
<td>64.5</td>
<td>71.5</td>
<td>67.8</td>
<td>39.3</td>
</tr>
<tr>
<td>CodeT5+ 220M</td>
<td>77.7</td>
<td>70.8</td>
<td>92.4</td>
<td>75.6</td>
<td>76.1</td>
<td>69.8</td>
<td>77.1</td>
<td>72.7</td>
<td>43.3</td>
</tr>
<tr>
<td>CodeT5+ 770M</td>
<td><strong>78.0</strong></td>
<td><strong>71.3</strong></td>
<td><strong>92.7</strong></td>
<td><strong>75.8</strong></td>
<td><strong>76.2</strong></td>
<td><strong>70.1</strong></td>
<td><strong>77.4</strong></td>
<td><strong>74.0</strong></td>
<td><strong>44.7</strong></td>
</tr>
</tbody>
</table>