# ARiSE Lab
# Columbia University
# Computer Science

Research effort in Machine Learning for Source Code Analysis

# Do we really need AI/ML for code analysis?

```java
public Color[][] smooth(
        Color[][] image,
        int neighborhoodSize) {
    assert … … … ;
    … … …
}
```

```
MethodDeclaration
    Modifier public
        SimpleName smooth
    Parameter
        ArrayType
            ArrayType
                ClassOrInterfaceType
                    SimpleName  Color
        SimpleName image
    Parameter
        PrimitiveType int
        SimpleName neighborhoodSize
    ArrayType
        ArrayType
            ClassOrInterfaceType
                SimpleName Color
    BlockStmt
        AssertStmt
            … … …
        … … …
```

# Perhaps we DO need AI/ML in SE

```
Sort a List of Tuples by first element.
```

```java
static Tuple[] sortArray(Tuple[] uns){
   return Arrays.sort(
     uns, new Comparator<Tuple>() {
      public int compare(
       Tuple o1, Tuple o2) {
          return o1.get(0) == o2.get(0);
       }
   });
}
```

```python
def sort_list(uns):
    return sorted(uns, key=lambda x:x[0])
```

# Perhaps we DO need AI/ML in SE.

**Sort a List of Tuples by first element.**

Program Synthesis Task

```java
static Tuple[] sortArray(Tuple[] uns){
    return Arrays.sort(
      uns, new Comparator<Tuple>() {
       public int compare(
        Tuple o1, Tuple o2) {
            return o1.get(0) == o2.get(0);
        }
    });
}
```

```python
def sort_list(uns):
    return sorted(uns, key=lambda x:x[0])
```

# Perhaps we DO need AI/ML in SE.

**Sort a List of Tuples by first element.**

```java
static Tuple[] sortArray(Tuple[] uns){
    return Arrays.sort(
      uns, new Comparator<Tuple>() {
        public int compare(
          Tuple o1, Tuple o2) {
            return o1.get(0) == o2.get(0);
          }
    });
}
```

Code Translation

Task

```python
def sort_list(uns):
    return sorted(uns, key=lambda x:x[0])
```

# Perhaps we DO need AI/ML in SE.

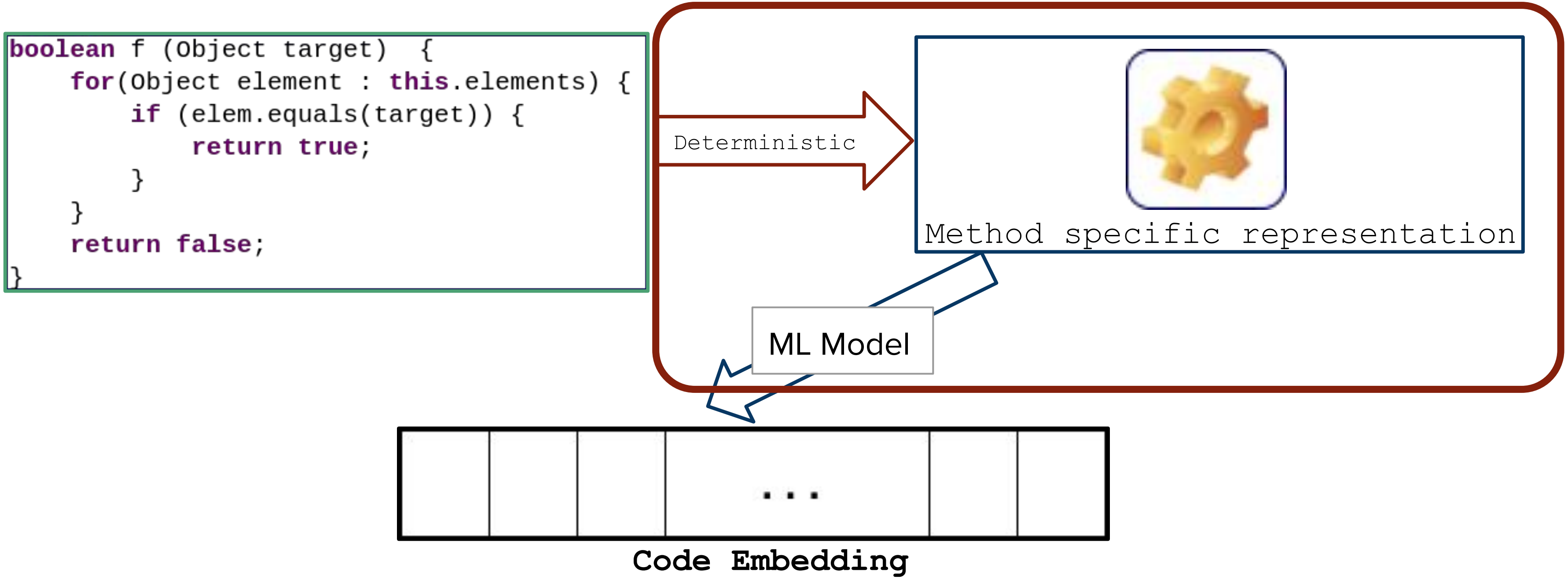
```
void action(char *data) const {
    for (; *data != '\0'; data++){
        foo(data);
        bar(data);
        if (*data == SEARCH_CHAR){
            ...
            break;
        }
    }
    free(data);
}
```

**Vulnerability Detection Task**

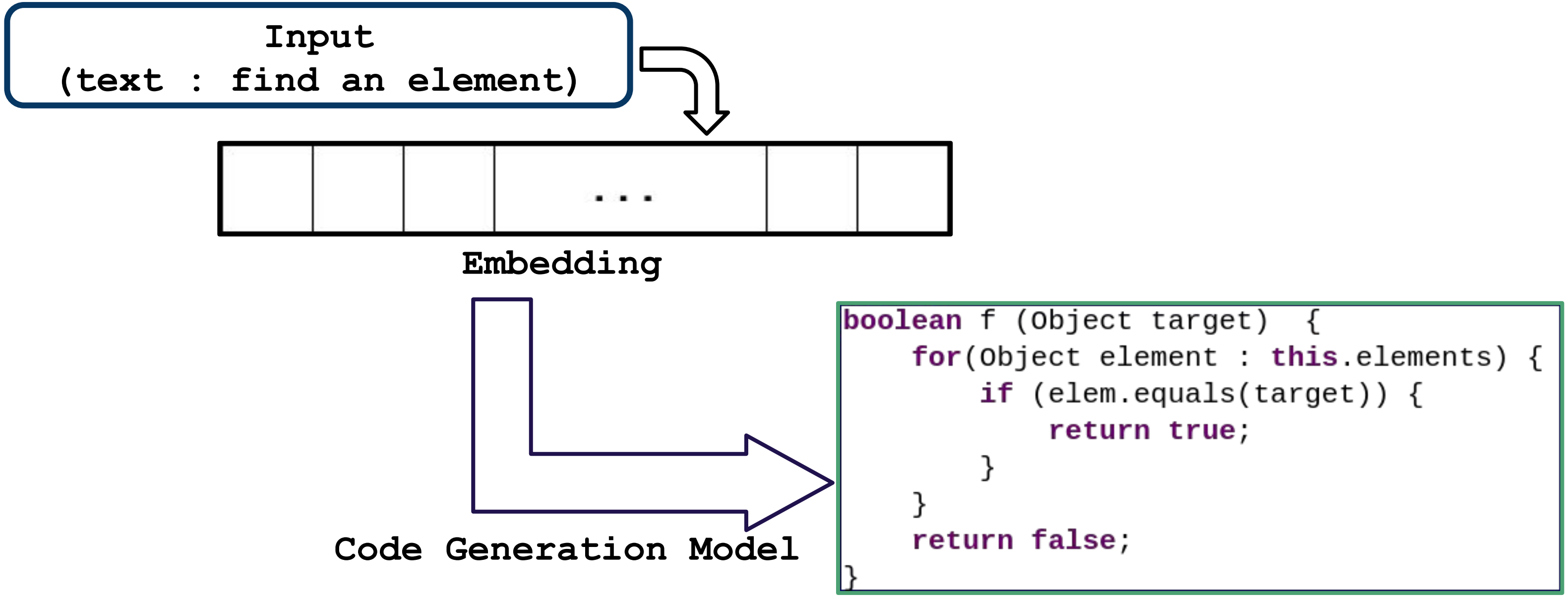
# Our Effort in AI for Source Code Analysis

1. Understanding Source Code
    a. Code Completion.
    b. Vulnerability Detection.
2. **Learning to Represent Source Code**
    a. **Code Comprehension/Summarization.**
    b. **Code generation.**
    c. **Code translation.**
3. **Learning to Edit Code**
    a. **Automated Code Change.**
    b. **Program Repair.**

# Source Code Representation (understanding)

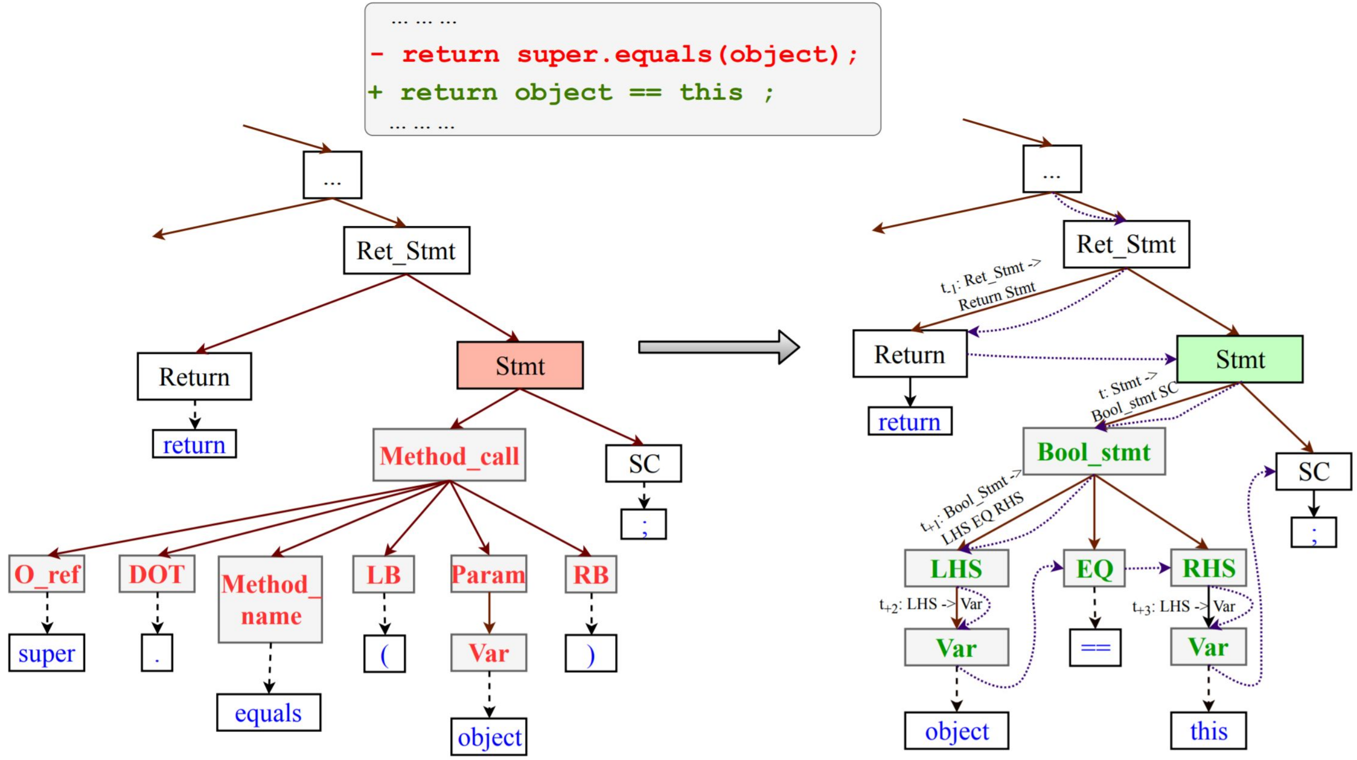


# Source Code Generation

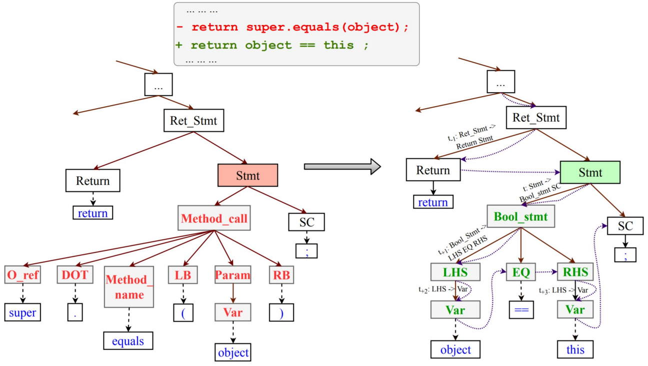


# Learning to Edit Code

# Application - 1 (Code Editing) (CODIT - TSE’20)

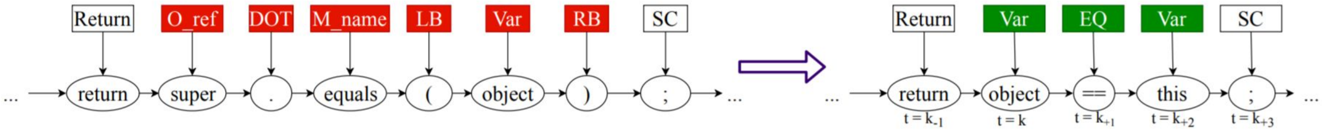


# CODIT(contd.)



- return super.equals(object);
+ return object == this ;
Ret_Stmt
Return
Stmt
return
Method_call
SC
O_ref
DOT
Method_name
LB
Param
RB
super
equals
Var
object
Bool_stmt
LHS
EQ
RHS
==
this


... ... ...
Ret_Stmt → Return Stmt
Stmt → M_Call SC
Method_Call → O_ref Dot Method_Name LB Param RB
Param → Var
... ... ...
... ... ...
Ret_Stmt → Return Stmt
Stmt → Bool_stmt SC
Bool_stmt → LHS EQ RHS
LHS → Var
RHS → Var
... ... ...

Return
O_ref
DOT
M_name
LB
Var
RB
SC
return
super
.
equals
(
object
)
;
Return
Var
EQ
Var
SC
return
object
==
this
;

# CODIT (contd.)

| Method | | Code Change Data | | | Pull Request Data | | |
|---|---|---|---|---|---|---|---|
| | | Number of examples : 5143 | | | Number of examples : 613 | | |
| | | Top-1 | Top-2 | Top-5 | Top-1 | Top-2 | Top-5 |
| **Token Based** | Seq2Seq | 107 (2.08%) | 149 (2.9%) | 194 (3.77%) | 45 (7.34%) | 55 (8.97%) | 69 (11.26%) |
| | Tufano *et al.* | 175 (3.40%) | 238 (4.63%)) | 338 (6.57%) | **81 (13.21%)** | 104 (16.97%) | 145 (23.65%) |
| | SequenceR | **282 (5.48%)** | 398 (7.74%) | 502 (9.76%) | 39 (6.36%) | **137 (22.35%)** | 162 (26.43%) |
| **Tree Based** | Tree2Seq | 147 (2.86%) | 355 (6.9%) | 568 (11.04%) | 39 (6.36%) | 89 (14.52%) | 144 (23.49%) |
| | Code2seq | 58 (1.12%) | 82 (1.59%) | 117 (2.27%) | 4 (0.65%) | 7 (1.14%) | 10 (1.63%) |
| | CODIT | 201 (3.91%) | **571 (11.10%)** | **820 (15.94%)** | 57 (9.3%) | 134 (21.86%) | **177 (28.87%)** |
| **IR based** | $\mathscr{B}_{ir}$ | 40 (0.77%) | 49 (0.95%) | 61 (1.18%) | 8 (1.30%) | 8 (1.30%) | 9 (1.46%) |

# CODIT - Examples

```
void visit(JSession * session , ...) throws Exception
{
    visit (((JNode) (* session)), ...);
}
```

```
public abstract void removeSessionCookies (...)
 {
     throw new android...MustOverrideException();
 }
```

```
public void copyFrom( java.lang.Object arr){
+      try{
        android.os.Trace.traceBegin (...);
+      finally{
        android.os.Trace.traceEnd(...);
+      }
}
```

# CODIT - Takeaway

1. Neural Machine Translation is really useful for learning code change patterns.
2. Tree can be generated by sampling from CFG.
3. A tree is syntactically correct.
4. CODIT builds tree instead of code.

# Example of Invalid code.

Syntactically Incorrect

```
boolean f (Object target)  {
    for(Object element : if.elements) {
        break (elem.equals(target)) {
            return continue;
        }
    }
    return false;
}
```

Contextually Incorrect

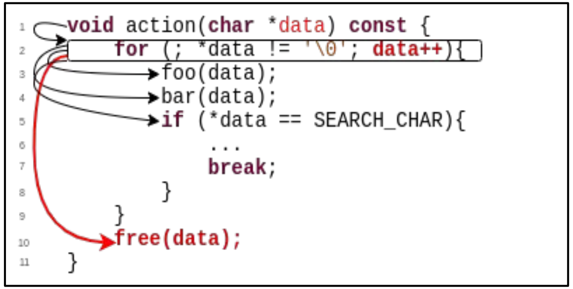
```
boolean f (Object target)  {
    for(Object elem : this.elements) {
        if (elem.equals(f)) {
            return null;
        }
    }
    return false;
}
```

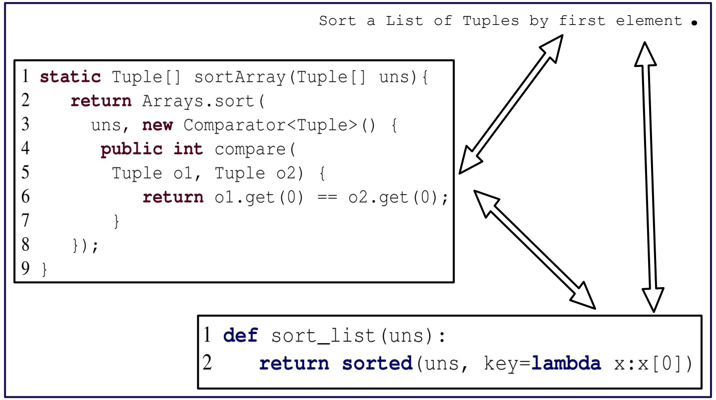
Stylistic Incorrect

```
void __write_to_file (
  FILE *_fp_, char* data){
    _fp_->write(data);
    fclose(fp);
}
```

# Learning to Represent Source Code

# Some Interesting Points to note

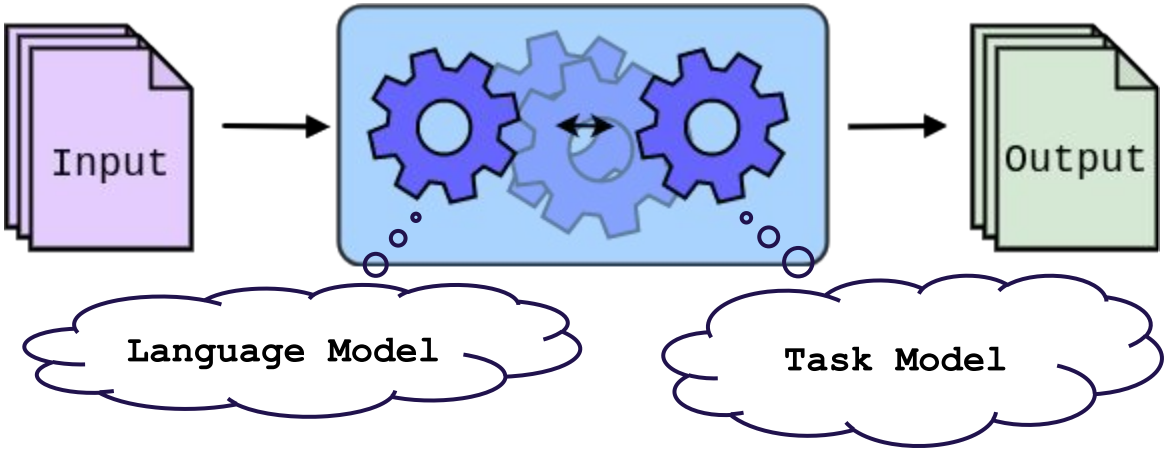



**Learning about the “Language”**

**Learning about the “Task”**

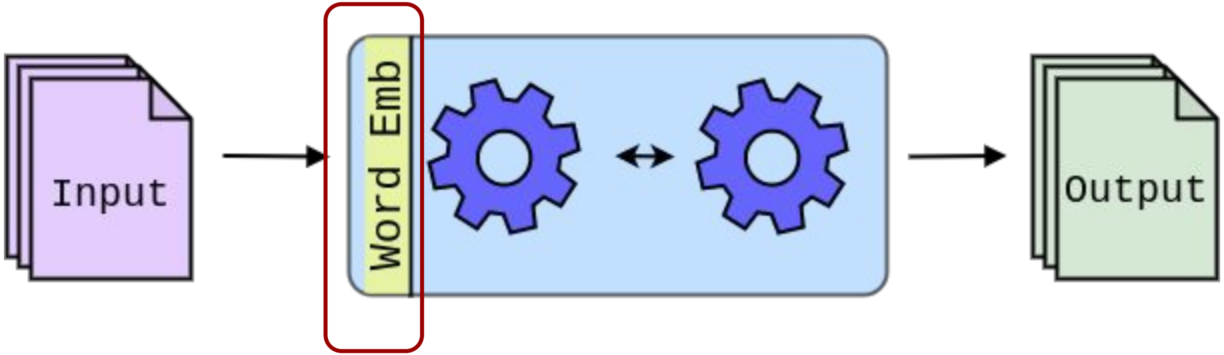
# Some Interesting points to note



# Can we lessen the burden for model?

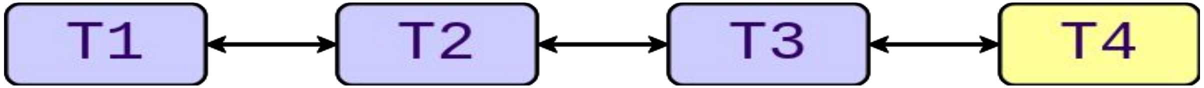
Can we transfer any knowledge from elsewhere?

1. Word2Vec in code (used by VulDeePecker, SySeVR, Devign) can be a way.
2. Code2Vec; another way.

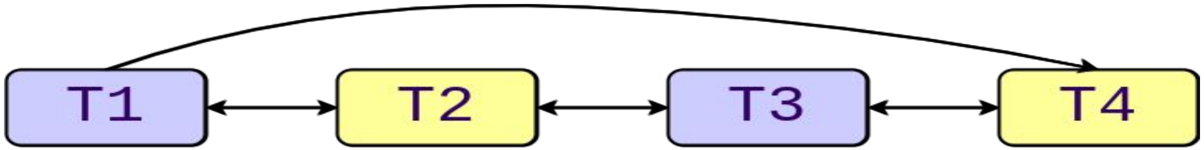


# Related topic - Different Models

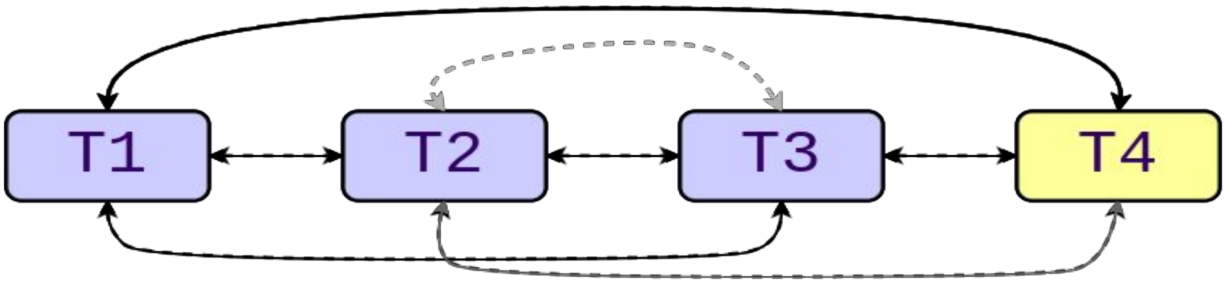
1. Sequence Based Models




2. Graph Based Models

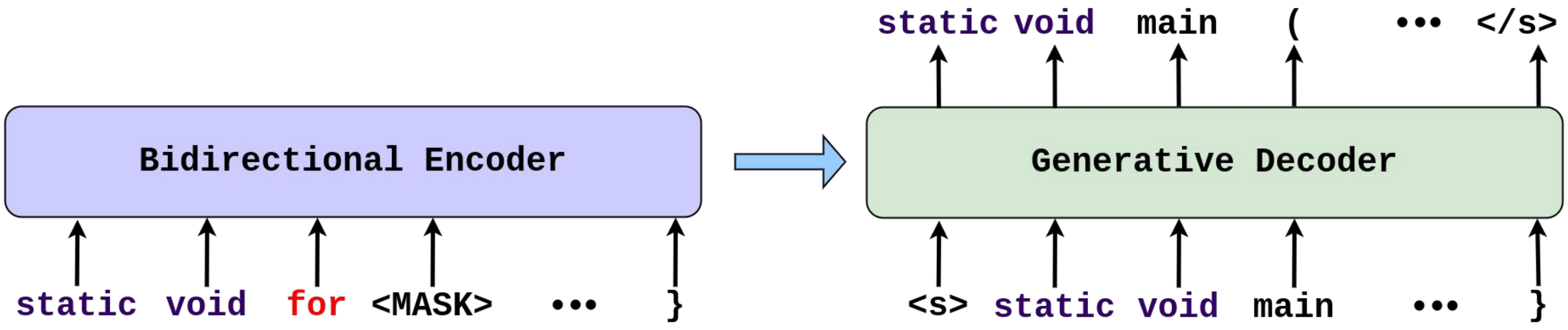


# Related topic - Transformer




1. Implicitly learns non-linear structure in the input data.
2. Often very large/deep models with very high capabilities.
3. Learns the syntactic and semantic relationship very well.

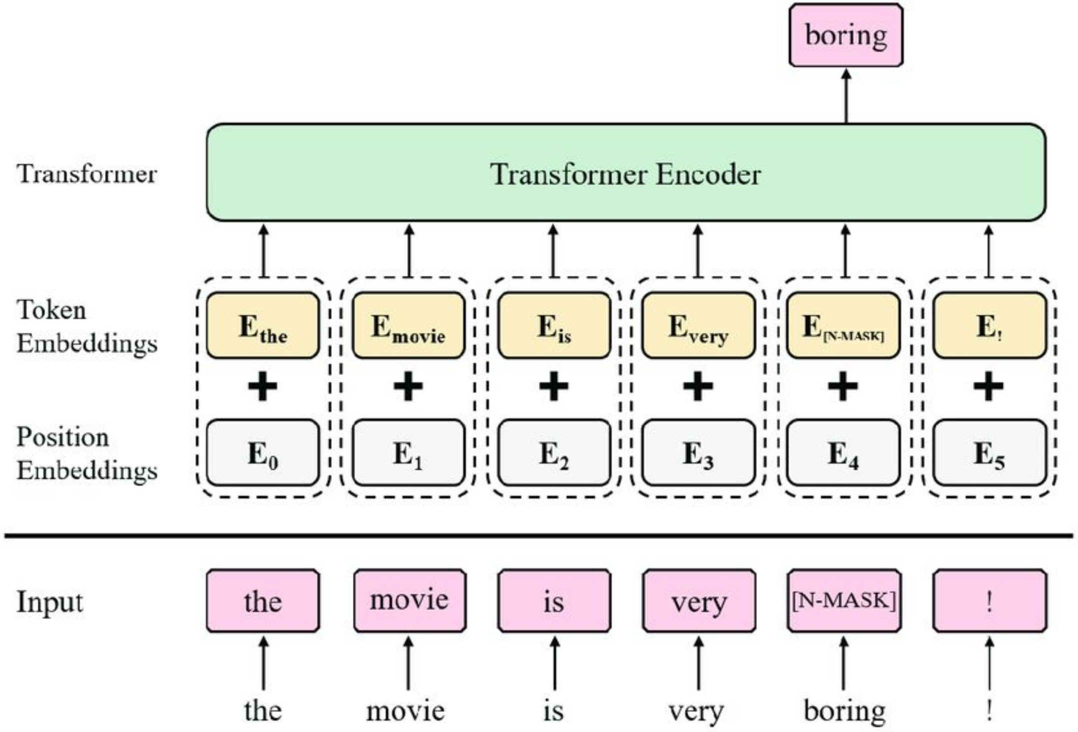
# PLBART - NAACL'21




PLBART:

1. Trained on 470M Java code, 210M Python Code, 47M Stackoverflow posts.
2. Multiple languages - for **pre-training** one model for different SE tasks.

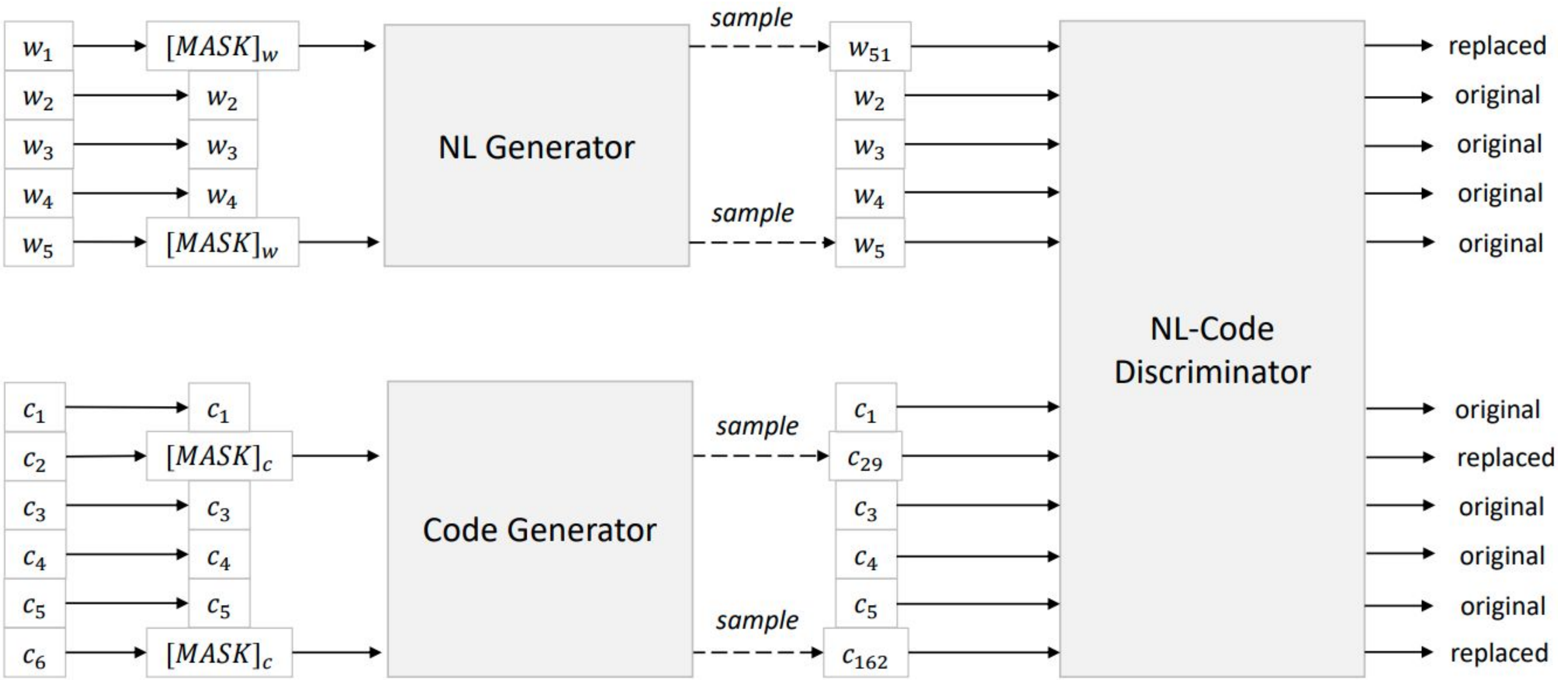
# Existing Approach - BERT



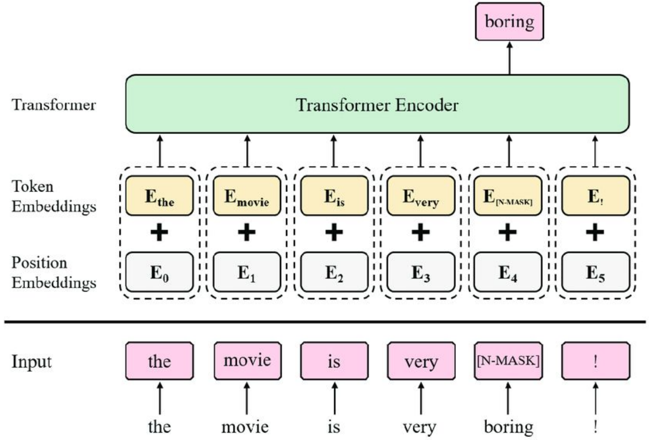

**Pre-training:**
Task agnostic Masked Language Model.

**Fine Tuning:**
Task Specific Objective.

# CodeBERT - BERT for Code



# BERT - Any Problem?




1. Just a Transformer encoder.
2. Works very well for Understanding/Descriminative tasks.
3. Must be accompanied with a decoder (trained from scratch during fine-tuning).
4. Decoder itself may demand high volume of data.

# PLBART - Take Away points.

1. Transformer (specially very large architectured) are powerful enough to learn syntactic and contextually correct code.
2. Pre-training a model to understand and generate code simultaneously enables the model to learn the language as well as engineering constructs.
3. Denoising task in PLBART enables the model to understand the code syntax and semantics.
4. Auto-encoding task in PLBART enables it to correctly generate code.

# Some Interesting examples of PLBART

Example 1 : ✔

Input Code : C#

```
public int GetCells(){
    int size = 0;
    foreach (char c in cells.Keys){
        Cell e = At(c);
        if (e.cmd >= 0 || e.@ref >= 0){
            size++;
        }
    }
    return size;
}
```

Generated Code : Java

```
public int getCells() {
    Iterator<Character> i =
      cells.keySet().iterator();
    int size = 0;
    for (; i.hasNext();) {
      Character c = i.next();
        Cell e = at(c);
        if (e.cmd >= 0 || e.ref >= 0){
            size++;
        }
    }
    return size;
}
```

# Some Interesting examples of PLBART

Input Code : Java

```java
public
  PersianNormalizationFilterFactory(
      Map<String,String> args) {
    super(args);
    if (!args.isEmpty()) {
      throw
        new IllegalArgumentException(
          "Unknown parameters: "
          + args);
    }
}
```

Generated Code : C#

```csharp
public
  PersianNormalizationFilterFactory(
    IDictionary<string, string> args)
        : base(args){
    if (args.Count > 0){
      throw new System.ArgumentException(
        "Unknown parameters: "
        + args
      );
    }
}
```

# Some Interesting examples of PLBART

**Input :** Returns the count to which the specified key is mapped in this frequency counter , or 0 if the map contains no mapping for this key .

<u>Reference Code</u>

```
Integer function (T arg0) {
    Integer loc0 = counter.get(arg0);
    if (loc0 == null) {
        return 0 ;
    }
    return loc0;
}
```

<u>Generated Code</u>

```
int function (T arg0) {
    Integer loc0 = counter.get(arg0);
    if (loc0 == null) {
        return 0 ;
    }
    else {
        return loc0;
    }
}
```

# Some Interesting results from PLBART (generative)

| Methods | Ruby | Javascript | Go | Python | Java | PHP | Overall |
|---|---|---|---|---|---|---|---|
| Seq2Seq | 9.64 | 10.21 | 13.98 | 15.93 | 15.09 | 21.08 | 14.32 |
| Transformer | 11.18 | 11.59 | 16.38 | 15.81 | 16.26 | 22.12 | 15.56 |
| RoBERTa | 11.17 | 11.90 | 17.72 | 18.14 | 16.47 | 24.02 | 16.57 |
| CodeBERT | 12.16 | 14.90 | 18.07 | 19.06 | 17.65 | **25.16** | 17.83 |
| PLBART | **14.11** | **15.56** | **18.91** | **19.30** | **18.45** | 23.58 | **18.32** |

Code Summarization

Code Synthesis

| Methods | EM | BLEU | CodeBLEU |
|---|---|---|---|
| Seq2Seq | 3.05 | 21.31 | 17.61 |
| Guo et al. (2019) | 10.05 | 24.40 | 20.99 |
| Iyer et al. (2019) | 12.20 | 26.60 | - |
| GPT-2 | 17.35 | 25.37 | 22.79 |
| CodeGPT-2 | 18.25 | 28.69 | 25.69 |
| CodeGPT-adapted | **20.10** | 32.79 | 27.74 |
| PLBART | 18.75 | **36.69** | **38.52** |
| $\text{PLBART}_{10K}$ | 17.25 | 31.40 | 33.32 |
| $\text{PLBART}_{20K}$ | 18.45 | 34.00 | 35.75 |
| $\text{PLBART}_{50K}$ | 17.70 | 35.02 | 37.11 |

Code Translation

| Methods | Java to C# | | | C# to Java | | |
|---|---|---|---|---|---|---|
| | BLEU | EM | CodeBLEU | BLEU | EM | CodeBLEU |
| Naive Copy | 18.54 | 0 | 34.20 | 18.69 | 0 | 43.04 |
| PBSMT | 43.53 | 12.50 | 42.71 | 40.06 | 16.10 | 43.48 |
| Transformer | 55.84 | 33.00 | 63.74 | 50.47 | 37.90 | 61.59 |
| RoBERTa (code) | 77.46 | 56.10 | 83.07 | 71.99 | 57.90 | 80.18 |
| CodeBERT | 79.92 | 59.00 | 85.10 | 72.14 | 58.80 | 79.41 |
| GraphCodeBERT | 80.58 | 59.40 | - | 72.64 | 58.80 | - |
| PLBART | **83.02** | **64.60** | **87.92** | **78.35** | **65.00** | **85.27** |

# Some Interesting results from PLBART (understanding)

| Tasks | Vulnerability Detection | Clone Detection |
|---|---|---|
| Transformer | 61.64 | - |
| CodeBERT | 62.08 | 96.5 |
| GraphCodeBERT | - | 97.1 |
| PLBART | **63.18** | **97.2** |

# How things are done in literature (Encoding)

1. Sequence of tokens

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

```
boolean f ( Object target ) { for (Object element ... return false ; }
```

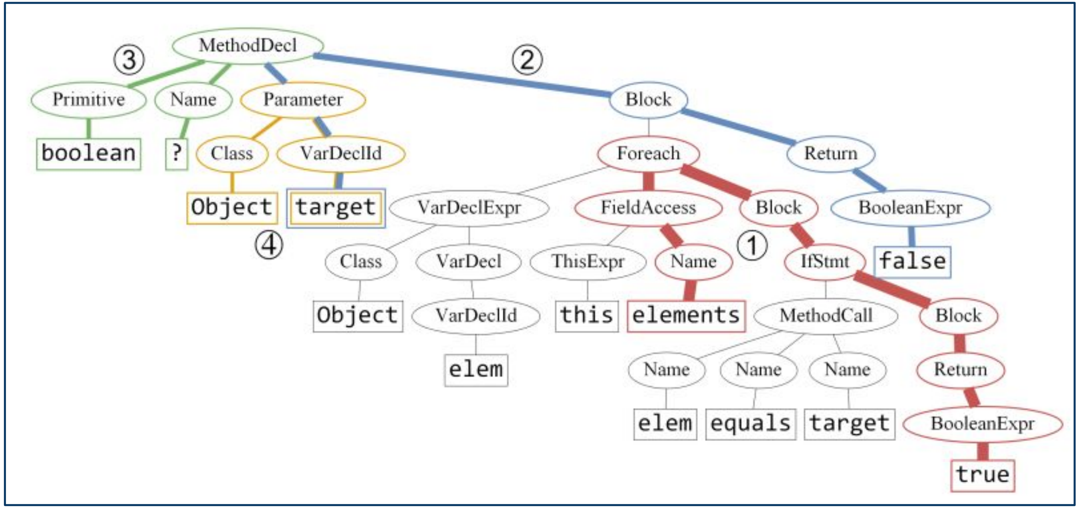
Russell et. al. `boolean ID ( ID ID ) { for (ID ID ... return false ; }`

**Used models : RNN, LSTM, CNN, etc.**

# How things are done in literature (Encoding)

2. AST

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

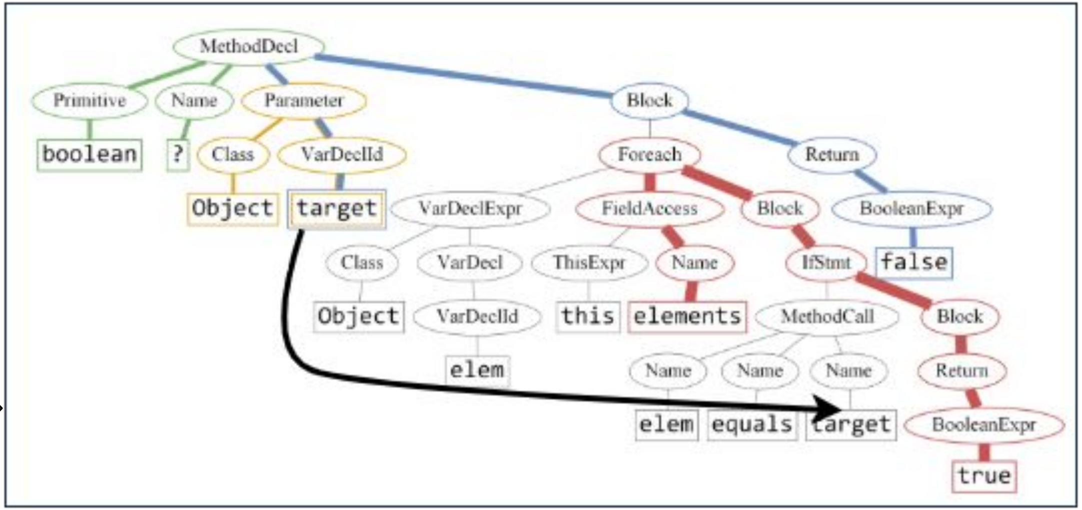



**`Used models : ASTNN (Zhang et. al.), Hierarchical RNN (Code2Vec)`**

# How things are done in literature (Encoding)

3. Graph

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



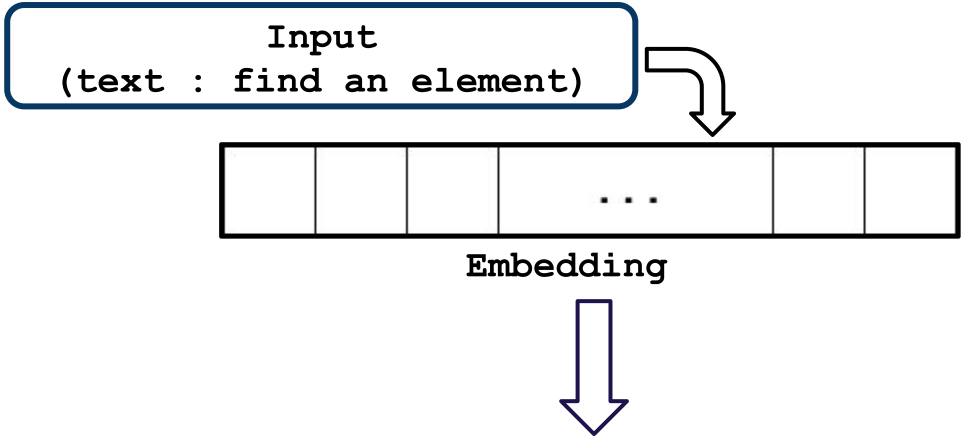

**Used models : Gated Graph Neural Network (Allamanis et. al., Devign)**

# Pros. and Cons. (Encoding)

| | Sequence | Tree | Graph |
|---|---|---|---|
| Pros | - Faster and Simpler methods. | - Capture syntax.<br>- Can reason about the syntactic dependencies. | - Captures both syntax and semantic dependencies.<br>- Good for reasoning about semantic relationship between tokens. |
| Cons | - Not merely a sequence of tokens.<br>- Lacks Syntax info.<br>- Lacks Semantic info. | - Slightly more complicated models.<br>- Still lack the semantic dependencies (data flow). | - Very complex models.<br>- Sometimes the yield is not so much worth the complexity. |

# How things are done in literature (Generation)

1. Sequence based generation

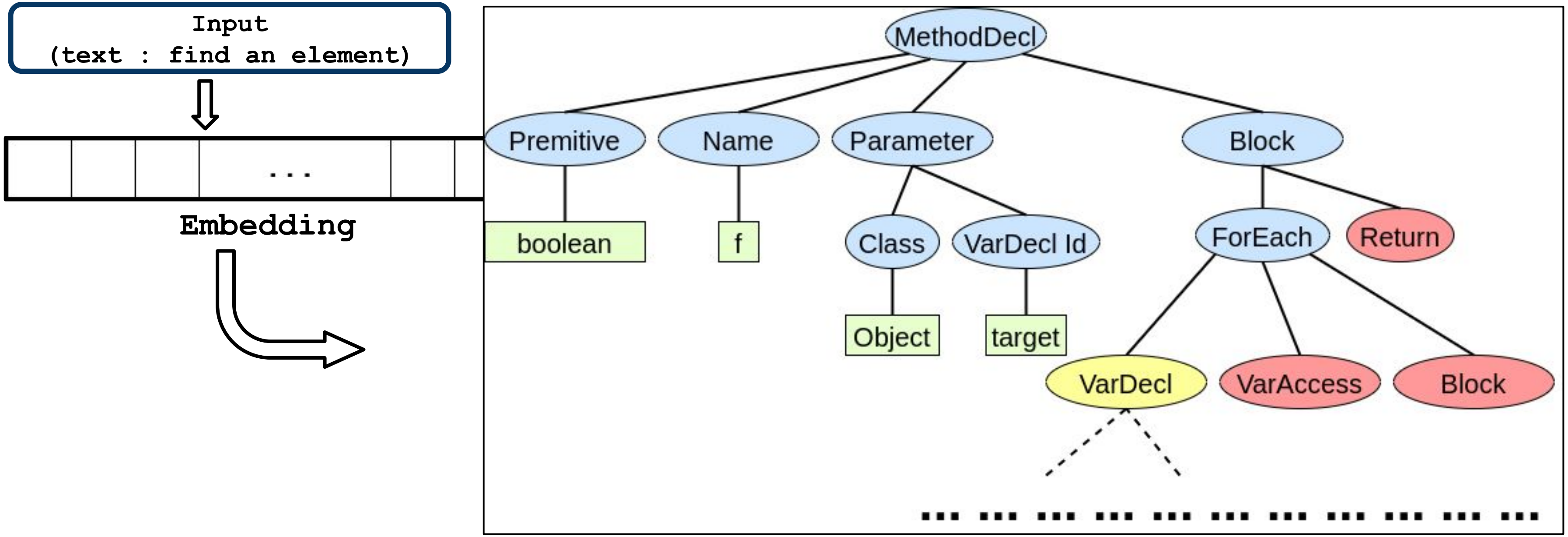



```
boolean f ( Object target ) { for (Object element ... return false ; }
```

**Used models : RNN, GRU, LSTM (all with beam search)**

# How things are done in literature (Generation)

1. Tree/Grammar based generation



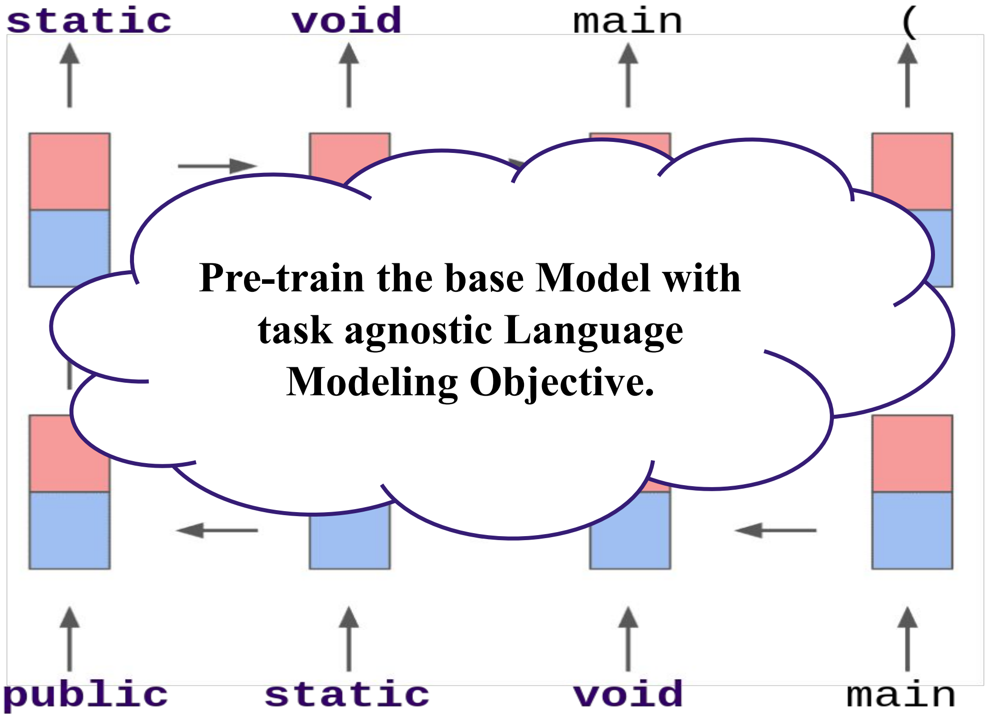
# Pros. and Cons. (Decoding/Generation)

| | Sequence | Tree |
|---|---|---|
| Pros | - Easier to implement.<br>- Off the shelf models can be used directly. | - Generates Syntactically correct code.<br>- Easier when the goal is to generate template rather the the full code. |
| Cons | - May generate syntactically invalid code.<br>- Might also create semantically wrong code. | - More complex models.<br>- Often difficult to model because of the large grammar.<br>- Modeling tokens/identifiers still remains a challenge<br>- Semantic correctness is still not guaranteed. |

# What are the challenges in joint learning?

1. Most of the task needs annotation/objective to update the model.
2. Demand for data increases with the complexity of the task.
3. Data is highly demanded by more complex models.

# Task agnostic **“Pre-Training”** (ELMo)



# ELMo (pros and cons)

- Pros:
    - Reduces burden on learning task specific reasoning.
- Cons:
    - Uses (Bidirectional)LSTM as base model.
    - Cannot capture the non-linear language constructs in code.

- Prospective Solution :
    - Pretrain tree of graph based models.

# Take Away Points

1. Machine learning in source code analysis showed a lot of promise over the years.
2. Source code exhibit different information through different input modalities, such as identifier names, syntax, semantic interaction between identifiers.
3. A good model for a particular task should exploit appropriate information modality.
4. Code synthesis is fundamentally different and more challenging than code understanding.
5. Annotated data scarcity can be overcome by unsupervised pre-training of a model.
6. A pretrained model should contain multiple modality (implicit/explicit), since pre-training is very expensive.

# Questions?

# Thanks!