Any-Code Completion

```java
public static Path[] stat2Paths(FileStatus[] stats) {
    if (stats == null)
        return null;
    Path[] ret = new Path[stats.length];
    for (int i = 0; i < stats.length; ++i){
        ret[i] = stats[i].getPath();
    }
    return ret;
}
```

Generated: (Java)

- `stats[i].getPath()` (25.2%)
- `new Path(stats[i])` (3.3%)
- `new Path(stats[i], charset)` (2.5%)
Overview: a **Structural** Language Model

```java
stats[i].getPath()
```
public static Path[] stat2Paths(FileStatus[] stats) {
    if (null) return null;
    Path[] ret = new Path[stats.length];
    for (int i = 0; i < stats.length; ++i) {
        ret[i] = stats[i].getPath();
    }
    return ret;
}
Structural Language Models of Code
ICML’2020

Uri Alon  
Technion

Roy Sadaka  
Technion

Omer Levy  
Tel-Aviv University  
Facebook AI Research

Eran Yahav  
Technion
Language modeling of code

- Code completion
- Validate existing code, detect unlikely code.
Key Idea #1: predict a missing subtree

Instead of representing the task as:
“predict a missing sentence in a text”

Represent the task as:
“predict a missing subtree in a tree”.

Learn syntactic patterns, instead of sequential patterns
Abstract Syntax Tree

Any valid code snippet can be parsed into an Abstract Syntax Tree (AST). The AST is composed of nodes and user-defined values in its leaves.

```
stats[i].getPath()
```
Key Idea #2: a structural language model (SLM)

In a natural-language model:

\[ Pr(Y) = Pr(y_1, y_2, \ldots, y_n) = \prod_{t=1}^{n} Pr \left( y_t \mid y < t \right) \]

But how can we compute the probability of a tree?
Key Idea #2: a structural language model (SLM)

Given a tree $\mathcal{A}$ (can be an arbitrary graph)

Induce an ordering over its nodes: $a_0, a_1, \ldots, a_n \in \mathcal{A}$ (in practice: DFS)

A structural language model (SLM) computes the probability of the tree $\mathcal{A}$:

$$Pr(\mathcal{A}) = \prod_{t=0}^{n} Pr\left(a_t \mid a_{<t}\right)$$

But, how can we represent the partial tree $a_{<t}$ when computing $Pr\left(a_t \mid a_{<t}\right)$?
The fundamental tradeoff in code representation

Learning Effort
- Implicitly re-learn syntactic & semantic regularities
- Model size, data, time...
- Sweet-spot
- Requires expertise, language-specific, task-specific model

Analysis Effort

- Surface text (token stream)
- AST Paths
- Handcrafted features
- Data flow Analysis
- Control flow Analysis
- ... [“A General Path-based Representation …”, PLDI’2018]
- [“code2vec”, POPL’2019]
Key Idea #3: a partial tree as AST paths

We compute the probability of a node \( Pr(a_t \mid a_{<t}) \) by considering the paths in the Abstract Syntax Tree (AST) from all leaves into \( a_t \).
AST Paths

AST Paths are simple paths over nodes in the AST.
In previous works, we used AST paths to read code.
In this work, we generate code by predicting the next node in a set of AST paths.

[“code2seq”, ICLR’2019]
public static Path[] stat2Paths(FileStatus[] stats) {
    if (stats == null)
        return null;
    Path[] ret = new Path[stats.length];
    for (int i = 0; i < stats.length; ++i) {
        ret[i] = stats[i].getPath();
    }
    return ret;
}
Model

• Any sequential encoder to encode each arbitrary-length path into a fixed-length vector separately (e.g., LSTM, transformer encoder)
• Any contextualizer to let all paths interact (e.g., transformer encoder)
• Attend to the contextualized paths using the root path as the query
Model

- Encode paths
- Contextualize
- Attend
- Predict node

Greater
Generate the Tree of: $x > 1$
Generate the Tree of: $x > 1$
Generate the Tree of: $x > 1$
Generate the Tree of: $x > 1$
Generate the Tree of: \( x > 1 \)
Generate the Tree of: $x > 1$
Copy Mechanism

```java
myNewFoo = myObj.getFoo();
myNewFoo.setFooId(id);
```
public static Path[] stat2Paths(FileStatus[] stats) {
    if (stats == null)
        return null;
    Path[] ret = new Path[stats.length];
    for (int i = 0; i < stats.length; ++i){
        ret[i] = stats[i].getPath();
    }
    return ret;
}

Generated: (Java)

    stats[i].getPath()  (25.2%)
    new Path(stats[i])  (3.3%)
    new Path(stats[i], charset)  (2.5%)
Example - C#

```csharp
public static string Camelize(this string input)
{
    var word = input.Pascalize();
    return word.Length > 0 ? word.Substring(0, 1).ToLower() + word.Substring(1) : word;
}
```

Generated: (C#)

- `word.Substring(0, 1)` (14.1%)
- `word.trim()` (8.2%)
- `word.Substring(1)` (5.8%)
Java Results (trained on 1.3M examples)

- seq2prod
- seq2tree
- LSTMs+attn+copy
- Transformer-small+copy
- Transformer-base+copy
- SLM

<table>
<thead>
<tr>
<th>Model</th>
<th>acc@1</th>
<th>acc@5</th>
<th>tree@1</th>
<th>tree@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq2prod</td>
<td>8.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seq2tree</td>
<td></td>
<td>16.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTMs+attn+copy</td>
<td>16.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transformer-small+copy</td>
<td>18.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transformer-base+copy</td>
<td>11.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLM (this work)</td>
<td>23.0</td>
<td>23.2</td>
<td>21.4</td>
<td>24.8</td>
</tr>
<tr>
<td>SLM</td>
<td>24.1</td>
<td>24.8</td>
<td>14.2</td>
<td>18.0</td>
</tr>
</tbody>
</table>

SLM: 1.4, 3.8, 4.4, 13.6, 7.9, 5.6
SLM (this work): 3.8, 38.1, 4.8, 31.8, 34.7, 39.1

a.b > 1

NAME.NAME > INT
## C# Results

<table>
<thead>
<tr>
<th>Method</th>
<th>acc@1</th>
<th>acc@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHOG</td>
<td>30.2</td>
<td>24.6</td>
</tr>
<tr>
<td>GNN → NAG</td>
<td>22.4</td>
<td>11.2</td>
</tr>
<tr>
<td>seq2seq +copy</td>
<td>26.4</td>
<td>15.3</td>
</tr>
<tr>
<td>seq2tree +copy</td>
<td>22.3</td>
<td></td>
</tr>
<tr>
<td>SLM (this work)</td>
<td>37.6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>acc@1</th>
<th>acc@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHOG</td>
<td>12.0</td>
<td>18.5</td>
</tr>
<tr>
<td>GNN → NAG</td>
<td></td>
<td>27.1</td>
</tr>
<tr>
<td>seq2seq +copy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seq2seq +copy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>seq2tree +copy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLM (this work)</td>
<td>37.9</td>
<td>35.9</td>
</tr>
</tbody>
</table>
Error Analysis

What kind of mistakes are responsible for the gap between $\text{acc}@k$ and $\text{tree}@k$?

SLM (this work):
Error Analysis

What kind of mistakes are responsible for the gap between $\text{acc@k}$ and $\text{tree@k}$?

74%: Single-token mismatch

30%: Single-subtoken mismatch

[Diagram showing error analysis with percentages and values]
Error Analysis

```java
public float getProgress() {
    this.readLock.lock();
    try {
        if (this.currentAttempt != null) {
            return this.currentAttempt.getProgress();
        }
        return 0;
    } finally {
        this.readLock.unlock();
    }
}
```

<table>
<thead>
<tr>
<th>Generated:</th>
<th>Exact-match</th>
<th>Tree-match</th>
<th>Compiles</th>
</tr>
</thead>
<tbody>
<tr>
<td>this.currentAttempt.getCount()</td>
<td>(31.3%)</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>-1</td>
<td>(30.6%)</td>
<td>x</td>
<td>✔</td>
</tr>
<tr>
<td>this.currentAttempt.get()</td>
<td>(1.5%)</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>this.currentAttempt.getTime()</td>
<td>(1.2%)</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>this.currentAttempt.getProgress()</td>
<td>(0.9%)</td>
<td>✓</td>
<td>✔</td>
</tr>
</tbody>
</table>
public float getProgress() {
    this.readLock.lock();
    try {
        if (this.currentAttempt != null) {
            return this.currentAttempt.getProgress();
        }
        return 0;
    } finally {
        this.readLock.unlock();
    }
}
```java
public static Path[] stat2Paths(FileStatus[] stats) {
    if (??) return null;
    Path[] ret = new Path[??];
    for (int i = 0; i < stats.length; ++i) {
        ret[i] = ??
    }
    return ret;
}
```
public static Path[] stat2Paths(FileStatus[] stats) {
    if (??) return null;
    Path[] ret = new Path[??];
    for (int i = 0; i < stats.length; ++i) {
        ret[i] = stats[i].getPath();
    }
    return ret;
}
Structural Language Models of Code

Key points:

1. Predicting a missing **subtree** in a **tree**

2. A structural language model over **trees**

3. A partial AST as a set of paths

---

Pr(\(\mathcal{A}\)) = \(\prod_{i=0}^{n} Pr(a_i | a_{<i})\)

http://AnyCodeGen.org

uralon@cs.technion.ac.il