Deep Just-In-Time Inconsistency Detection Between Comments and Source Code

Sheena Panthaprackel, Junyi Jessy Li, Milos Gligoric, Raymond J. Mooney
Source Code Comments

Document functionality, usage, implementation, error cases, ...

```java
/** Computes the highest value from the list of scores. */
public int getBestScore() {
    return Collections.max(scores);
}
```
Source Code Comments

When developers make code changes, they often fail to update comments accordingly.

```
/**Computes the highest value from the list of scores.*/
public int getBestScore() {
    return Collections.max(scores);
}
```

Leads to time-wasting confusion and vulnerability to bugs.

```
/**Computes the highest value from the list of scores.*/
public int getBestScore() {
    return Collections.min(scores);
}
```

Automatically detect inconsistent comments
Detecting Inconsistent Comments

### Just-In-Time Inconsistency Detection

```java
/** Computes the highest value from the list of scores. */
public int getBestScore() {
    return Collections.max(scores);
}
```

- return Collections.max(scores);
+ return Collections.min(scores);

### Post Hoc Inconsistency Detection

```java
/** Computes the highest value from the list of scores. */
public int getBestScore() {
    return Collections.max(scores);
}
```

- public int getBestScore() {
+ public double getBestScore() {

Inconsistent ✗ Consistent ✓
Outline

- Task
- Architecture
- Data
- Intrinsic Evaluation
- Extrinsic Evaluation
- Summary
Outline

● Task
● Architecture
● Data
● Intrinsic Evaluation
● Extrinsic Evaluation
● Summary
Task

Determine whether a comment (C) is inconsistent with a method (M)

```
/** Computes the highest value from the list of scores. */
public int getBestScore() {
    return Collections.max(scores);
}
```

Post Hoc: Given C and M

Just-In-Time: Given C, M, and M_{old}
Outline

- Task
- Architecture
- Data
- Intrinsic Evaluation
- Extrinsic Evaluation
- Summary
/** Computes the highest value from the list of scores. */
Code Representations

**Sequence-based**

Post hoc ($M$): $M$ as a sequence of tokens

```java
public int getBestScore() { return Collections.min(scores); }
```

Just-In-Time ($M_{edit}$): Edits between $M_{old}$ and $M$ as a sequence of tokens

```java
public int getBestScore() { return Collections.<Keep>max.<ReplaceOld>min.<ReplaceNew>Collections.<KeepEnd>
```

**AST-based**

Just-In-Time ($T_{edit}$): Graph representation of AST node edits between $M_{old}$ and $M$
Encoding Code Representations

- Sequence-based representations ($\mathbf{M}, \mathbf{M}_{\text{edit}}$) encoded with a biGRU.
- AST-based representations ($\mathbf{T}, \mathbf{T}_{\text{edit}}$) encoded with GGNNs.

Gated Graph Neural Network (GGNN)

- Node state representations are updated across 8 steps of message passing
- Message passing is done through different edge types
- Initial state representations consist of a word embedding, concatenated with an edit embedding (for the just-in-time setting)
Architecture

/* Computes the highest value from the list of scores. */

Comment encoder (biGRU) → Self-attention → Contextualized comment encoder (biGRU)

Code encoder → Multi-head attention

Sequence code encoder (biGRU) and/or
AST code encoder (GGNN)

Fully connected layer → Softmax

Inconsistent?
Outline

- Task
- Architecture
- Data
- Intrinsic Evaluation
- Extrinsic Evaluation
- Summary
Data Collection

<table>
<thead>
<tr>
<th>Commit History</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit 1</td>
</tr>
<tr>
<td>Commit 2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Commit N-1</td>
</tr>
<tr>
<td>Commit N</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

- Inconsistent: \( C \neq C_{\text{new}} \) and \( M_{\text{old}} \neq M \) - positive examples
- Consistent: \( C = C_{\text{new}} \) and \( M_{\text{old}} \neq M \) - negative examples

- \((C, M, M_{\text{old}})\) in Our Dataset

<table>
<thead>
<tr>
<th></th>
<th>Train</th>
<th>Valid</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td>32,988</td>
<td>3,756</td>
<td>3,944</td>
<td>40,688</td>
</tr>
<tr>
<td>Projects</td>
<td>829</td>
<td>332</td>
<td>357</td>
<td>1,518</td>
</tr>
</tbody>
</table>

- \@return, \@param, and summary comments
- Balanced label distribution
- Cleaned 300 examples from test set for more reliable evaluation
Outline

- Task
- Architecture
- Data
- Intrinsic Evaluation
- Extrinsic Evaluation
- Summary
## Intrinsic Evaluation: Results

<table>
<thead>
<tr>
<th>Cleaned Test Sample</th>
<th>Full Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Intrinsic Evaluation: Results

- Liu et al. (2018)

**Cleaned Test Sample**
- F1: 75.8
- Accuracy: 76.3

**Full Test Set**
- F1: 70.0
- Accuracy: 72.6
Intrinsic Evaluation: Results

<table>
<thead>
<tr>
<th>Liu et al. (2018)</th>
<th>Post Hoc SEQ</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cleaned Test Sample</strong></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>Accuracy</td>
</tr>
<tr>
<td>75.8</td>
<td>63.0</td>
</tr>
<tr>
<td>76.3</td>
<td>60.3</td>
</tr>
<tr>
<td><strong>Full Test Set</strong></td>
<td></td>
</tr>
<tr>
<td>F1</td>
<td>Accuracy</td>
</tr>
<tr>
<td>70.0</td>
<td>66.3</td>
</tr>
<tr>
<td>72.6</td>
<td>62.8</td>
</tr>
</tbody>
</table>
Intrinsic Evaluation: Results

- Our Just-In-Time approach can outperform post hoc and baseline models
Intrinsic Evaluation: Results

- Our Just-In-Time approach can outperform post hoc and baseline models
Intrinsic Evaluation: Results

Our Just-In-Time approach can outperform post hoc and baseline models.

No significant difference between SEQ, GRAPH, and HYBRID approaches.
Intrinsic Evaluation: Results

- Our Just-In-Time approach can outperform post hoc and baseline models
- No significant difference between SEQ, GRAPH, and HYBRID approaches
Intrinsic Evaluation: Results

- Our Just-In-Time approach can outperform post hoc and baseline models
- No significant difference between SEQ, GRAPH, and HYBRID approaches
- Incorporating auxiliary features can further boost performance
Intrinsic Evaluation: Results

- Our Just-In-Time approach can outperform post hoc and baseline models
- No significant difference between SEQ, GRAPH, and HYBRID approaches
- Incorporating auxiliary features can further boost performance
- Analogous performance between cleaned and full test sets
Integrating with Update

On its own, inconsistency detection can only flag comments that developers failed to update.

```java
/** Computes the highest value from the list of scores. */
public int getBestScore() {
    return Collections.max(scores);
}
```

Automatically detect and update inconsistent comments
Outline

- Task
- Architecture
- Data
- Intrinsic Evaluation
- Extrinsic Evaluation
- Summary
Extrinsic Evaluation: Integrating with Update

Given \( C, M_{\text{old}} \), and \( M \), produce \( C_{\text{new}} \), which is consistent with \( M \).

\[
(C, M_{\text{old}}), \quad (C_{\text{new}}, M) \rightarrow \text{Inconsistent} \rightarrow \text{positive examples}
\]

\[
[(C_1, M_1), (C_2, M_2)] = [(C, M_{\text{old}}), (C_{\text{new}}, M)]
\]

Our Dataset

\[
(C \neq C_{\text{new}} \text{ and } M_{\text{old}} \neq M)
\]

\[
(C = C_{\text{new}} \text{ and } M_{\text{old}} \neq M)
\]

\[
\text{Inconsistent} \rightarrow \text{positive examples}
\]

\[
\text{Consistent} \rightarrow \text{negative examples}
\]

\[
(C, M_{\text{old}}), \quad M \rightarrow C_{\text{new}} = \begin{cases} 
\text{Generate } C_{\text{new}} \text{ by updating } C \text{ to reflect } M & \text{if } C \text{ is inconsistent with } M \\
\text{Copy } C_{\text{new}} = C & \text{else}
\end{cases}
\]
Extrinsic Evaluation: Integrating with Update

Given $C, M_{\text{old}}$, and $M$, produce $C_{\text{new}}$, which is consistent with $M$. There are many cases in which code changes do not require a comment update.

$[(C_1, M_1), (C_2, M_2)] = [(C, M_{\text{old}}), (C_{\text{new}}, M)]$

Comment Update Model
(Panthaplackel et al., ACL 2020)

If $C$ is inconsistent with $M$
Generate $C_{\text{new}}$ by updating $C$ to reflect $M$
Else
Copy $C_{\text{new}} = C$

Inconsistent -> positive examples
Consistent -> negative examples

(C, $M_{\text{old}}$), $M_{\text{old}} \neq M$

(C, $M_{\text{old}}$), $M_{\text{old}} \neq M$

(C, $M_{\text{old}}$), $M_{\text{old}} \neq M$

Our Dataset

(C, $M_{\text{old}}$), $M_{\text{old}} \neq M$

(C, $M_{\text{old}}$), $M_{\text{old}} \neq M$

(C, $M_{\text{old}}$), $M_{\text{old}} \neq M$

(C, $M_{\text{old}}$), $M_{\text{old}} \neq M$
Extrinsic Evaluation: Integrating with Update

**Update w/ Implicit Detection**

- **Training**
  - $+$, $-$
  - Update
    - Supervision: $C_{\text{new}}$

- **Inference**
  - $P_{\text{inconsistency label}} = \text{output(Update)} \neq C$
  - $P_{\text{comment}} = \text{output(Update)}$

**Pretrained Update and Detection**

- $+$
  - Update
    - Supervision: $C_{\text{new}}$

**Jointly Trained Update and Detection**

- $+$, $-$
  - Update
    - Supervision: $C_{\text{new}}$

- Detection
  - Supervision: $C \neq C_{\text{new}}$

- $+$, $-$
  - Shared: Embeddings, $C$
    - $M_{\text{edit}}$
    - $C_{\text{encoder}}$

- Detection
  - Supervision: $C \neq C_{\text{new}}$

- $P_{\text{inconsistency label}} = \text{output(Detection)}$

  - if output(Detection) is Inconsistent:
    - $P_{\text{comment}} = \text{output(Update)}$
  - else:
    - $P_{\text{comment}} = C$
Extrinsic Evaluation: Results

Cleaned Test Sample
Extrinsic Evaluation: Results

Cleaned Test Sample

- BLEU-4
  - Never Update: 72.1
  - Panthaplackel et al. (2020): 68.7

- GLEU
  - Never Update: 68.2
  - Panthaplackel et al. (2020): 67.4

- F1
  - Never Update: 69.0
  - Panthaplackel et al. (2020): 69.0

- Acc
  - Never Update: 50.0
  - Panthaplackel et al. (2020): 57.1
Extrinsic Evaluation: Results

Cleaned Test Sample

<table>
<thead>
<tr>
<th></th>
<th>BLEU-4</th>
<th>GLEU</th>
<th>F1</th>
<th>Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never Update</td>
<td>72.1</td>
<td>68.2</td>
<td>0.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Panthapackel et al. (2020)</td>
<td>74.7</td>
<td>67.4</td>
<td>37.7</td>
<td>57.1</td>
</tr>
<tr>
<td>Update w/ Implicit Detection</td>
<td>68.7</td>
<td>67.4</td>
<td>69.0</td>
<td>61.7</td>
</tr>
</tbody>
</table>
Extrinsic Evaluation: Results

CLEANED TEST SAMPLE

- BLEU-4
- GLEU
- F1
- Acc

- Never Update
- Panthaplackel et al. (2020)
- Update w/ Implicit Detection
- Pretrained Update + Detection
- Jointly Trained Update + Detection
Extrinsic Evaluation: Results

- Including an explicit inconsistency detection leads to improved performance across update and detection metrics.
Extrinsic Evaluation: Results

- Including an explicit inconsistency detection leads to improved performance across update and detection metrics.
- Pretrained and jointly trained systems perform similarly.
Outline

- Task
- Architecture
- Data
- Intrinsic Evaluation
- Extrinsic Evaluation
- Summary
Summary

- We formulated a deep learning approach for just-in-time inconsistency detection between comments and code by learning to relate comments and code changes.
- We show that our approach can outperform multiple baselines as well as post hoc models.
- We also demonstrate that our approach can be used to build a comprehensive comment maintenance system which detects and updates inconsistent comments.

Code and data available: https://github.com/panthap2/deep-jit-inconsistency-detection
Contact: Sheena Panthaplackel <spantha@cs.utexas.edu>