Programming Language Processing
Learning To Edit Code

Saikat Chakraborty
Ph.D. Candidate
Department of Computer Science
Columbia University.
Automatic Code Editing
**Adding Feature**

```java
public String removeComment(String leftOver) {
    -  if (hasBlockComment(leftOver)) {
+   while (hasBlockComment(leftOver)) {
          leftOver = removeBlockComment(leftOver);
        }
    -  if (hasLineComment(leftOver)) {
+   while (hasLineComment(leftOver)) {
          leftOver = removeLineComment(leftOver);
        }
    return leftOver;
}
```

**Refactoring**

```java
void visit(JSession xsession, Timer t) throws Exception {
    if (xsession != null && t.getTime() > xsession.getStartTime()) {
        visit((JNode) xsession, t);
    }
    else {
        visit(new JNode(), new Timer());
    }
```
Code Edits Are Repetitive [Meng et. al. 2011, 2013], [Ray et. al. 2015]

Is it possible to Automate Such Code Edits?
Automating Code Edits - Template/Search based

Edit Template
...
- return super.equals(object);
+ return this == object;
...

Match Found

```java
public Model copy(Model instance){
    ...
    instance.notify();
    if (super.equals(instance) && !instance.isEmpty()){
        return instance.clone();
    }
    ...
}
```
Automating Code Edits - Template/Search based

Edit Template

... 
- return super.equals(object);
+ return this == object;
...

Patch Applied

```
public Model copy(Model instance){
    ...
    instance.notify();
    if (this == instance && !instance.isEmpty()){
        return instance.clone();
    }
    ...
}
```

Problems

- Too many templates to write.
- Bottleneck: Template Matching / Code Search
Code Editing Task - Learning Based Solution [Tufano et. al. 2019]

Example Code Edits

```
- a b c
+ x y z
```

```
return super.equals(object)  
```

Code Before Edit

```
return this == object
```

Code After Edit
Code Editing Task - Learning Based Solution

Encoder and Decoder learns Code Edit Patterns from example edits.

Encoder: return super.equals(object)

Decoder: return this == object
Code Editing Task - Learning Based Solution

Encoder and Decoder learns to Apply the Pattern in Similar Context.
Encoder-Decoder

Encoder encodes the input code to a vector or matrix.

Decoder generates the edited code.

```
return super.equals(object)
```

```
return this == object
```
Code as sequence of tokens [Tufano et. al. 2018, 2019]

Encoder

$\text{return super.equals(object)}$

Decoder

$\text{return this == object}$

Recurrent Network for both Encoder and Decoder
Code as sequence of tokens [Tufano et. al. 2018, 2019]

Problem?

return super.equals(object)

return this == object
Desired Properties of Encoder and Decoder
Properties of Source Code.

```java
return super.equals(object);
```

Code are **Highly Structured.**

Invisible properties. *e.g.* **Non-Terminals**
Properties of Source Code.

1. `public boolean checkEqual(
2.          Object inst, MyClass object){
3.          MyClass tmp = new MyClass();
4.          if (inst == null) {
5.              MyClass tmp2 = new MyClass();
...
20.          }
21.          return super.equals(object);
22.      }
Desired Properties of the Encoder

Source Code

Encoder

Understand Structure

Understand Dependency

Understand Long Range Relations
Desired Properties of the Decoder

1. **Syntactic** correctness.
2. **Contextual** correctness.

```java
boolean f (Object target) {
    for (Object elem : if.elements) {
        if (elem.equals(target)) {
            return true;
        }
    }
    return false;
}
```

```java
boolean f (Object target) {
    for (Object elem : this.elements) {
        if (elem.equals(f)) {
            return null;
        }
    }
    return false;
}
```
1. **Enforcing the Syntactic Correctness**
CODIT: Code Editing With Tree Based Neural Models
TSE 2020

2. **Learning the Contextual Correctness**
Unified Pre-training for Program Understanding and Generation - NAACL 2021

3. **Multi-Modal Code Editing**
Work in progress
CODIT: Code Editing With Tree Based Neural Models

TSE - 2020

Findings

Generation of Syntax Tree instead of code Guarantees Syntactic Correctness.

Contribution

Tree/Grammar Based Model for Automatic Code Editing.
Is it possible to Edit Code with Guaranteed Syntactic Correctness?
Generate Syntactically Correct Code

if (super.equals(instance))

if (this == instance)
return this == object;
CODIT: Code Editing With Tree Based Neural Models

Code Before Edit
\[
\text{return super.equals(object);}
\]

Code After Edit
\[
\text{return this == object;}
\]
CODIT Step 1
Tree Translation
CODIT: Code Editing With Tree Based Neural Models

```java
return super.equals(object);
```

Graph representation:
- **RetStmt**
  - **Return**
    - **return**
    - **Call**
      - **ORef**
        - **super**
      - **DOT**
      - **Method**
      - **LB**
        - **(**
        - **Var**
          - **object**
        - **)**
      - **Param**
      - **RB**

Edges:
- **RetStmt** → **Return**
- **Return** → **Call**
- **Call** → **ORef**
- **ORef** → **super**
- **Call** → **DOT**
- **Call** → **Method**
- **Call** → **LB**
- **Call** → **Param**
- **Call** → **RB**
- **Call** → **(**
- **Call** → **Var**
- **Call** → **)**
- **Call** → **object**
- **Stmt** → **Call SemiColon**
- **Call** → **O_Ref DOT Method LB Param RB**
- **Param** → **Var**
CODIT: Code Editing With Tree Based Neural Models

Rules sequence of Syntax Tree before edit

RetStmt → Return Stmt
Stmt → Call Semicolon
Call → O_Ref DOT Method
       LB Param RB
Param → Var

Rules sequence of Syntax Tree after edit

RetStmt → Return Stmt
Stmt → Bool_St Semicolon
Bool_St → LHS EQ RHS
LHS → VAR
RHS → VAR
CODIT: Code Editing With Tree Based Neural Models

RetStmt → Return Stmt
Stmt → BoolSt Semicolon
BoolSt → LHS EQ RHS
LHS → VAR
RHS → VAR
CODIT: Code Editing With Tree Based Neural Models

RetStmt → Return Stmt
Stmt → Call Semicolon
Call → O_Ref DOT Method
Param → Var

RetStmt → Return Stmt
Stmt → Bool_St Semicolon
Bool_St → LHS EQ RHS
LHS → VAR
RHS → VAR
CODIT: Code Editing With Tree Based Neural Models

Still not a complete code.
CODIT Step 2
Token Generation
CODIT: Code Editing With Tree Based Neural Models

Token Translation

Restricted By Grammar
public boolean checkEqual(
    Object inst, MyClass object) {
    MyClass tmp = new MyClass();
    if (inst == null) {
        MyClass tmp2 = new MyClass();
        ...
    }
    return super.equals(object);
}
CODIT Evaluation
## CODIT: Code Editing With Tree Based Neural Models

### Study Subjects

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Number of Projects</th>
<th>Number of Edit Examples</th>
<th>Code Fragment Size</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of Tokens</td>
<td>Max</td>
<td>Avg</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Number of Nodes</td>
<td>Max</td>
<td>Avg</td>
</tr>
<tr>
<td>Generic Code Edit from Github</td>
<td>48</td>
<td>32,473</td>
<td>Max - 38</td>
<td>Max - 47</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avg - 15</td>
<td>Avg - 20</td>
<td></td>
</tr>
<tr>
<td>Pull Request Edits [Tufano et. al. 2019]</td>
<td>3</td>
<td>5546</td>
<td>Max - 34</td>
<td>Max - 47</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avg - 17</td>
<td>Avg - 23</td>
<td></td>
</tr>
</tbody>
</table>
## Results (Accuracy in top 5 edited code)

<table>
<thead>
<tr>
<th>Method</th>
<th>Generic Code Edits</th>
<th>Pull Request Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequence Based</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSTM-Seq2Seq</td>
<td>3.77%</td>
<td>11.26%</td>
</tr>
<tr>
<td>Tufano et. al.</td>
<td>6.57%</td>
<td>23.65%</td>
</tr>
<tr>
<td>SequenceR</td>
<td>9.76%</td>
<td>26.43%</td>
</tr>
<tr>
<td><strong>Tree Based</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tree2Seq</td>
<td>11.04%</td>
<td>23.49%</td>
</tr>
<tr>
<td>CODIT</td>
<td><strong>15.94%</strong></td>
<td><strong>28.87%</strong></td>
</tr>
</tbody>
</table>
Application - Automatic Program repair

CODIT fixes **15 bugs completely** and **10 bugs partially**, out of 80 bugs in **Defects4j**.
Take Away Point

Generating Syntax Tree Guarantees Syntactic Correctness of Edited Code.
Contextual Correctness: Is it there?
```java
boolean checkEqual (Object target) {
    for (Object elem : this.elements) {
        if (elem.equals(checkEqual)) {
            return true;
        }
    }
    return false;
}

boolean checkEmpty (Object target) {
    if (target == null) {
        return null;
    }
    return false;
}

public void setPredecessor(Message m) {
    ... 
    Message sent = new Message();
    sent.setTo(m.orgin);
    sendMessage(m);
    return sendMessage(m);
}
```
Recurrent Model Vs. Transformer Model

1. Recurrent Model

2. Transformer Model
Is it possible to learn **Contextual Correctness**?
Unified Pre-training for Program Understanding and Generation (PLBART)

Findings
Contextual Correctness can be learned from large scale dataset.

Contribution
Developed large scale pre-trained models for different SE tasks.
Code Before Edit

Input

Language Model

Task Model

Code After Edit

Output
Pretraining - Embed the knowledge of input and output into the model.
BERT - Pretrained Transformer Encoder

Pre-training:
Task agnostic Masked Language Model.

3. GraphCodeBERT - Guo et. al. 2021

Knowledge about generation is not embedded in Decoder.
PLBART - Pretraining both encoder and decoder.

How do we train PLBART?
Unified Pre-training for Program Understanding and Generation

Denoising Auto-Encoding

Bidirectional Encoder

Noise Induced Input

static void for <MASK> ...

Type of noises

Replaced Token

Masked Token

Generative Decoder

Noise Removed Input

static void main { ...

<s> static void main ...

Unified Pre-training for Program Understanding and Generation
PLBART - what is it learning?

**Encoder**
- Learns the input structure implicitly.
- Learns long range dependencies.
- Learns to generate representations based on these features.

**Decoder**
- Learns to generate correct code
- Learn to reason about any missing and misplaced token in input.
PLBART Evaluation
## Downstream Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Dataset</th>
<th>Language</th>
<th>Train</th>
<th>Valid</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summarization</td>
<td>Husain et al. (2019)</td>
<td>Ruby</td>
<td>24,927</td>
<td>1,400</td>
<td>1,261</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Javascript</td>
<td>58,025</td>
<td>3,885</td>
<td>3,291</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Go</td>
<td>167,288</td>
<td>7,325</td>
<td>8,122</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Python</td>
<td>251,820</td>
<td>13,914</td>
<td>14,918</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java</td>
<td>164,923</td>
<td>5,183</td>
<td>10,955</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PHP</td>
<td>241,241</td>
<td>12,982</td>
<td>14,014</td>
</tr>
<tr>
<td>Generation</td>
<td>Iyer et al. (2018)</td>
<td>NL to Java</td>
<td>100,000</td>
<td>2,000</td>
<td>2,000</td>
</tr>
<tr>
<td>Translation</td>
<td>Code-Code (Lu et al., 2021)</td>
<td>Java to C#</td>
<td>10,300</td>
<td>500</td>
<td>1,000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C# to Java</td>
<td>10,300</td>
<td>500</td>
<td>1,000</td>
</tr>
<tr>
<td></td>
<td>Program Repair (Tufano et al., 2019)</td>
<td>Java_{small}</td>
<td>46,680</td>
<td>5,835</td>
<td>5,835</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java_{medium}</td>
<td>52,364</td>
<td>6,545</td>
<td>6,545</td>
</tr>
<tr>
<td>Classification</td>
<td>Vulnerability Detection</td>
<td>C/C++</td>
<td>21,854</td>
<td>2,732</td>
<td>2,732</td>
</tr>
<tr>
<td></td>
<td>(Zhou et al., 2019)</td>
<td>Java</td>
<td>100,000</td>
<td>10,000</td>
<td>415,416</td>
</tr>
<tr>
<td></td>
<td>Clone Detection (Wang et al., 2020)</td>
<td>Java</td>
<td>100,000</td>
<td>10,000</td>
<td>415,416</td>
</tr>
</tbody>
</table>
PLBART - Code Translation Result

**Dataset:** CodeXGLUE [Lu et. al. 2021]

**Metric:** EM (exact match %), BLEU-4 (max 100), CodeBLEU (max 100)

\[
\text{CodeBLEU} = 0.25 \times \text{token\_match} + 0.25 \times \text{keywords\_match} + 0.25 \times \text{syntax\_match} + 0.25 \times \text{dataflow\_match}
\]

<table>
<thead>
<tr>
<th>Methods</th>
<th>Java to C#</th>
<th>C# to Java</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BLEU</td>
<td>EM</td>
</tr>
<tr>
<td>Naive Copy</td>
<td>18.54</td>
<td>0</td>
</tr>
<tr>
<td>PBSMT</td>
<td>43.53</td>
<td>12.50</td>
</tr>
<tr>
<td>Transformer</td>
<td>55.84</td>
<td>33.00</td>
</tr>
<tr>
<td>RoBERTa (code)</td>
<td>77.46</td>
<td>56.10</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>79.92</td>
<td>59.00</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td>80.58</td>
<td>59.40</td>
</tr>
<tr>
<td>PLBART</td>
<td><strong>83.02</strong></td>
<td><strong>64.60</strong></td>
</tr>
</tbody>
</table>
Code Editing (Bug Fixing) Result

**Dataset:** Tufano *et. al.* 2019  
**Metric:** EM (exact match %), **BLEU-4** (max 100), **CodeBLEU** (max 100)

\[
\text{CodeBLEU} = 0.25 \times \text{token\_match} + 0.25 \times \text{keywords\_match} + 0.25 \times \text{syntax\_match} + 0.25 \times \text{dataflow\_match}
\]

<table>
<thead>
<tr>
<th>Methods</th>
<th>Small</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EM</td>
<td>BLEU</td>
</tr>
<tr>
<td>Naive Copy</td>
<td>0</td>
<td>78.06</td>
</tr>
<tr>
<td>Seq2Seq</td>
<td>10.00</td>
<td>76.76</td>
</tr>
<tr>
<td>Transformer</td>
<td>14.70</td>
<td>77.21</td>
</tr>
<tr>
<td>CodeBERT</td>
<td>16.40</td>
<td>77.42</td>
</tr>
<tr>
<td>GraphCodeBERT</td>
<td>17.30</td>
<td><strong>80.58</strong></td>
</tr>
<tr>
<td>PLBART</td>
<td><strong>19.21</strong></td>
<td>77.02</td>
</tr>
</tbody>
</table>
Code Translation Example

Input Code : C#

1 public int GetCells()
2 {
3     int size = 0;
4     foreach (char c in cells.Keys)
5     {
6         Cell e = At(c);
7         if (e.cmd >= 0 || e.@ref >= 0)
8             size++;
9     }
10     return size;
11 }

Generated Code : Java

1 public int getCells() {
2     Iterator<Character> i =
3         cells.keySet().iterator();
4     int size = 0;
5     for (; i.hasNext();)
6     {
7         Character c = i.next();
8         Cell e = at(c);
9         if (e.cmd >= 0 || e.@ref >= 0)
10             size++;
11     }
12     return size;
13 }
Take Away Point

Large scale model and pre-training for code generation help generate contextually correct code.
What’s Next?

PLBART “Learns” Contextual Correctness.
Does not “Enforce” like CODIT does.
Further research on enforcing Contextual Correctness.
Thanks
Questions?