# Contextuality of Code Representation Learning

Yi Li [1]    Shaohua Wang [1]    Tien N. Nguyen [2]

[1] *Department of Informatics, New Jersey Institute of Technology*
[2] *Computer Science Department, The University of Texas at Dallas*

Presenter: [Aashish Yadavally] [2]

**Q**. How can we *automate the process of answering questions* about two words, their meanings, and their relationships?

**Q**. How can we *automate the process of answering questions* about two words, their meanings, and their relationships?

# WordNet: A Lexical Database for English

George A. Miller

[1] George A. Miller. 1995. WordNet: a lexical database for English. Commun. ACM 38, 11 (Nov. 1995), 39–41. https://doi.org/10.1145/219717.219748

**Q**. How can we *automate the process of answering questions* about two words, their meanings, and their relationships?

## WordNet: A Lexical Database for English

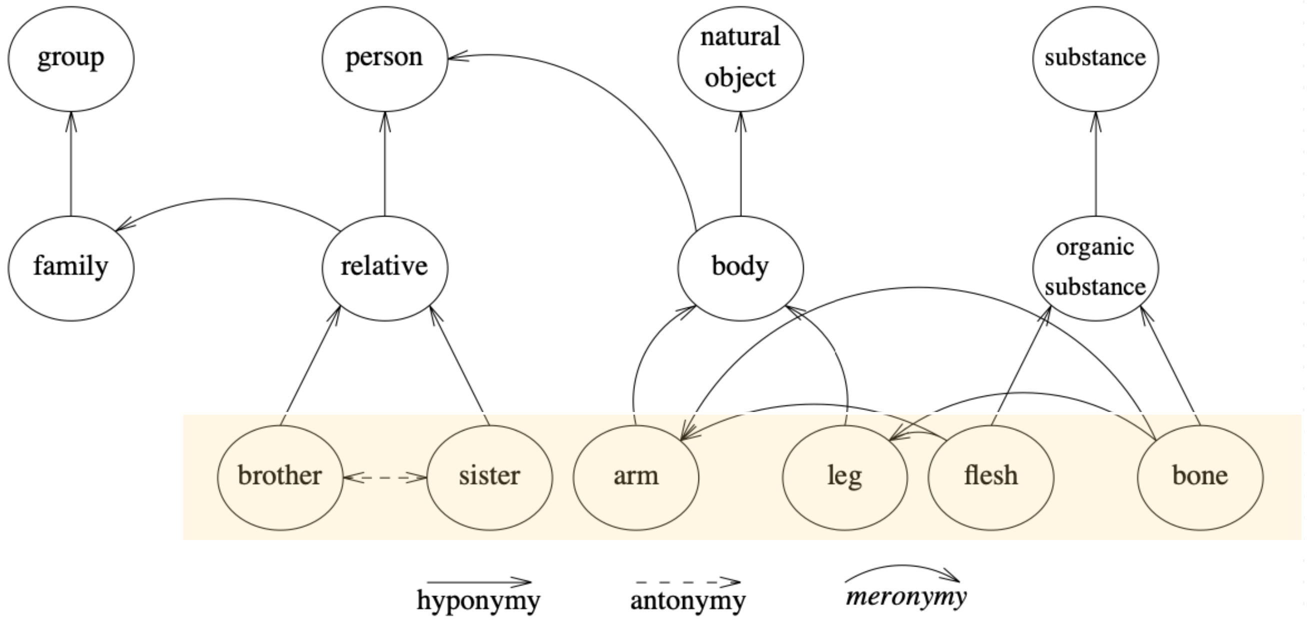
George A. Miller

"...inspired by psycholinguistic theories of human lexical memory..."

[1] George A. Miller. 1995. WordNet: a lexical database for English. Commun. ACM 38, 11 (Nov. 1995), 39–41. https://doi.org/10.1145/219717.219748

Q. How can we *automate the process of answering questions* about two words, their meanings, and their relationships?

Figure 2. Network representation of three semantic relations among an illustrative variety of lexical concepts




[2] George A. Miller, Nouns in WordNet: A Lexical Inheritance System, *International Journal of Lexicography*, Volume 3, Issue 4, Winter 1990, Pages 245-264, https://doi.org/10.1093/ijl/3.4.245

**Q**. How can we *automate the process of answering questions* about two words, their meanings, and their relationships?

Figure 2. Network representation of three semantic relations

**not scalable**





[2] George A. Miller, Nouns in WordNet: A Lexical Inheritance System, *International Journal of Lexicography*, Volume 3, Issue 4, Winter 1990, Pages 245-264, https://doi.org/10.1093/ijl/3.4.245

**Q**. How can we *automate the process of answering questions* about two words, their meanings, and their relationships?

**A.** word embeddings

**Q**. How can we *automate the process of answering questions* about two words, their meanings, and their relationships?

**A.** word embeddings: *real-valued vector* representations.

**Q**. How to generate such word embeddings?

**Q**. How to generate such word embeddings?

Challenge: Words have *different meanings* in *different contexts*.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| I | shall | get | to | the | University | at | three | PM | **Arrive** |
| I | will | go | get | a | pen | | | | **Possession** |
| I | get | it | | | | | | | **Understanding** |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| I | shall | get | to | the | University | at | three | PM |
| I | will | go | get | a | pen | | | |
| I | get | it | | | | | | |

**Arrive**

**Possession**

**Understanding**

**"word senses"**

**Q**. How can we *automate the process of answering questions* about two words and their meanings?

**A.** word embeddings: *real-valued vector* representations.

**Polysemy**

**Word2Vec**

**GloVe**

(*Global Vectors for Word Representations*)

**FastText**

**static**

**ELMo**

(*Embeddings from Language Models*)

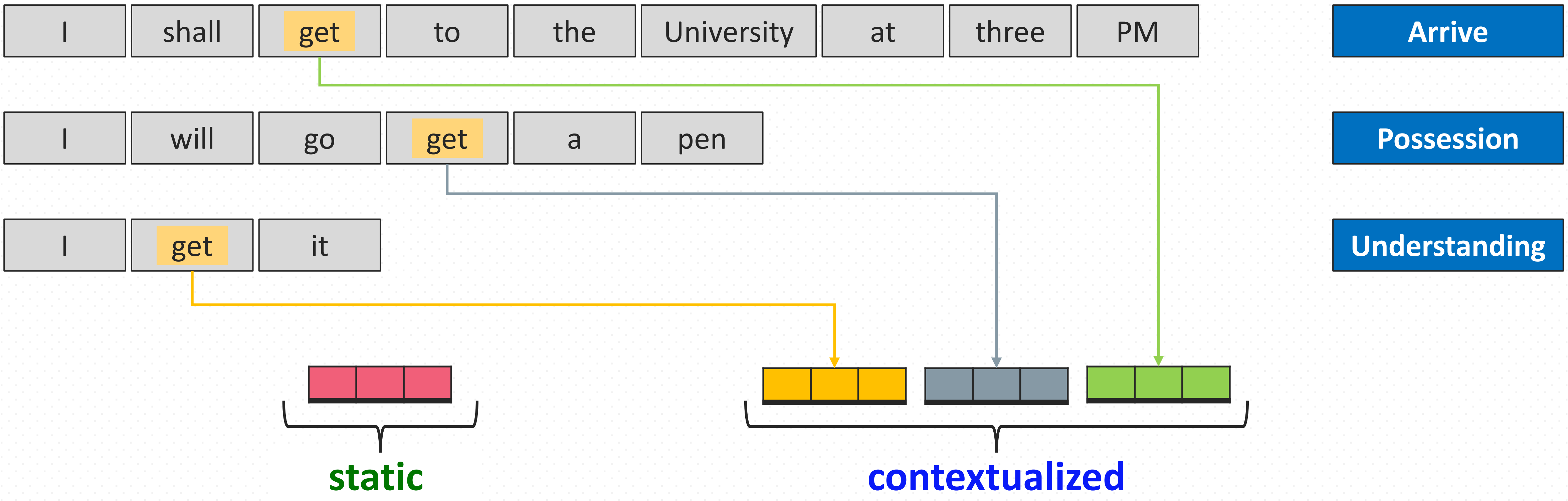
**BERT**

(*Bidirectional Encoder Representations from Transformers*)

**ERNIE**

(*Enhanced Language Representation with Informative Entities*)

**contextualized**


I shall get to the University at three PM
I will go get a pen
I get it
Arrive
Possession
Understanding
static
contextualized

# Contextuality of Code Representation Learning

# Motivating Example

```
private static ArrayList<Word>
      postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
    if(length > 0) {
      String prevWord =
            newSentence.get(newSentence.size()-1).toString();
      String curWord = word.toString();
      ...
}

public ArrayList<Word> greedilySegmentWords(String s) {
    List<Word> segmentedWords = new ArrayList<>();
    int length = s.length();
    int start = 0;
    while (start < length) {
      int end = Math.min(length, start + maxLength);
      while (end > start + 1) {
        String nextWord = s.substring(start, end);
        if (words.contains(nextWord)) {
          segmentedWords.add(new Word(nextWord));
      ...
}
```

# Motivating Example

```
private static ArrayList<Word>
      postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
    if(length > 0) {

public ArrayList<Word> greedilySegmentWords(String s) {
    List<Word> segmentedWords = new ArrayList<>();
    int length = s.length();
    int start = 0;
    while (start < length) {
      int end = Math.min(length, start + maxLength);
      while (end > start + 1) {
        String nextWord = s.substring(start, end);
        if (words.contains(nextWord)) {
          segmentedWords.add(new Word(nextWord));
      ...
}
```

**Observation 1.1**

Context, in source code is **important** and can provide the semantics for variable/method names.

# Motivating Example

```
private static ArrayList<Word>
      postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
    if(length > 0) {
```

**Observation 1.1**

Context, in source code is **important** and can provide the semantics for variable/method names.

```

public ArrayList<Word> greedilySegmentWords(String s) {
    List<Word> segmentedWords = new ArrayList<>();
```

**Observation 1.2**

Keywords (e.g., ‘for’, ‘while’, ‘if’), operators (e.g., ‘+’, ‘-’), and separators (e.g., ‘;’) maintain same meaning throughout.

```
        String nextWord = s.substring(start, end);
        if (words.contains(nextWord)) {
          segmentedWords.add(new Word(nextWord));
      ...
}
```

# Motivating Example

```
private static ArrayList<Word>
      postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
   if(length > 0) {
```

**Observation**

**[Mixed Polysemy]** *Code tokens exhibit mixed polysemy in which some tokens have different meanings depending on different contexts, while others maintain the same meaning regardless of context.*

```
public ArrayList<Word> greedilySegmentWords(String s) {
   List<Word> segmentedWords = new ArrayList<>();
   int length = s.length();
   int start = 0;
   while (start < length) {
     int end = Math.min(length, start + maxLength);
     while (end > start + 1) {
       String nextWord = s.substring(start, end);
       if (words.contains(nextWord)) {
         segmentedWords.add(new Word(nextWord));
     ...
}
```

**Q**. Which of the static or contextualized embeddings fit better with the mixed polysemy nature of source code?

# Contextuality of Code Representation Learning

```
private static ArrayList<Word>
      postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
    if(length > 0) {
      String prevWord =
            newSentence.get(newSentence.size()-1).toString();
      String curWord = word.toString();
      ...
}

public ArrayList<Word> greedilySegmentWords(String s) {
    List<Word> segmentedWords = new ArrayList<>();
    int length = s.length();
    int start = 0;
    while (start < length) {
      int end = Math.min(length, start + maxLength);
      while (end > start + 1) {
        String nextWord = s.substring(start, end);
        if (words.contains(nextWord)) {
          segmentedWords.add(new Word(nextWord));
      ...
}
```

Source Code

```java
private static ArrayList<Word>
     postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
    if(length > 0) {
      String prevWord =
           newSentence.get(newSentence.size()-1).toString();
      String curWord = word.toString();
      ...
}

public ArrayList<Word> greedilySegmentWords(String s) {
    List<Word> segmentedWords = new ArrayList<>();
    int length = s.length();
    int start = 0;
    while (start < length) {
      int end = Math.min(length, start + maxLength);
      while (end > start + 1) {
        String nextWord = s.substring(start, end);
        if (words.contains(nextWord)) {
          segmentedWords.add(new Word(nextWord));
      ...
}
```

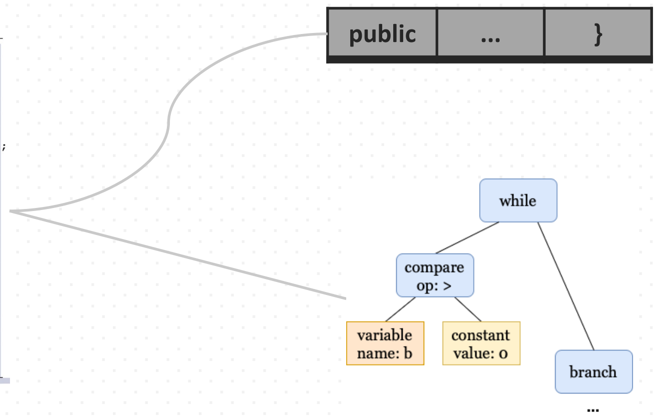
**Source Code**


...it can be represented as:

- a ***sequence*** of tokens

```java
private static ArrayList<Word>
     postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
    if(length > 0) {
      String prevWord =
           newSentence.get(newSentence.size()-1).toString();
      String curWord = word.toString();
      ...
}

public ArrayList<Word> greedilySegmentWords(String s) {
    List<Word> segmentedWords = new ArrayList<>();
    int length = s.length();
    int start = 0;
    while (start < length) {
      int end = Math.min(length, start + maxLength);
      while (end > start + 1) {
        String nextWord = s.substring(start, end);
        if (words.contains(nextWord)) {
          segmentedWords.add(new Word(nextWord));
      ...
}
```

**Source Code**

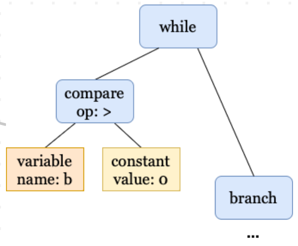


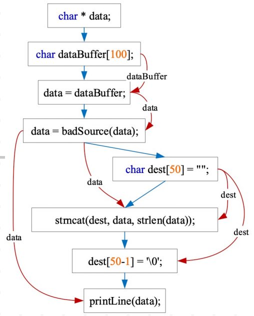

...it can be represented as:

- a *sequence* of tokens
- abstract syntax ***tree***

```
private static ArrayList<Word>
    postProcessSentence(ArrayList<Word> sentence) {
 ArrayList<Word> newSentence = new ArrayList<>();
 int length = sentence.size();
 for(Word word : sentence) {
  if(length > 0) {
   String prevWord =
       newSentence.get(newSentence.size()-1).toString();
   String curWord = word.toString();
   ...
}

public ArrayList<Word> greedilySegmentWords(String s) {
  List<Word> segmentedWords = new ArrayList<>();
  int length = s.length();
  int start = 0;
  while (start < length) {
   int end = Math.min(length, start + maxLength);
   while (end > start + 1) {
    String nextWord = s.substring(start, end);
    if (words.contains(nextWord)) {
     segmentedWords.add(new Word(nextWord));
   ...
}
```

**Source Code**

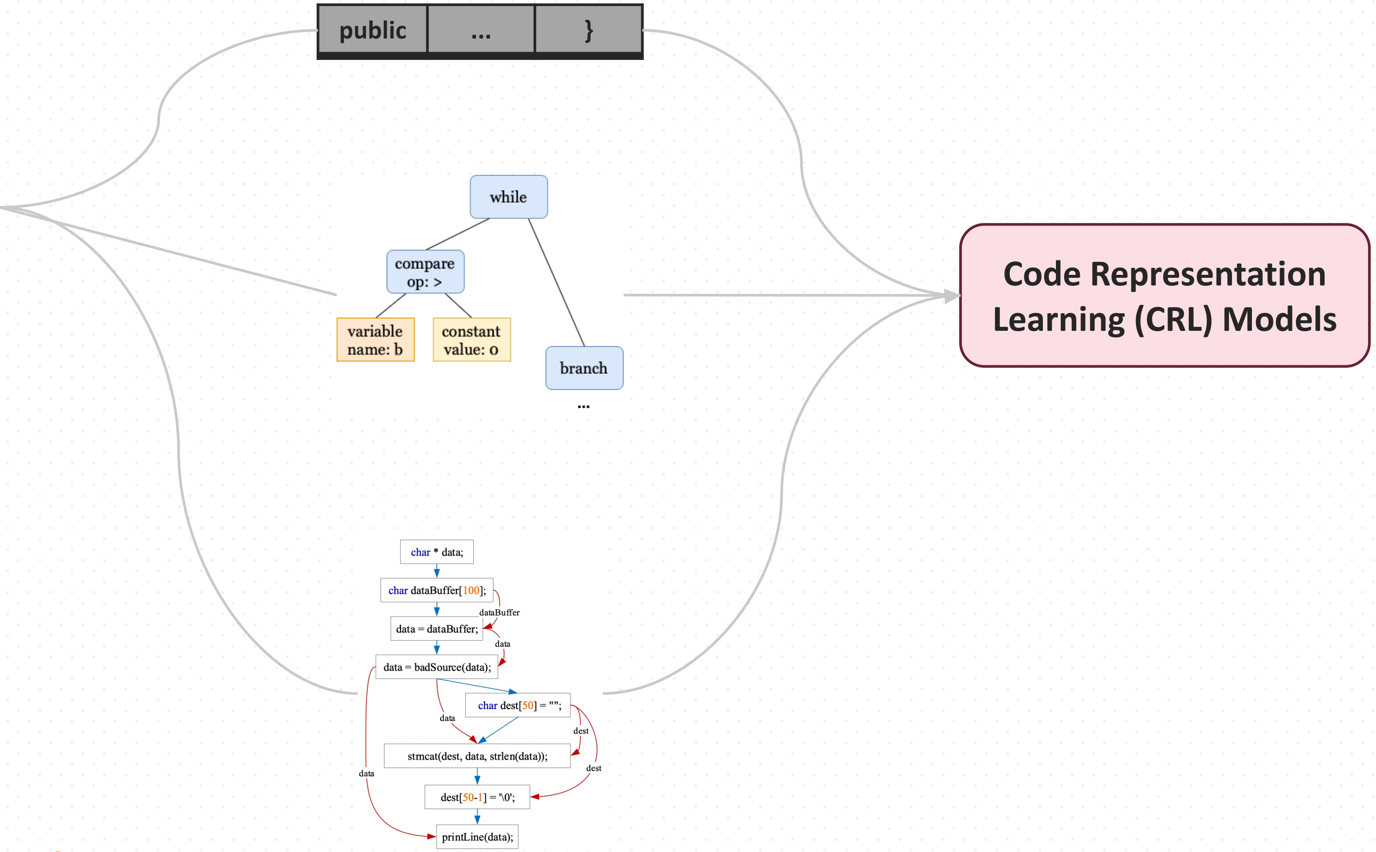




...it can be represented as:

- a *sequence* of tokens
- abstract syntax *tree*
- program dependence ***graph***

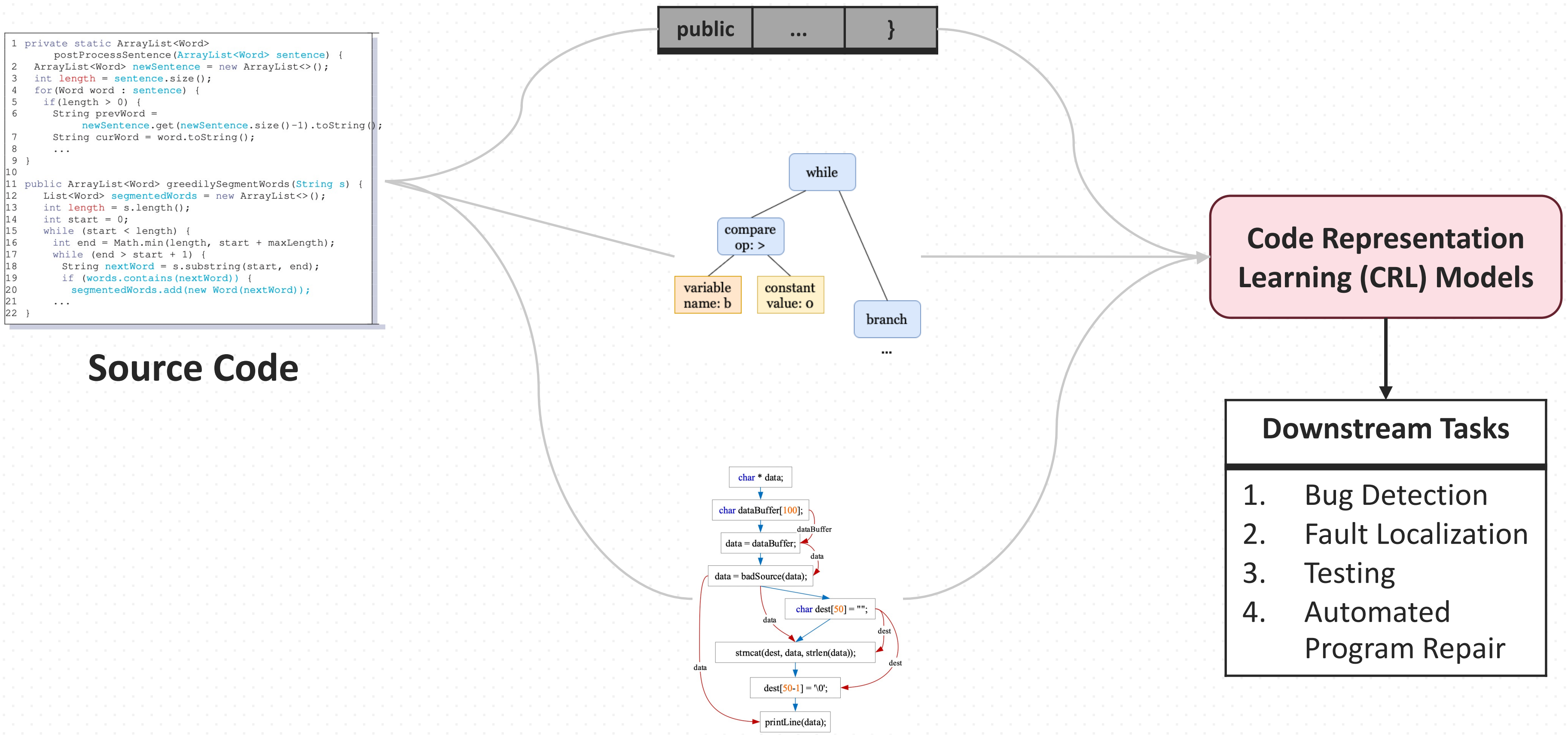
```
private static ArrayList<Word>
      postProcessSentence(ArrayList<Word> sentence) {
  ArrayList<Word> newSentence = new ArrayList<>();
  int length = sentence.size();
  for(Word word : sentence) {
    if(length > 0) {
      String prevWord =
            newSentence.get(newSentence.size()-1).toString();
      String curWord = word.toString();
      ...
}

public ArrayList<Word> greedilySegmentWords(String s) {
    List<Word> segmentedWords = new ArrayList<>();
    int length = s.length();
    int start = 0;
    while (start < length) {
      int end = Math.min(length, start + maxLength);
      while (end > start + 1) {
        String nextWord = s.substring(start, end);
        if (words.contains(nextWord)) {
          segmentedWords.add(new Word(nextWord));
      ...
}
```

**Source Code**




...it can be represented as:

- a ***sequence*** of tokens
- abstract syntax ***tree***
- program dependence ***graph***


public
...
}
while
compare op: >
variable name: b
constant value: 0
branch
...
Source Code
Code Representation Learning (CRL) Models
Downstream Tasks
1. Bug Detection
2. Fault Localization
3. Testing
4. Automated Program Repair

# Empirical Evaluation

**(RQ1)** Which of the static or contextualized embeddings fit better with the mixed polysemy nature of source code?

What is the degree of contextuality for sequence-based, tree-based, and graph-based code representation learning models?

# Empirical Evaluation (RQ1)

## Dataset

- *Code2Vec*, with 10,222 top-ranked GitHub Java projects
- 1.8M+ unique methods
- 10 samples, each containing 18K+ methods

# Empirical Evaluation (RQ1)

## Models

- a ***sequence*** of tokens → Word2Vec, GloVe, FastText, ELMo, BERT, CodeBERT

# Empirical Evaluation (RQ1)

## Models

- a ***sequence*** of tokens → Word2Vec, GloVe, FastText, ELMo, BERT, CodeBERT
- abstract syntax ***tree*** → TreeLSTM, ASTNN, TBCNN, TreeCAPS

# Empirical Evaluation (RQ1)

## Models

- a ***sequence*** of tokens → Word2Vec, GloVe, FastText, ELMo, BERT, CodeBERT
- abstract syntax ***tree*** → TreeLSTM, ASTNN, TBCNN, TreeCAPS
- program dependence ***graph*** → Node2Vec, DeepWalk, Graph2Vec

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.

# Empirical Evaluation (RQ1)

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.

2. Intra-Similarity (IntraSim): The average cosine similarity between a program unit's vector representation and the average of those vectors for the units in that context.

# Empirical Evaluation (RQ1)

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.

2. Intra-Similarity (IntraSim): The average cosine similarity between a program unit's vector representation and the average of those vectors for the units in that context.

IntraSim ↓ SelfSim ↓

*CRL model gives each program unit a vector representation that is distinct from all other vectors in the context.*

# Empirical Evaluation (RQ1)

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.

2. Intra-Similarity (IntraSim): The average cosine similarity between a program unit's vector representation and the average of those vectors for the units in that context.

IntraSim ↓ SelfSim ↓

*CRL model gives each program unit a vector representation that is distinct from all other vectors in the context.*

IntraSim ↑ SelfSim ↓

*CRL model simply contextualizes the program units by making the representation vectors converge.*

# Empirical Evaluation (RQ1)

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.

2. Intra-Similarity (IntraSim): The average cosine similarity between a program unit's vector representation and the average of those vectors for the units in that context.

3. Maximum Explainable Variance (MEV): The proportion of variance in the contextualized representations of program unit that can be explained by its first principal component.

# Empirical Evaluation (RQ1)

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.

2. Intra-Similarity (IntraSim): The average cosine similarity between a program unit's vector representation and the average of those vectors for the units in that context.

3. Maximum Explainable Variance (MEV): The proportion of variance in the contextualized representations of program unit that can be explained by its first principal component.

MEV $\rightarrow$ 0

*Static embedding is a poor replacement.*

# Empirical Evaluation (RQ1)

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.
2. Intra-Similarity (IntraSim): The average cosine similarity between a program unit's vector representation and the average of those vectors for the units in that context.
3. Maximum Explainable Variance (MEV): The proportion of variance in the contextualized representations of program unit that can be explained by its first principal component.

MEV $\rightarrow$ 0

*Static embedding is a poor replacement.*

MEV $\rightarrow$ 1

*Static embedding is perfect replacement for contextualized representations.*

# Empirical Evaluation (RQ1)

## Contextuality Measurement

1. Self-Similarity (SelfSim): The average cosine similarity between a program unit's contextualized representation vectors across all unique contexts.

2. Intra-Similarity (IntraSim): The average cosine similarity between a program unit's vector representation and the average of those vectors for the units in that context.

3. Maximum Explainable Variance (MEV): The proportion of variance in the contextualized representations of program unit that can be explained by its first principal component.

4. Anisotropy: Vector representations are more isotropic when the average cosine similarity between uniformly randomly sampled program units is close to 0.

   The more contextualized the vector representations, the more anisotropic they are (i.e., the closer that average is to 1).

# Empirical Evaluation (RQ1)

TABLE I: Average `SelfSim` for Sequence-based Models (RQ1)

| Unit/Context | Unit: Sub-token, Context: Statement | | | | | Unit: Token, Context: Statement | | | | | Statement/Method |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Identifier | Keyword | Separator | Operator | Literal | Identifier | Keyword | Separator | Operator | Literal | |
| ELMo | 0.75 | 0.61 | 0.97 | 0.82 | 0.75 | 0.65 | 0.63 | 0.94 | 0.75 | 0.82 | 0.52 |
| BERT | 0.34 | 0.25 | 0.83 | 0.41 | 0.56 | 0.31 | 0.29 | 0.77 | 0.37 | 0.53 | 0.19 |
| CodeBERT | 0.32 | 0.26 | 0.79 | 0.40 | 0.54 | 0.30 | 0.78 | 0.75 | 0.35 | 0.49 | 0.20 |

# Empirical Evaluation (RQ1)

TABLE I: Average `SelfSim` for Sequence-based Models (RQ1)

| Unit/Context | Unit: Sub-token, Context: Statement | | | | | Unit: Token, Context: Statement | | | | | Statement/Method |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Identifier | Keyword | Separator | Operator | Literal | Identifier | Keyword | Separator | Operator | Literal | |
| ELMo | 0.75 | 0.61 | 0.97 | 0.82 | 0.75 | 0.65 | 0.63 | 0.94 | 0.75 | 0.82 | 0.52 |
| BERT | 0.34 | 0.25 | 0.83 | 0.41 | 0.56 | 0.31 | 0.29 | 0.77 | 0.37 | 0.53 | 0.19 |
| CodeBERT | 0.32 | 0.26 | 0.79 | 0.40 | 0.54 | 0.30 | 0.78 | 0.75 | 0.35 | 0.49 | 0.20 |

# Empirical Evaluation (RQ1)

TABLE I: Average `SelfSim` for Sequence-based Models (RQ1)

| Unit/Context | Unit: Sub-token, Context: Statement | | | | | Unit: Token, Context: Statement | | | | | Statement/Method |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Identifier | Keyword | Separator | Operator | Literal | Identifier | Keyword | Separator | Operator | Literal | |
| ELMo | 0.75 | 0.61 | 0.97 | 0.82 | 0.75 | 0.65 | 0.63 | 0.94 | 0.75 | 0.82 | 0.52 |
| BERT | 0.34 | 0.25 | 0.83 | 0.41 | 0.56 | 0.31 | 0.29 | 0.77 | 0.37 | 0.53 | 0.19 |
| CodeBERT | 0.32 | 0.26 | 0.79 | 0.40 | 0.54 | 0.30 | 0.78 | 0.75 | 0.35 | 0.49 | 0.20 |

TABLE II: Avg. `IntraSim` for Sequence-based Models (RQ1)

| | Unit: Sub-token | Unit: Token | Unit: Statement |
|---|---|---|---|
| Word2vec | 0.84 | 0.72 | 0.73 |
| GloVe | 0.77 | 0.72 | 0.76 |
| FastText | 0.82 | 0.73 | 0.70 |
| ELMo | 0.69 | 0.43 | 0.58 |
| BERT | 0.42 | 0.29 | 0.31 |
| CodeBERT | 0.43 | 0.27 | 0.29 |

# Empirical Evaluation (RQ1)

TABLE V: Average `SelfSim` for Tree-based Models (RQ2)

| Unit/Context | AST-node for token/AST-subtree for Statement | | | | | Statement/ Method |
|---|---|---|---|---|---|---|
| | Identifier | Keyword | Separator | Operator | Literal | |
| Tree-LSTM | 1 | - | - | 1 | 1 | 1 |
| ASTNN | 1 | - | - | 1 | 1 | 0.65 |
| TBCNN | 0.71 | - | - | 0.87 | 0.77 | 0.67 |
| TreeCaps | 0.54 | - | - | 0.72 | 0.65 | 0.52 |

TABLE VI: Average `IntraSim` for Tree-based Models (RQ2)

| | Unit: AST-node for Token | Unit: AST-subtree for Stmt |
|---|---|---|
| Tree-LSTM | 0.76 | 0.73 |
| ASTNN | 0.62 | 0.64 |
| TBCNN | 0.67 | 0.66 |
| TreeCaps | 0.51 | 0.54 |

# Empirical Evaluation (RQ1)

TABLE IX: Contextuality for Graph-based Models (RQ3)

| | SelfSim | IntraSim | MEV | Anisotropy |
|---|---|---|---|---|
| Node2Vec | 0.72 | 0.63 | 0.02 | 0.53 |
| Deepwalk | 0.65 | 0.58 | 0.01 | 0.62 |
| Graph2Vec | 0.38 | 0.27 | 0.01 | 0.75 |

# Empirical Evaluation (RQ1)

*Neither static nor contextualized models produce embeddings that fit with the nature of mixed polysemy of source code.*

| | | | | |
|---|---|---|---|---|
| Deepwalk | 0.65 | 0.58 | 0.01 | 0.62 |
| Graph2Vec | 0.38 | 0.27 | 0.01 | 0.75 |

# Empirical Evaluation

**(RQ1)** Which of the static or contextualized embeddings fit better with the mixed polysemy nature of source code?

What is the degree of contextuality for sequence-based, tree-based, and graph-based code representation learning models?

**(RQ2)** Impact of Contextuality on Bug Detection

# Empirical Evaluation (RQ2)

TABLE X: Impacts on Statement-level Bug Detection (RQ4)

| A. Sequence-based | Precision | Recall | F-score | AUC |
|---|---|---|---|---|
| Word2vec | 0.19 | 0.17 | 0.18 | 0.54 |
| GloVe | 0.21 | 0.18 | 0.19 | 0.56 |
| FastText | 0.18 | 0.18 | 0.18 | 0.52 |
| ELMo | 0.34 | 0.23 | 0.28 | 0.61 |
| BERT | 0.46 | 0.33 | 0.39 | 0.67 |
| CodeBERT | 0.47 | 0.34 | 0.40 | 0.69 |
| **B. Tree-based** | **Precision** | **Recall** | **F-score** | **AUC** |
| Tree-LSTM | 0.25 | 0.22 | 0.23 | 0.56 |
| ASTNN | 0.27 | 0.24 | 0.25 | 0.64 |
| TBCNN | 0.35 | 0.26 | 0.30 | 0.62 |
| TreeCaps | 0.39 | 0.31 | 0.35 | 0.66 |
| **C. Graph-based** | **Precision** | **Recall** | **F-score** | **AUC** |
| Node2vec | 0.28 | 0.25 | 0.26 | 0.63 |
| DeepWalk | 0.32 | 0.26 | 0.29 | 0.61 |
| Graph2vec | 0.40 | 0.33 | 0.36 | 0.68 |

# Empirical Evaluation (RQ2)

TABLE XI: Impacts on Method-level Bug Detection (RQ4)

| A. Sequence-based | Precision | Recall | F-score | AUC |
|---|---|---|---|---|
| Word2vec | 0.21 | 0.27 | 0.24 | 0.59 |
| GloVe | 0.26 | 0.26 | 0.26 | 0.62 |
| FastText | 0.24 | 0.28 | 0.26 | 0.57 |
| ELMo | 0.37 | 0.45 | 0.41 | 0.67 |
| BERT | 0.52 | 0.56 | 0.54 | 0.72 |
| CodeBERT | 0.54 | 0.59 | 0.56 | 0.73 |
| **B. Tree-based** | **Precision** | **Recall** | **F-score** | **AUC** |
| Tree-LSTM | 0.33 | 0.26 | 0.29 | 0.60 |
| ASTNN | 0.46 | 0.41 | 0.43 | 0.67 |
| TBCNN | 0.45 | 0.42 | 0.43 | 0.66 |
| TreeCaps | 0.51 | 0.47 | 0.49 | 0.69 |
| **C. Tree-based** | **Precision** | **Recall** | **F-score** | **AUC** |
| Node2vec | 0.28 | 0.30 | 0.29 | 0.61 |
| DeepWalk | 0.32 | 0.31 | 0.31 | 0.65 |
| Graph2vec | 0.47 | 0.44 | 0.45 | 0.71 |

# Empirical Evaluation (RQ2)

TABLE XI: Impacts on Method-level Bug Detection (RQ4)

| A. Sequence-based | Precision | Recall | F-score | AUC |
|---|---|---|---|---|
| Word2vec | 0.21 | 0.27 | 0.24 | 0.59 |
| GloVe | 0.26 | 0.26 | 0.26 | 0.62 |
| FastText | 0.24 | 0.28 | 0.26 | 0.57 |
| ELMo | 0.37 | 0.45 | 0.41 | 0.67 |
| BERT | 0.52 | 0.56 | 0.54 | 0.72 |
| ASTNN | [illegible] | [illegible] | [illegible] | [illegible] |
| TBCNN | 0.45 | 0.42 | 0.43 | 0.66 |
| TreeCaps | 0.51 | 0.47 | 0.49 | 0.69 |
| C. Tree-based | Precision | Recall | F-score | AUC |
| Node2vec | 0.28 | 0.30 | 0.29 | 0.61 |
| DeepWalk | 0.32 | 0.31 | 0.31 | 0.65 |
| Graph2vec | 0.47 | 0.44 | 0.45 | 0.71 |

Higher contextuality, higher performance in Bug Detection

# Empirical Evaluation

**(RQ1)** Which of the static or contextualized embeddings fit better with the mixed polysemy nature of source code?

What is the degree of contextuality for sequence-based, tree-based, and graph-based code representation learning models?

**(RQ2)** Impact of Contextuality on Bug Detection

**(RQ3)** Hybrid Code Representation Learning Model

# Empirical Evaluation (RQ3)

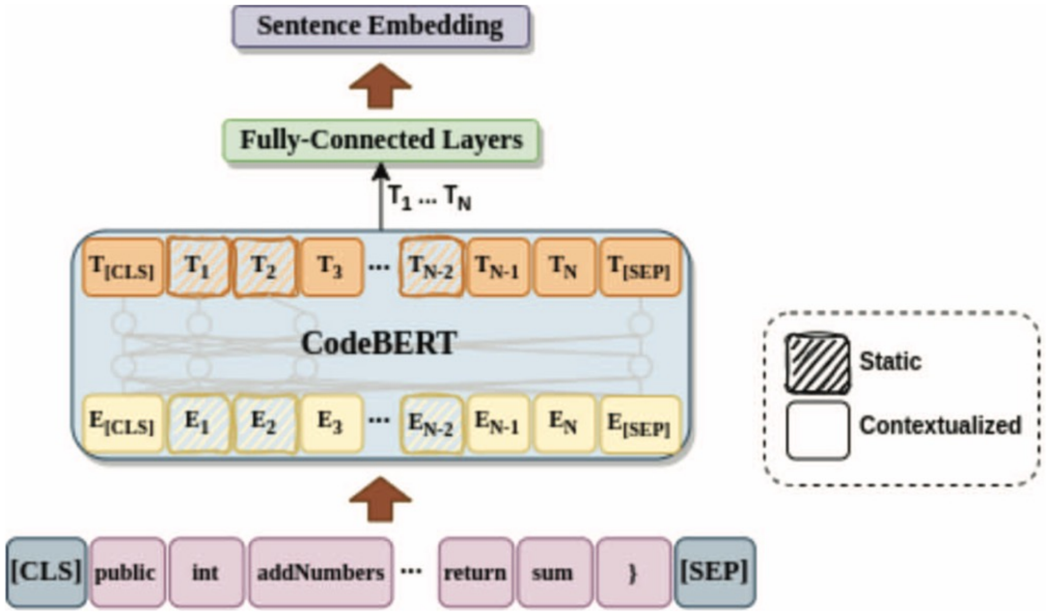



Fig. 2: HYCODE: Hybrid CRL Model

# Empirical Evaluation (RQ3)

TABLE XII: HyCode in Statement-level Bug Detection

| | Precision | Recall | F-score | AUC | Time (mins) |
|---|---|---|---|---|---|
| CodeBERT | 0.46 | 0.33 | 0.39 | 0.67 | 414 |
| CodeBERT-RM | 0.48 | 0.37 | 0.43 | 0.61 | 389 |
| BW1 | 0.48 | 0.35 | 0.40 | 0.69 | 272 |
| BW2 | 0.47 | 0.36 | 0.41 | 0.70 | 291 |
| HyCode | 0.67 | 0.63 | 0.65 | 0.80 | 335 |

TABLE XIII: HyCode in Method-level Bug Detection

| | Precision | Recall | F-score | AUC | Time (mins) |
|---|---|---|---|---|---|
| CodeBERT-RM | 0.55 | 0.49 | 0.52 | 0.68 | 395 |
| CodeBERT | 0.52 | 0.56 | 0.54 | 0.72 | 437 |
| BW1 | 0.54 | 0.58 | 0.56 | 0.73 | 285 |
| BW2 | 0.54 | 0.59 | 0.57 | 0.73 | 296 |
| HyCode | 0.76 | 0.64 | 0.68 | 0.80 | 385 |

# Empirical Evaluation (RQ3)

TABLE XII: HYCODE in Statement-level Bug Detection

| | Precision | Recall | F-score | AUC | Time (mins) |
|---|---|---|---|---|---|
| CodeBERT | 0.46 | 0.33 | 0.39 | 0.67 | 414 |
| CodeBERT-RM | 0.48 | 0.37 | 0.43 | 0.61 | 389 |
| BW1 | 0.48 | 0.35 | 0.40 | 0.69 | 272 |
| BW2 | 0.47 | 0.36 | 0.41 | 0.70 | 291 |
| HYCODE | 0.67 | 0.63 | 0.65 | 0.80 | 335 |

HYCODE achieves better F1-Score and AUC than all other baselines, while also saving on the running time.

| | Precision | Recall | F-score | AUC | Time (mins) |
|---|---|---|---|---|---|
| CodeBERT-RM | 0.55 | 0.49 | 0.52 | 0.68 | 395 |
| CodeBERT | 0.52 | 0.56 | 0.54 | 0.72 | 437 |
| BW1 | 0.54 | 0.58 | 0.56 | 0.73 | 285 |
| BW2 | 0.54 | 0.59 | 0.57 | 0.73 | 296 |
| HYCODE | 0.76 | 0.64 | 0.68 | 0.80 | 385 |

# Future Directions

1. Currently, this study is only limited to Java. It would be interesting to see how these results extend to other programming languages.

2. HYCODE is still very ad-hoc, and our results are encouraging to want to explore more fundamental ways for incorporating the static or contextual nature of (sub)-tokens in source code into the architecture design and pre-training process itself.

   The general idea is to enforce the staticization of keywords, separators, and operators; and contextualization of identifiers and literals.

3. We are open to discuss ideas and potentially collaborate ☺

# Thank you!



Scan QR code to access the replication package, data, and supplementary material.

# EXTRA SLIDES

# Insights for Sequence-Based Models

**Insight S1 in RQ1. CodeBERT :> BERT :> ELMo**

CodeBERT's embeddings are more contextualized than BERT's, which are more contextualized than ELMo's. Word2Vec, GloVe, and FastText produce static embeddings.

**Insight S2 in RQ1. Mismatch between Contextualized Embeddings and Non-polysemous Tokens**

While CodeBERT, BERT, and ELMo produce contextualized vectors for identifiers and literals as expected, they produce contextualized vectors for the static tokens including keywords, operators, and separators.

**Insight S3 in RQ1. Contextuality for Statements**

1) CodeBERT, BERT, and ELMo **vectors for statements** are more contextualized than **the tokens' vectors**, which are more contextualized than the **sub-tokens' vectors**.
2) CodeBERT, BERT, and ELMo have more distinctive vectors for **the tokens within a statement** than for **the statements within a method**.

# Insights for Tree-Based Models

**Insight T1 in RQ2. Contexuality of Tree-based Models for Tokens: TreeCaps :> TBCNN; Tree-LSTM, ASTNN: static**

For the AST-nodes of tokens, TreeCaps produces more contextualized vectors than TBCNN. Tree-LSTM and ASTNN produce static embeddings for all tokens.

**Insight T2 in RQ2. Contextuality of Tree-based Models for Stmts: TreeCaps :> TBCNN :> ASTNN; Tree-LSTM: static**

1) For AST-subtrees of statements, TreeCaps produces more contextualized vectors than TBCNN, while Tree-LSTM produces static embeddings.
2) Even though ASTNN is a static model for tokens, it can capture contexts for statements via its mechanism to build statement vectors. Thus, its vectors for AST-subtrees for statements are contextualized. This shows that **it is possible to build a contextualized tree-based models for statements on top of static vectors for tokens**.

**Insight T3 in RQ2. Tree-based Models ASTNN and TBCNN**

For the AST-nodes of tokens, despite that ASTNN's vectors are static, ASTNN produces the same level of distinct vectors for the units within a context as the contextualized model TBCNN.

# Insights for Graph-Based Models

| **Insight G1 in RQ3. Graph-based Models: Graph2Vec :> DeepWalk :> Node2Vec** |
| --- |
| Graph-based models capture structural contexts. Graph2vec, DeepWalk, Node2Vec give contextualized vectors in this order. |

# Insights for Impact of Contextuality on Bug Detection

**Insight B1 in RQ4. Impact of Contextuality of an CRL Model on Bug Detection**

The more contextualzed vectors a model produces, the higher accuracy a downstream bug detection model at both the statement and method levels can achive.

Sequence-based models: (CodeBERT :> BERT :> ELMo :> Word2Vec, GloVe, FastText (static))

Tree-based models: TreeCaps :> TBCNN :> ASTNN, Tree-LSTM (static)

Graph-based models: Graph2Vec :> DeepWalk :> Node2vec

# Insights for Hybrid Code Representation Model

| **Insight B1 in RQ5. The Hybrid Model, HyCode, and Bug Detection Performance** |
|---|
| 1. HyCode, **a hybrid code representation learning model** between contextualized and static ones **yields better bug detection performance** at both statement and method levels.<br>2. Simply combining the static model Word2Vec (running on context-insensitive tokens) and BERT (running on literals and identifiers) reduces the overall training time, while maintaining similar BD accuracy (BW1/BW2).<br>3. A model that discards the context-insensitive tokens does not perform as well as the hybrid model, HyCode. |