Pre-trained Models and Benchmark for Code Intelligence
(from the perspective of an NLP researcher)

Nan Duan (段楠)
Principal Researcher & Research Manager
Natural Language Computing Group
Microsoft Research Asia

Keynote at KDD workshop on Programming Language Processing (PLP), 2021-08-15
I’d like to thank my colleagues/students that contributed to this code intelligence project, including Duyu Tang (Tencent), Shuai Lu (Microsoft), Daya Guo (Sun Yat-sen University), Junjie Huang (Beihang University), Shuo Ren (MSRA), Zhangyin Feng (Harbin Institute of Technology), Long Zhou (Microsoft), Shujie Liu (Microsoft), Ambrosio Blanco (Microsoft), Ming Zhou (Sinovation), Lidong Zhou (Microsoft), Ming Gong (Microsoft), Linjun Shou (Microsoft), Daxin Jiang (Microsoft), Alexey Svyatkovskiy (Microsoft), Shengyu Fu (Microsoft), Colin Clement (Microsoft), Shao Kun Deng (Microsoft), Dawn Drain (Microsoft), Michele Tufano (Microsoft), Neel Sundaresan (Microsoft), Marc Brockschmidt (Microsoft), Miltiadis Allamanis (Microsoft).
Today’s Agenda

• Background

• Pre-trained Models for Code Intelligence

• Benchmark for Code Intelligence

• Conclusion & Future Work
Current NLP Paradigm: Large-scale Pre-trained Models

BERT (Devlin et al., 2018)

XLM (Lample and Conneau, 2019)

Unicoder-VL (Li et al., 2020)

GPT-3 (Brown et al., 2020)

CLIP (Radford et al., 2021)
Self-supervised Learning in Language Pre-training

**Auto-regressive Decoding**
- GPT-3 (Brown et al., 2020)
  - maximize the likelihood under the forward auto-regressive factorization

**Denoising Auto-encoding**
- BERT (Devlin et al., 2018)
  - reconstruct masked words/spans/sentences from corrupted inputs

**Contrastive Learning**
- SimCSE (Gao et al., 2021)
  - learn representations such that similar samples stay close to each other
Large-scale Pre-training for Code Intelligence

To build large-scale pre-trained models for code to help developers to improve their programming productivity.
Today’s Agenda

• Background

• Pre-trained Models for Code Intelligence

• Benchmark for Code Intelligence

• Conclusion & Future Work
# Three Pre-trained Models for Code Understanding

<table>
<thead>
<tr>
<th>Understanding</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input tokens</strong></td>
<td><strong>previous code tokens</strong></td>
</tr>
<tr>
<td>CodeBERT &amp; GraphCodeBERT</td>
<td>GPT-C</td>
</tr>
<tr>
<td>FFNN + Softmax</td>
<td>Output Code Sketch with Holes</td>
</tr>
</tbody>
</table>

- **CodeBERT & GraphCodeBERT**
  - Input tokens: [CLS] text/code [SEP] code [SEP]
  - FFNN + Softmax
  - Category distribution

- **GPT-C**
  - Previous code tokens

- **Grammformer**
  - Input Code
  - Grammformer Encoder
  - Grammformer Decoder
  - Output Code Sketch with Holes

---

**Accepted by EMNLP 2020 and ICLR 2021.**

**Submitted to NeurIPS 2021.**

**Submitted to EMNLP 2021.**
CodeBERT: Pre-Train with Code

Input

```
def max(a, b):
    x = 0
    if b > a:
        x = b
    else:
        x = a
    return x
```

Output

```
[CLS] def max(a, b): x=0 [MASK] b>a: x=b else x= [MASK] return x [SEP]
```

Randomly mask 15% of tokens

Predict the masked code token with the output of CodeBERT

Transformer

```
def max(a, b):
    x = 0
    if b > a:
        x = b
    else:
        x = a
    return x
```

Source code

```
def max(a, b):
    x = 0
    if b > a:
        x = b
    else:
        x = a
    return x
```
CodeBERT: Pre-Train with Code+Text

Predict the masked code/text tokens with the output of CodeBERT

Source code

```python
def max(a, b):
x=0
if b>a:
x=b
else:
x=a
return x
```

Comment

Return maximum value

def max(a, b):
    x=0
    if b>a:
        x=b
    else:
        x=a
    return x

TreeSitter
(public tool)
graph code bert: Pre-Train with Code + Text + Structure

Source code

def max(a, b):
  x=0
  if b>a:
    x=b
  else:
    x=a
  return x

Comment
Return maximum value

Variable relationship

GraphCodeBERT: Pre-Train with Code + Text + Structure

Source code

def max(a, b):
  x=0
  if b>a:
    x=b
  else:
    x=a
  return x

Comment
Return maximum value

Variable relationship

GraphCodeBERT: Pre-Train with Code + Text + Structure

Source code

def max(a, b):
  x=0
  if b>a:
    x=b
  else:
    x=a
  return x

Comment
Return maximum value

Variable relationship

GraphCodeBERT: Pre-Train with Code + Text + Structure

Source code

def max(a, b):
  x=0
  if b>a:
    x=b
  else:
    x=a
  return x

Comment
Return maximum value

Variable relationship

GraphCodeBERT: Pre-Train with Code + Text + Structure
## Understanding Results

Results on code search.

<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>BiRNN</td>
<td>0.213</td>
<td>0.193</td>
<td>0.688</td>
<td>0.290</td>
<td>0.304</td>
<td>0.338</td>
<td>0.338</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>0.587</td>
<td>0.517</td>
<td>0.850</td>
<td>0.587</td>
<td>0.599</td>
<td>0.560</td>
<td>0.617</td>
</tr>
<tr>
<td>RoBERTa (code)</td>
<td>0.628</td>
<td>0.562</td>
<td>0.859</td>
<td>0.610</td>
<td>0.620</td>
<td>0.579</td>
<td>0.643</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>0.679</td>
<td>0.620</td>
<td>0.882</td>
<td>0.672</td>
<td>0.676</td>
<td>0.628</td>
<td>0.693</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td><strong>0.703</strong></td>
<td><strong>0.644</strong></td>
<td><strong>0.897</strong></td>
<td><strong>0.692</strong></td>
<td><strong>0.691</strong></td>
<td><strong>0.649</strong></td>
<td><strong>0.713</strong></td>
</tr>
</tbody>
</table>

**Input:**

`def read_text_file(filename, encoding="utf-8");`

```
...  
Reads a file under python3 with encoding (default UTF-8). 
Also works under python2, without encoding. 
Uses the EAFP (https://docs.python.org/2/glossary.html#term-eafp) principle.
... 
try: 
    with open(filename, 'r', encoding) as f: 
        r = f.read() 
except TypeError: 
    with open(filename, 'r') as f: 
        r = f.read() 
return r
```

**Output:**

*code source from [GitHub](https://github.com/microsoft/CodeBERT)*
## Generation Results

### Results on code translation.

<table>
<thead>
<tr>
<th>Model</th>
<th>Java→C#</th>
<th>C#→Java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BLEU</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Naïve</td>
<td>18.54</td>
<td>0.0</td>
</tr>
<tr>
<td>PBSMT</td>
<td>43.53</td>
<td>12.5</td>
</tr>
<tr>
<td>RoBERTa (code)</td>
<td>77.46</td>
<td>56.1</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>79.92</td>
<td>59.0</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td><strong>80.58</strong></td>
<td><strong>59.4</strong></td>
</tr>
</tbody>
</table>

### Results on code refinement.

<table>
<thead>
<tr>
<th>Model</th>
<th>Small</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BLEU</td>
<td>Accuracy</td>
</tr>
<tr>
<td>Naïve</td>
<td>78.06</td>
<td>0.0</td>
</tr>
<tr>
<td>LSTM</td>
<td>76.76</td>
<td>10.0</td>
</tr>
<tr>
<td>RoBERTa (code)</td>
<td>77.30</td>
<td>15.9</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>77.42</td>
<td>16.4</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td><strong>80.02</strong></td>
<td><strong>17.3</strong></td>
</tr>
</tbody>
</table>

*Results on code translation.*

*Results on code refinement.*
Three Pre-trained Models for Code

<table>
<thead>
<tr>
<th>Understanding</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input tokens</td>
<td></td>
</tr>
<tr>
<td>[CLS] token, text/code [SEP] code [SEP]</td>
<td></td>
</tr>
<tr>
<td>CodeBERT &amp; GraphCodeBERT</td>
<td></td>
</tr>
<tr>
<td>FFNN + Softmax</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>category distribution</td>
<td></td>
</tr>
</tbody>
</table>

Submitted to EMNLP 2021.

previous code tokens

GPT-C

next code tokens

Grammformer Encoder

Grammformer Decoder

Input Code

Output Code Sketch with Holes

Submitted to NeurIPS 2021.

Accepted by EMNLP 2020 and ICLR 2021.
GPT-C: Multilingual Pre-trained Model

Predict next code token given context of previous tokens

```
def max(a, b):
    x = 0
    if b > a:
        x = b
    else:
        x = a
    return x
```

Trained for 10 PLs: JavaScript, C, Java, Go, PHP, Python, C++, C#, Ruby, TypeScript
Evaluation on Code Completion

• Corpus
  • 10 programming languages
  • 354K code projects
  • 18B code lines
  • Split 8:1:1 as Train/Dev/Test

<table>
<thead>
<tr>
<th>Programming language</th>
<th>#Projects</th>
<th>#Files (l)</th>
<th>#Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>JavaScript</td>
<td>113,890</td>
<td>15,330,706</td>
<td>4,226,121,235</td>
</tr>
<tr>
<td>C</td>
<td>19,900</td>
<td>13,462,890</td>
<td>7,253,471,852</td>
</tr>
<tr>
<td>Java</td>
<td>46,921</td>
<td>10,385,540</td>
<td>1,491,132,997</td>
</tr>
<tr>
<td>Go</td>
<td>17,922</td>
<td>5,720,219</td>
<td>1,997,845,604</td>
</tr>
<tr>
<td>PHP</td>
<td>24,625</td>
<td>4,691,140</td>
<td>653,891,761</td>
</tr>
<tr>
<td>Python</td>
<td>71,343</td>
<td>4,465,808</td>
<td>854,503,198</td>
</tr>
<tr>
<td>C++</td>
<td>20,958</td>
<td>4,293,413</td>
<td>1,400,309,370</td>
</tr>
<tr>
<td>C#</td>
<td>17,387</td>
<td>3,765,835</td>
<td>550,267,681</td>
</tr>
<tr>
<td>Ruby</td>
<td>17,804</td>
<td>1,663,262</td>
<td>137,558,948</td>
</tr>
<tr>
<td>TypeScript</td>
<td>3,801</td>
<td>466,924</td>
<td>64,671,728</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Test language</th>
<th>ROUGE-L</th>
<th>Average edit similarity (Levenshtein, %)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Precision</td>
<td>Recall</td>
</tr>
<tr>
<td>GPT-C (12L)</td>
<td>Python</td>
<td>0.72</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>C#</td>
<td>0.56</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>JavaScript</td>
<td>0.71</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>Go</td>
<td>0.65</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>Scala (zero-shot)</td>
<td>0.41</td>
<td>0.54</td>
</tr>
</tbody>
</table>
Extended Context for Code Completion

1. Take the concrete syntax tree of the source file;

2. Prioritize the syntactic elements;
   i. signature and docstring of the focal method;
   ii. global import statements and assigned values;
   iii. class attributes, peer class method signatures, class docstring, peer class method docstrings;
   iv. global expressions and code bodies of peer class methods.

3. Take elements based on their priorities until the context window has been filled.

We reserve 3/4 (768/1,024) tokens for the extended context and 1/4 (256/1024) tokens for the local context.
GPT-C with Extended Context

Predict next code token given normal context and extended context

import torch
CTX_LEN = 512
class EC_Model(...)
def forward(...)
self.outputs = self.

import statement
global variables
class signature
function signature

normal context

extended context

Output
Transformer
Input

encoder nn ...
: ...
a

distribution over vocab

L1
L2
L12

FFNN + Softmax

15% 0.1% 0.4% 0.1%
Evaluation on Code Completion

- **Pre-training data**
  - Python dataset used in multilingual GPT-C pre-training

- **Evaluation data**
  - PY150 test set in CodeXGLUE

<table>
<thead>
<tr>
<th>Model</th>
<th>Test language</th>
<th>ROUGE-L</th>
<th>Average edit similarity (Levenshtein, %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-C (12L)</td>
<td>Python</td>
<td>Precision 0.81</td>
<td>Recall 0.94 F1 0.87</td>
</tr>
<tr>
<td>GPT-C with Extended Context (12L)</td>
<td>Python</td>
<td><strong>0.90</strong></td>
<td><strong>0.96</strong> F1 <strong>0.93</strong></td>
</tr>
</tbody>
</table>
# Three Pre-trained Models for Code Understanding

<table>
<thead>
<tr>
<th>Understanding</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input tokens</strong></td>
<td><strong>previous code tokens</strong></td>
</tr>
<tr>
<td>[CLS], text/code, [SEP], code, [SEP]</td>
<td>GPT-C, next code tokens</td>
</tr>
<tr>
<td>CodeBERT &amp; GraphCodeBERT</td>
<td></td>
</tr>
<tr>
<td>FFNN + Softmax</td>
<td></td>
</tr>
<tr>
<td>0, category distribution</td>
<td></td>
</tr>
</tbody>
</table>

### Accepted by EMNLP 2020 and ICLR 2021.

### Submitted to EMNLP 2021.

### Submitted to NeurIPS 2021.
Challenge in Code Completion: Ambiguities

```python
1 import argparse
2 ap = argparse.ArgumentParser()
3 ap.add_argument("--release",
4             action="store_true")
5 ap.add_argument("--prerelease",
6             action="store_true")
7 ```
def sum_to_n ( n : int) :
    # Sums the numbers from 1 to n-1
    return $Name ( $Call )

def sum_to_n ( n : int) :
    # Sums the numbers from 1 to n-1
    return $Name ( $Call )

Choose one to expand

Policy Network

Grammformer Encoder

Encoded Representations

Grammformer Decoder

Grammformer Inference
Grammformer Training

Algorithm 1 GRAMMMFORMER generative process, given an input sequence $x_0$.

\[
\text{for } t = 0, 1, 2, \ldots \text{ do} \\
\quad i_t \sim P_s(i \mid x_t, N(x_t)) \quad \triangleright \text{Sample non-terminal position from } N(x_t) \text{ to expand} \\
\quad \text{if } i_t = \otimes \text{ then} \quad \triangleright \text{if } x_t \text{ does not contain non-terminals or none was selected by } P_s \\
\quad \quad \text{break} \quad \triangleright \text{Stop generation} \\
\quad \hat{y}_{t \otimes i_t} \sim P_e(y \mid x_t, i_t) \quad \triangleright \text{Sample expansion of non-terminal at position } i_t \\
\quad x_{t+1} \leftarrow x_{t, <i_t} :: \hat{y}_{t \otimes i_t} :: x_{t, >i_t} \quad \triangleright \text{Create } x_{t+1} \text{ by expanding non-terminal at } i_t \text{ to } \hat{y}_{t \otimes i_t} \\
\quad x_{\text{out}} \leftarrow \text{NONTERMINALSTOHOLES}(x_t) \quad \triangleright \text{Convert any remaining non-terminals to holes} \\
\text{return } x_{\text{out}}
\]

\[
\mathcal{L}_{\text{train}}(x_0, x^*) = \left( r(x_{\text{out}}, x^*) - \tilde{r}(x_0) \right) \sum_{t=0}^{T} \left( -\log P_s(i_t \mid x_t) - \mathbb{I}(i_t \neq \otimes) \log P_e(\hat{y}_{t \otimes i_t} \mid x_t, i_t) \right)
\]
REGEXACC as Reward Function

\[
\text{REGEXACC}(\hat{s}, s^*) \triangleq \frac{n\text{Term}(\hat{s})}{n\text{Term}(s^*)} \cdot \text{matches}(\text{toRegex}(\hat{s}), s^*)
\]

- return the number of terminal symbols
- turn a predicted sketch into a regular expression by replacing all non-terminals with the wildcard matching any non-empty sequence
- return 1 if the regex matches the ground truth, and 0 otherwise
## Experiment

### Dataset

More than 20 stars in GitHub that contain C# and Python Code. Including 3.8M and 4.5 M files for C# and Python, respectively.

### Results

<table>
<thead>
<tr>
<th></th>
<th>C#</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>REGEXACC</td>
<td>ROUGE</td>
</tr>
<tr>
<td></td>
<td>Top 1</td>
<td>Top 5</td>
</tr>
<tr>
<td>$L \rightarrow R$</td>
<td>0.42</td>
<td>0.47</td>
</tr>
<tr>
<td>$L \rightarrow R + \circ$</td>
<td>0.45</td>
<td>0.54</td>
</tr>
<tr>
<td>Grammformer</td>
<td><strong>0.47</strong></td>
<td><strong>0.59</strong></td>
</tr>
</tbody>
</table>
Today’s Agenda

• Background

• Pre-trained Models for Code Intelligence

• Benchmark for Code Intelligence

• Conclusion & Future Work
## CodeXGLUE: 14 datasets for 10 Code-related tasks

<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>
<th>Dataset Provider</th>
<th>Task definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code-Code</strong></td>
<td>Clone Detection</td>
<td>BigCloneBench</td>
<td>Java</td>
<td>900K/416K/416K</td>
<td>CodeBERT</td>
<td>Univ. of Saskatchewan</td>
<td>Predict semantic equivalence for a pair of codes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>POJ-104</td>
<td>C/C++</td>
<td>32K/8K/12K</td>
<td></td>
<td>Peking Univ</td>
<td>Retrieve semantically similar codes.</td>
</tr>
<tr>
<td></td>
<td>Defect Detection</td>
<td>Defects4J</td>
<td>C</td>
<td>21k/2.7k/2.7k</td>
<td></td>
<td>Univ. of Washington</td>
<td>Identify whether a function is vulnerable.</td>
</tr>
<tr>
<td></td>
<td>Cloze Testing</td>
<td>CT-all</td>
<td>Python, Java, PHP,</td>
<td>-/-/176k</td>
<td></td>
<td>CodeSearchNet</td>
<td>Tokens to be predicted come from the entire vocab.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CT-max/min</td>
<td>JavaScript, Ruby, Go</td>
<td>-/-/2.6k</td>
<td></td>
<td>Created by MSRA based on CodeSearchNet</td>
<td>Tokens to be predicted come from (max, min).</td>
</tr>
<tr>
<td></td>
<td>Code Completion</td>
<td>PY150</td>
<td>Python</td>
<td>100k/5k/50k</td>
<td>CodeGPT</td>
<td>ETH Zurich, line-level data added by MSRA</td>
<td>Predict following tokens given contexts of codes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GitHub Java Corpus</td>
<td>Java</td>
<td>13k/7k/8k</td>
<td></td>
<td>Univ. of Edinburgh, line-level data added by MSRA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code Refinement</td>
<td>Bugs2Fix</td>
<td>Java</td>
<td>98K/12K/12K</td>
<td>Encoder-Decoder</td>
<td>The College of William and Mary</td>
<td>Automatically refine codes by fixing bugs.</td>
</tr>
<tr>
<td></td>
<td>Code Translation</td>
<td>CodeTrans</td>
<td>Java-C#</td>
<td>10K/0.5K/1K</td>
<td></td>
<td>MSRA</td>
<td>Translate the codes from one programming language to another programming language.</td>
</tr>
<tr>
<td><strong>Text-Code</strong></td>
<td>NL Code Search</td>
<td>CodeSearchNet, AdvTest</td>
<td>Python</td>
<td>251K/9.6K/19K</td>
<td>CodeBERT</td>
<td>GitHub + MSR Cambridge, test provided by MSRA</td>
<td>Given a natural language query as input, find semantically similar codes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>StacQC, WebQueryTest</td>
<td>Python</td>
<td>2.9k/0.9k/1.9k</td>
<td></td>
<td>The Ohio State Univ, test provided by MSRA</td>
<td>Given a pair of natural language and code, predict whether they are relevant or not.</td>
</tr>
<tr>
<td></td>
<td>Text-to-Code Generation</td>
<td>CONCODE</td>
<td>Java</td>
<td>100K/2K/2K</td>
<td>CodeGPT</td>
<td>Univ. of Washington</td>
<td>Given a natural language docstring/comment as input, generate a code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>JavaScript, Ruby, Go</td>
<td></td>
<td></td>
<td>MSRA</td>
<td></td>
</tr>
<tr>
<td><strong>Text-Text</strong></td>
<td>Documentation Translation</td>
<td>Microsoft Docs</td>
<td>English-Latvian/Danish/Norwegian/Chinese</td>
<td>156K/4K/4K</td>
<td></td>
<td>MSRA</td>
<td>Translate code documentation between human languages (e.g. En-Zh), intended to test low-resource multi-lingual translation.</td>
</tr>
</tbody>
</table>
CodeXGLUE: 14 datasets for 10 Code-related tasks

According to Evans Data Corporation, there are 23.8 million professional developers in 2019, and the population is expected to reach 26.7 million in 2024. With the growing

https://github.com/microsoft/CodeXGLUE

https://microsoft.github.io/CodeXGLUE/

accepted by NeurIPS 2021
## Current Submission Status

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th># of Submissions</th>
<th>Organizations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Code-Code</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clone Detection</td>
<td>BigCloneBench</td>
<td>2</td>
<td>UCLA &amp; Columbia University; HNUST;</td>
</tr>
<tr>
<td></td>
<td>POJ-104</td>
<td>2</td>
<td>UCLA &amp; Columbia University;</td>
</tr>
<tr>
<td>Defect Detection</td>
<td>Defects4J</td>
<td>4</td>
<td>IBM Research; Case Western Reserve University; UCLA &amp; Columbia University;</td>
</tr>
<tr>
<td>Cloze Test</td>
<td>CT-all</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CT-max/min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Completion</td>
<td>GitHub Java Corpus</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PY150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Repair</td>
<td>Bugs2Fix</td>
<td>2</td>
<td>Case Western Reserve University; UCLA &amp; Columbia University;</td>
</tr>
<tr>
<td>Code Translation</td>
<td>CodeTrans</td>
<td>1</td>
<td>UCLA &amp; Columbia University;</td>
</tr>
<tr>
<td><strong>Text-Code</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NL Code Search</td>
<td>AdvTest</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WebQueryTest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text-to-code</td>
<td>CONCODE</td>
<td>5</td>
<td>Wuhan University; Case Western Reserve University; UCLA &amp; Columbia University;</td>
</tr>
<tr>
<td>Generation</td>
<td></td>
<td></td>
<td>UBC Research;</td>
</tr>
<tr>
<td><strong>Code-Text</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code Summarization</td>
<td>CodeSearchNet</td>
<td>4</td>
<td>USTC &amp; MSRA; Case Western Reserve University; UCLA &amp; Columbia University;</td>
</tr>
<tr>
<td>Documentation</td>
<td>Microsoft Docs</td>
<td></td>
<td>UC Davis;</td>
</tr>
<tr>
<td><strong>Text-Text</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Translation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>20 (in total)</strong></td>
</tr>
</tbody>
</table>
Today’s Agenda

• Background

• Pre-trained Models for Code Intelligence

• Benchmark for Code Intelligence

• Conclusion & Future Work
Conclusion & Future Work

• Conclusion
  • Large-scale self-supervised pre-training can be successfully adapted to code scenarios.
  • Considering the uniqueness of code can lead to better pre-trained models for code.
  • CodeXGLUE is a useful benchmark and model evaluation platform for code intelligence.

• Future work
  • Pre-trained models with more code-related knowledge (syntax, structure, etc.) integrated
  • Pre-trained models to deal with downstream tasks with very long code inputs/outputs
  • Efficient training and inference for code completion/generation tasks
  • CodeXGLUE++ with more programming languages and more tasks
  • Serious consideration on privacy and intellectual property when using open-source codes in pre-training and downstream applications (e.g., OpenAI’s Copilot 😊)
Thank You
谢谢