Multi-Modal Program Representation Learning

Candidacy Exam Presentation

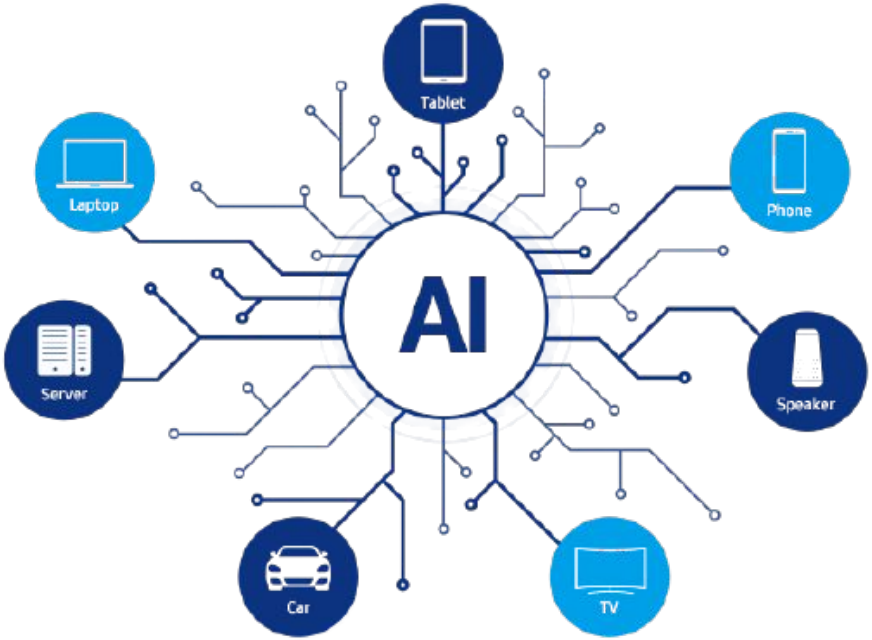
Saikat Chakraborty (sc4537)

Committee:

Dr. Baishakhi Ray Dr. Junfeng Yang Dr. Ronghui Gu

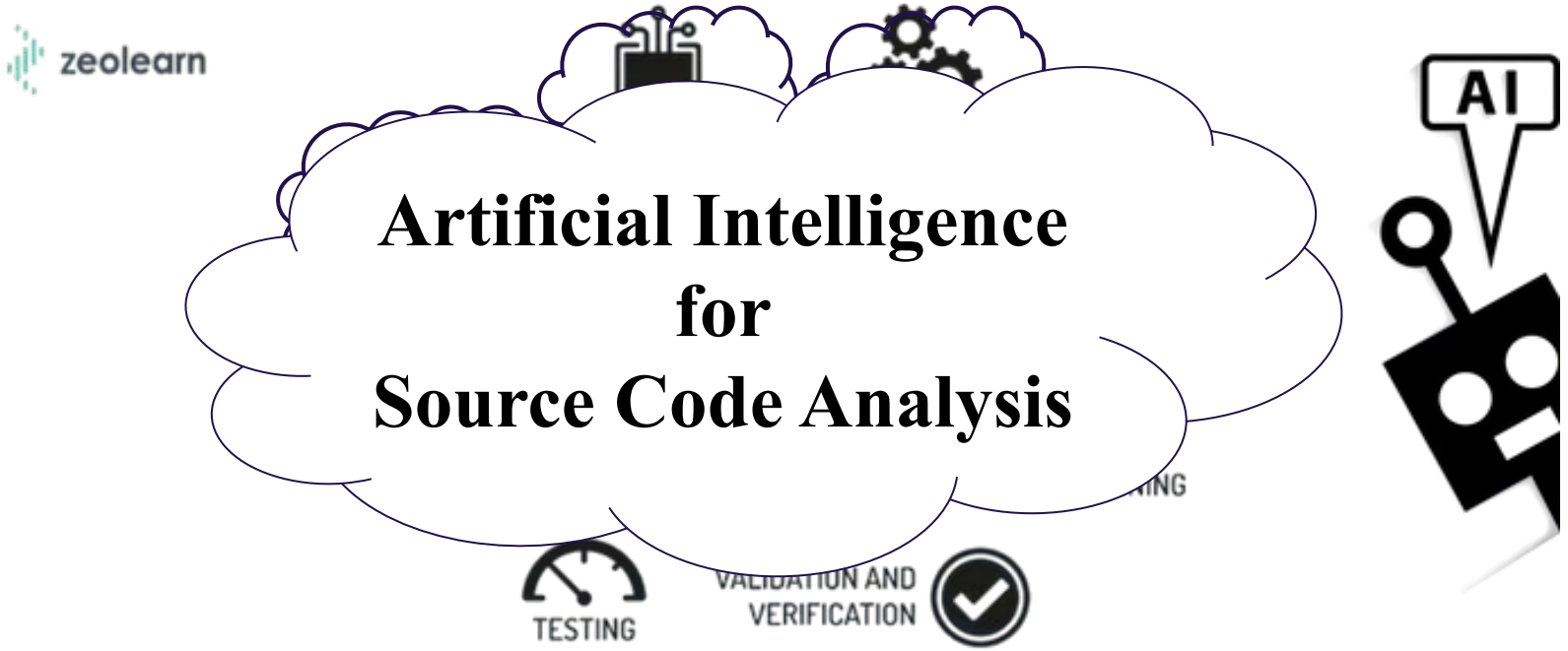
Department of Computer Science
Columbia University
New York, NY

Artificial Intelligence - Everywhere



[Image taken from samsung.com]

Artificial Intelligence for Software Engineers



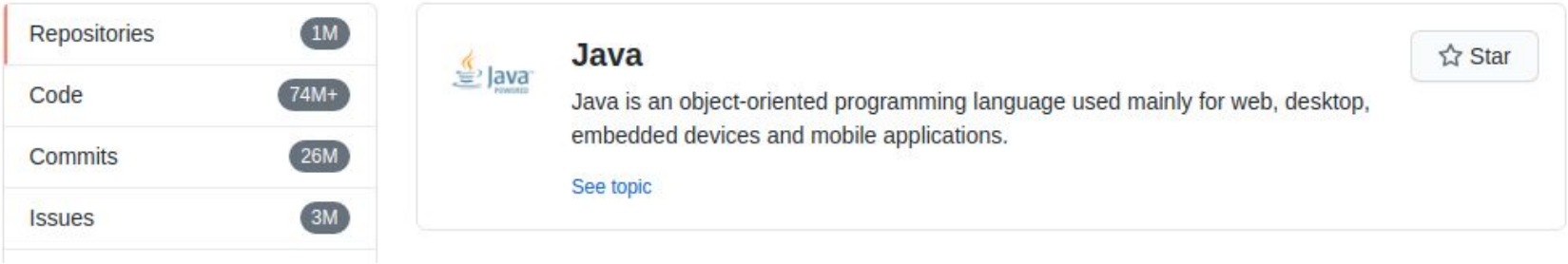
[Image taken from zeolearn]

Naturalness of Software (Hindle et. al. ICSE'12)

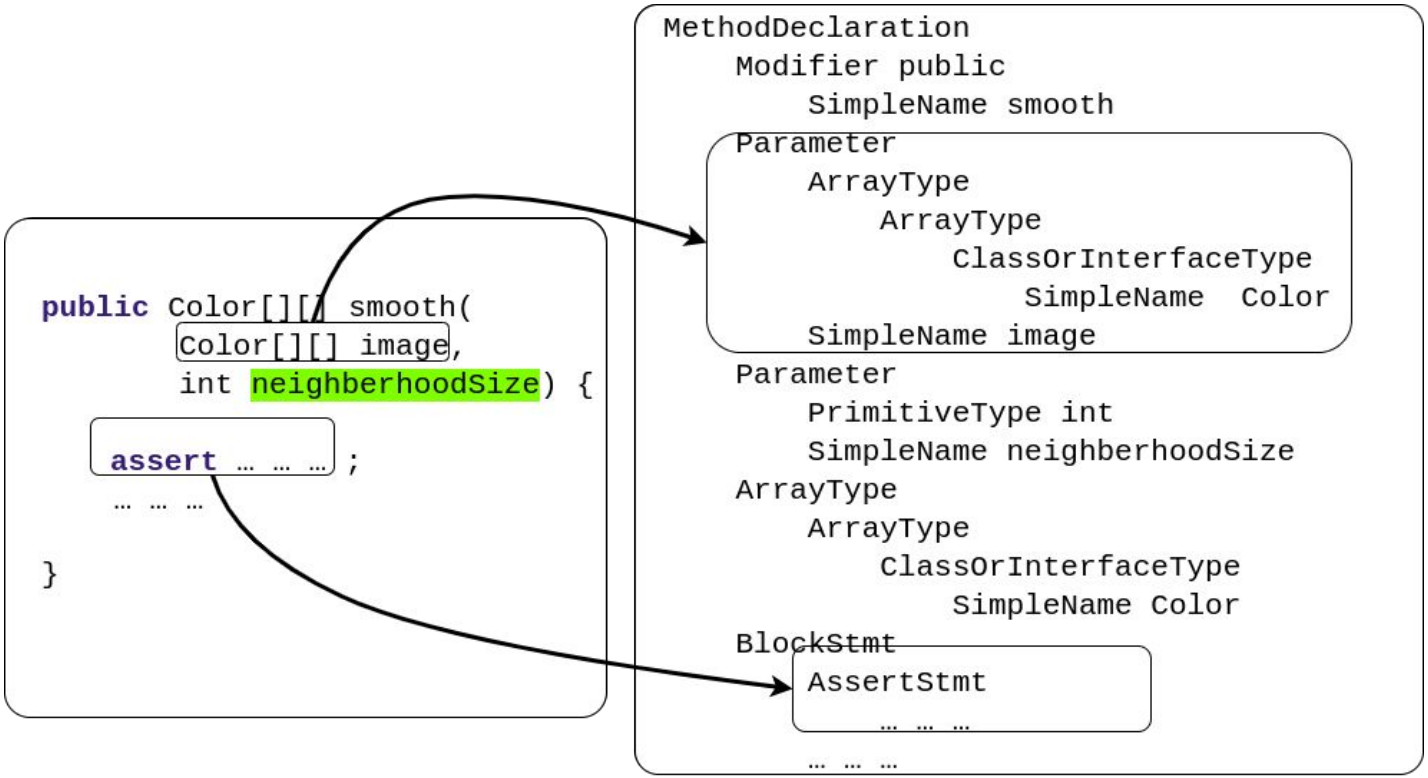
Programming languages, in theory, are complex, flexible and powerful, but the programs that real people actually write are mostly simple and rather repetitive, and thus they have usefully predictable statistical properties that can be captured in statistical language models and leveraged for software engineering t

```
void foo writeToFile(
  char *path, char *data){
    FILE *fp = fopen(path, "w");
    fp->write(data);
    // Unrelated work
    {
        // Do some unrelated work here.
    }
    fclose(fp);
}
```

Why AI/ML works in Software Engineering



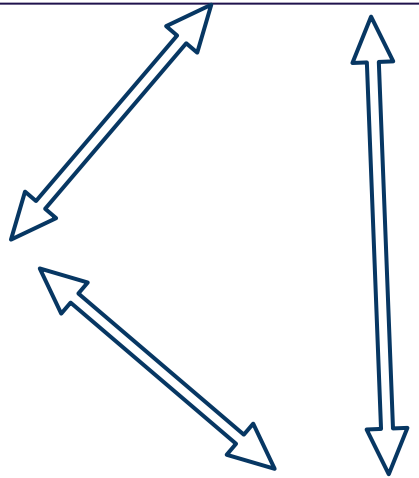
Do we really need AI/ML for code analysis?



Perhaps we DO need AI/ML in SE

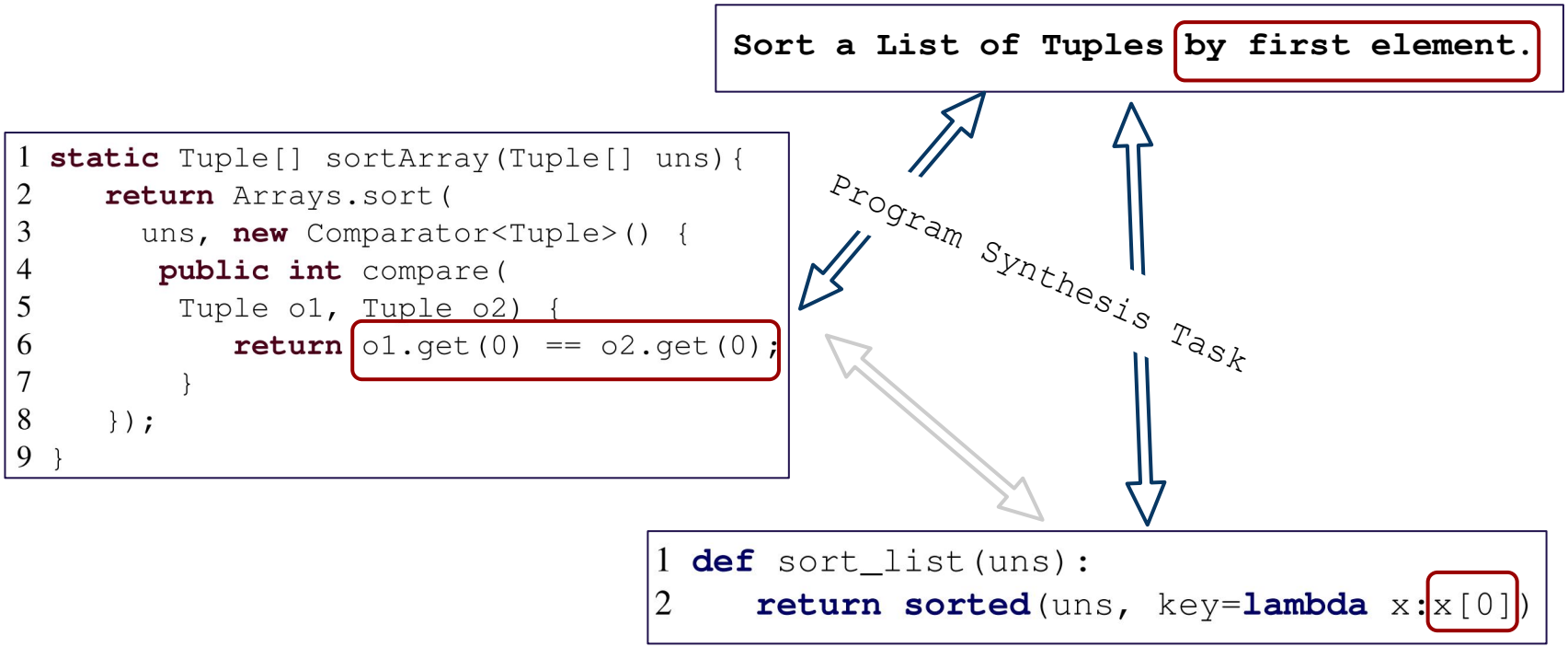
Sort a List of Tuples by first element.

```
1 static Tuple[] sortArray(Tuple[] uns){
2    return Arrays.sort(
3      uns, new Comparator<Tuple>() {
4       public int compare(
5        Tuple o1, Tuple o2) {
6           return o1.get(0) == o2.get(0);
7        }
8    });
9 }
```



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

Perhaps we DO need AI/ML in SE.



Perhaps we DO need AI/ML in SE.

Sort a List of Tuples by first element.

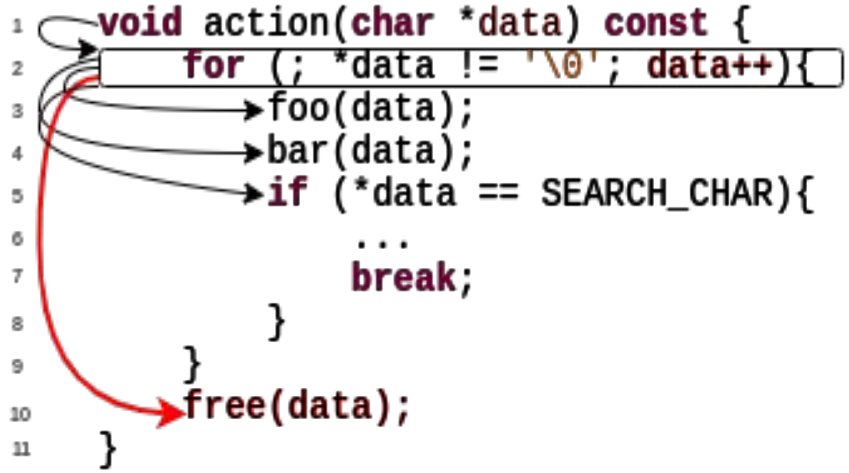
```
1 static Tuple[] sortArray(Tuple[] uns){
2    return Arrays.sort(
3      uns, new Comparator<Tuple>() {
4       public int compare(
5        Tuple o1, Tuple o2) {
6           return o1.get(0) == o2.get(0);
7        }
8    });
9 }
```

Code Translation

Task

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

Perhaps we DO need AI/ML in SE.



Vulnerability Detection Task

Application of AI in SE

1. Program Understanding (Discriminative)
    a. Vulnerability Detection (e.g. VulDeePecker, SySeVR, Russell et. al.)
    b. Clone Detection (e.g. Yu et. al.)
    c. Property Prediction (e.g. Code2Vec)
2. Program Generative Tasks
    a. Code Summarization (e.g. Ahmad et. al., Allamanis et. al., Code2Seq)
    b. Code Generation (e.g. Yin et. al., Sun et. al.)
    c. Code Translation (e.g. Codit, SequenceR, Drissi et. al., Chen et. al.)
3. Other Applications
    a. Code Completion (Hellendroon et. al., Parvez et. al.)

Challenge in Application of AI in SE

1. Program Understanding (Discriminative) Tasks
    a. Understand the program.
    b. Understand the non-linear relationship between tokens in code.
        i. Syntactic dependencies
        ii. Semantic dependencies
2. Generative Tasks
    a. Syntactic correctness guarantee.
    b. Semantic correctness guarantee.
    c. Stylistic correctness guarantee.

Example of Invalid code.

Syntactically Incorrect

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

Semantically Incorrect

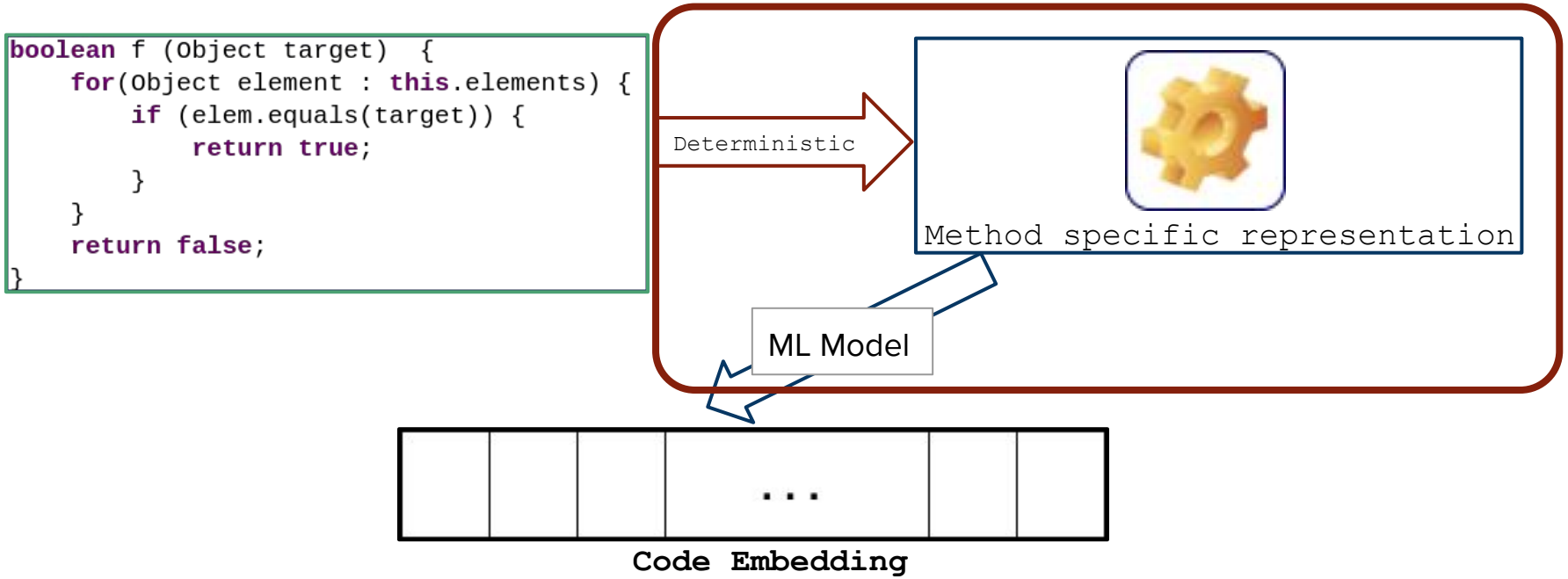
```
boolean f (Object target)  {
    for(Object elem : this.elements) {
        if (elem.equals(f)) {
            return null;
        }
    }
    return false;
}
```

Stylistic Incorrect

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

How things are done in literature (Encoding)

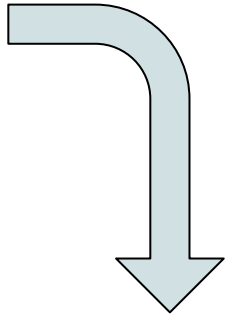
1. Program Understanding (Discriminative) Tasks



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.

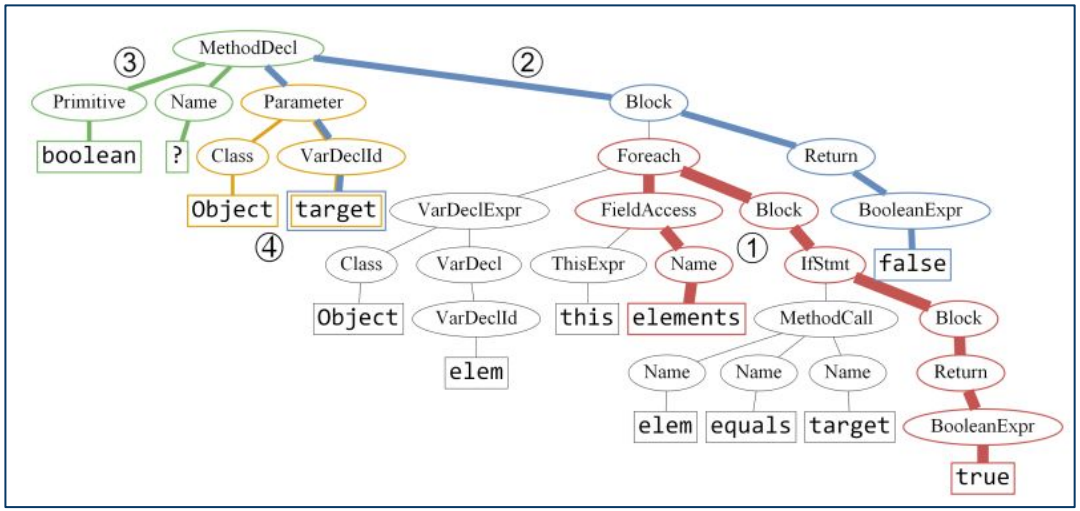
```
boolean ID ( ID ID ) { for (ID ID ... return false ; }
```

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;
}
```

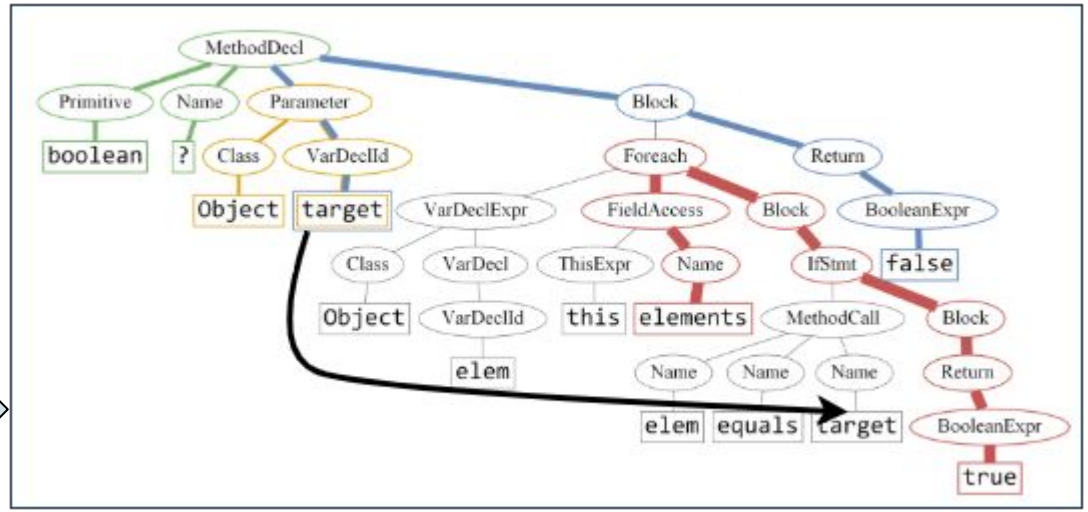


Used models : ASTNN (Zhang et. al.), Hierarchical RNN (Code2Vec)

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;
}
```



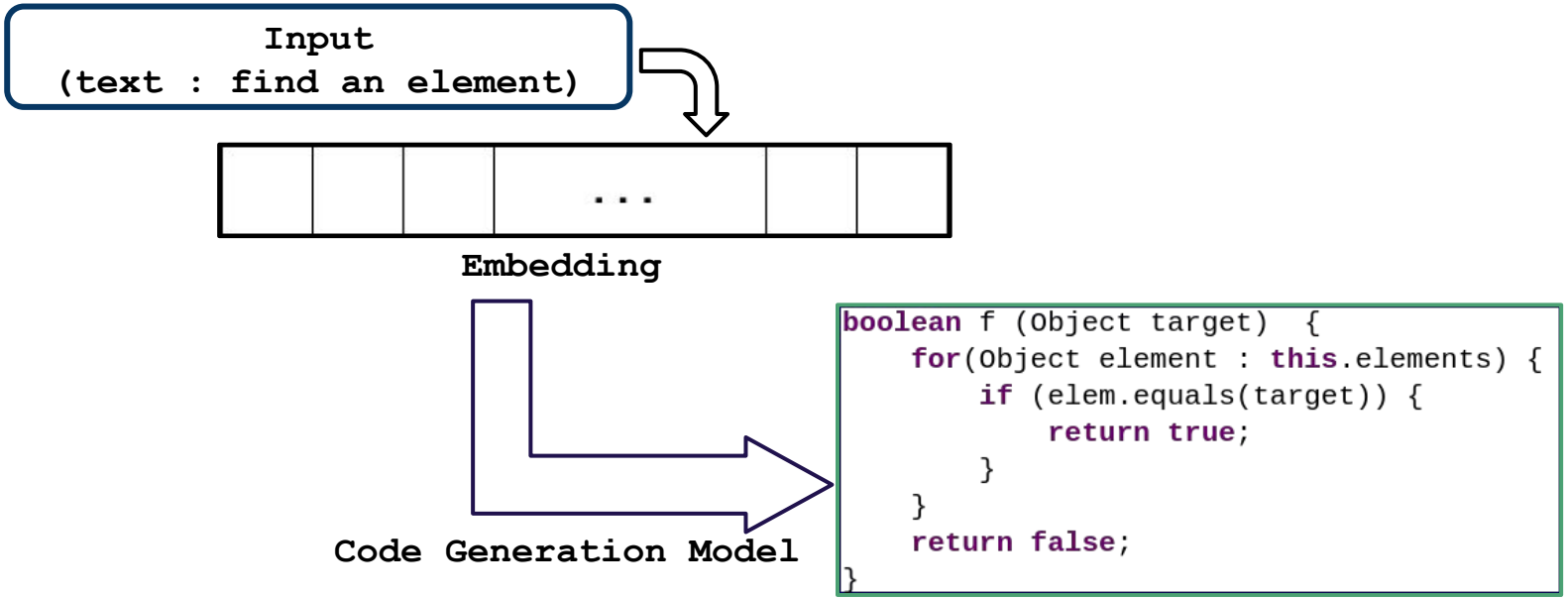
Used models : Gated Graph Neural Network (Allamanis et. al., Devign)

Pros. and Cons. (Encoding)

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

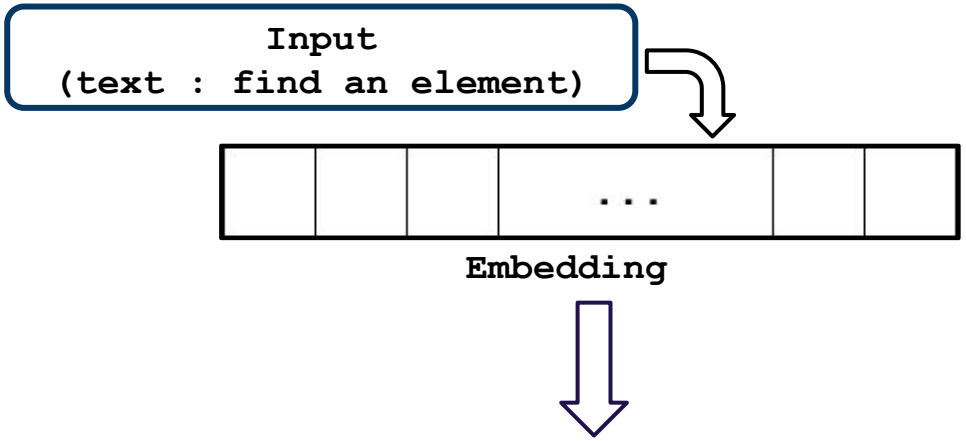
How things are done in literature (Generation)

1. Program Synthesis (Generative) Tasks



How things are done in literature (Generation)

1. Sequence based generation

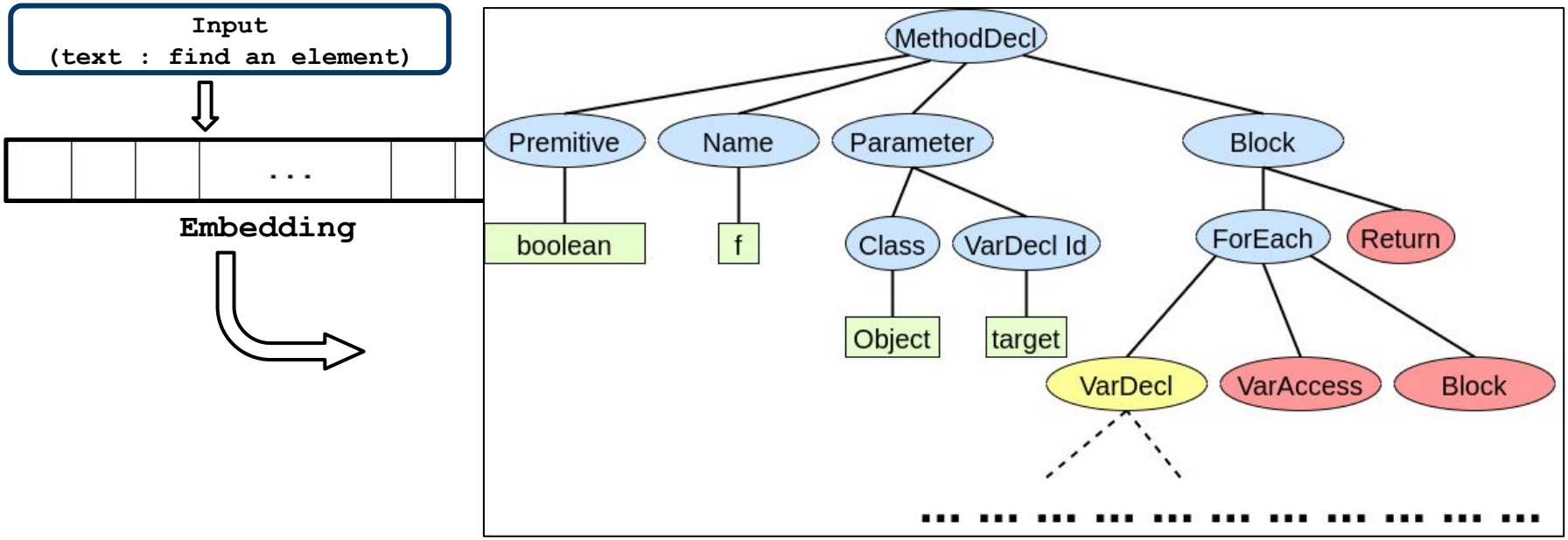



```
boolean f ( Object target ) { for (Object element ... return false ; }
```

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

Some Interesting Points to note

```
1  void action(char *data) const {
2      for (; *data != '\0'; data++){
3          foo(data);
4          bar(data);
5          if (*data == SEARCH_CHAR){
6              ...
7              break;
8          }
9      }
10     free(data);
11 }
```

Sort a List of Tuples by first element

```
1 static Tuple[] sortArray(Tuple[] uns){
2    return Arrays.sort(
3      uns, new Comparator<Tuple>() {
4       public int compare(
5        Tuple o1, Tuple o2) {
6          return o1.get(0) == o2.get(0);
7        }
8    });
9 }
```

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

1

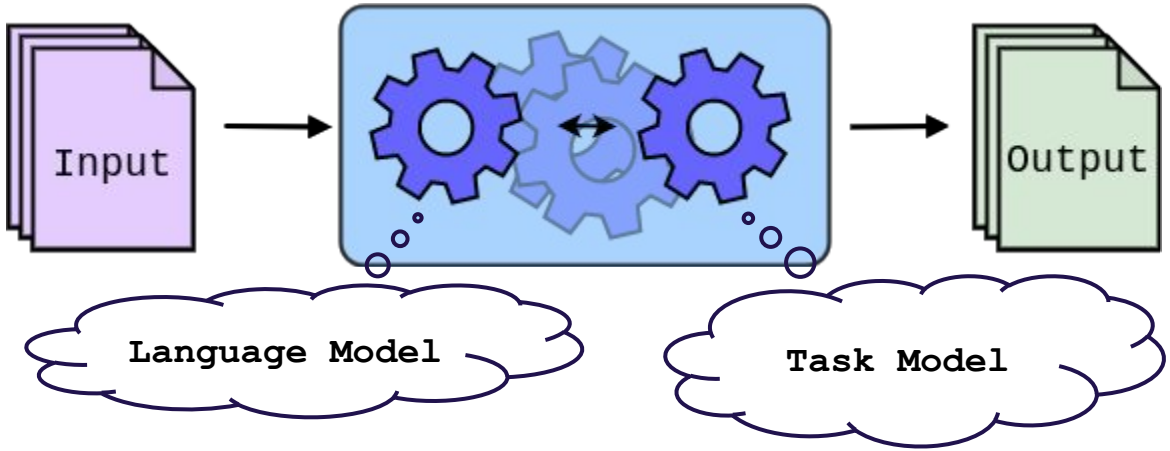
Learning about the "Language"

4. [illegible] code.

5

Learning about the "Task"

Some Interesting points to note



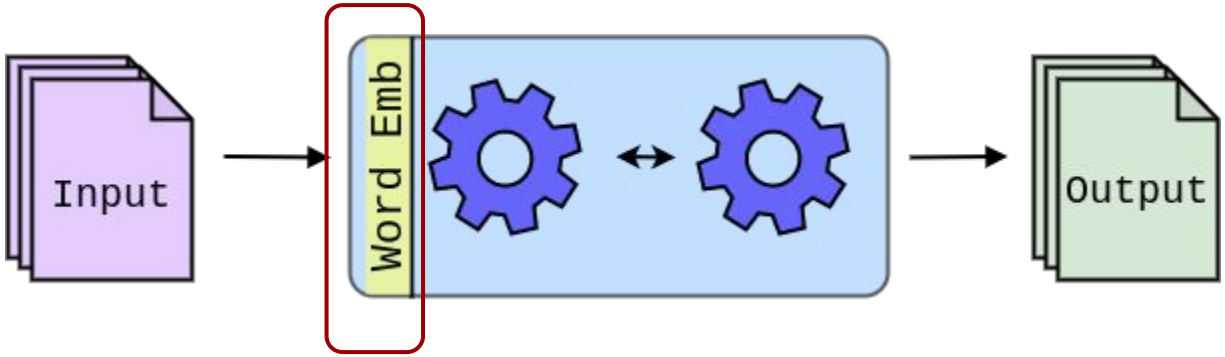
What are the challenges in joint learning?

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.

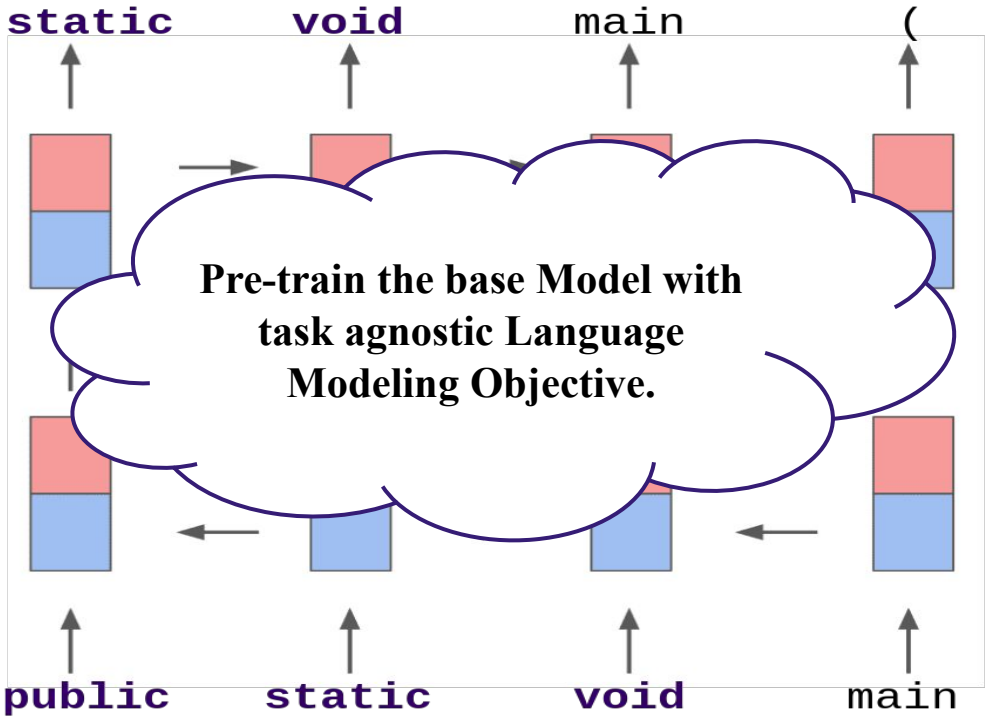
Can we lessen the burden for model?

Can we transfer any knowledge from elsewhere?

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



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



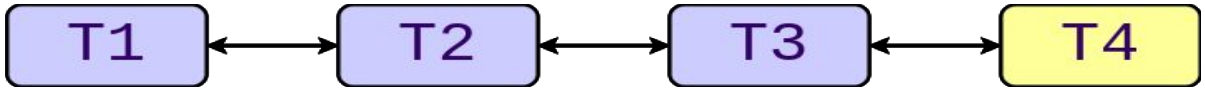
ELMo (pros and cons)

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

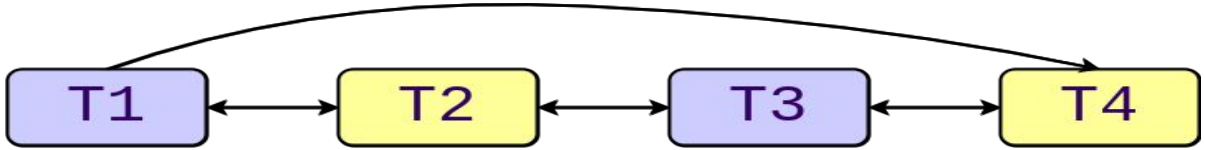
- Prospective Solution :
  - Pretrain tree of graph based models.

Information propagation in models

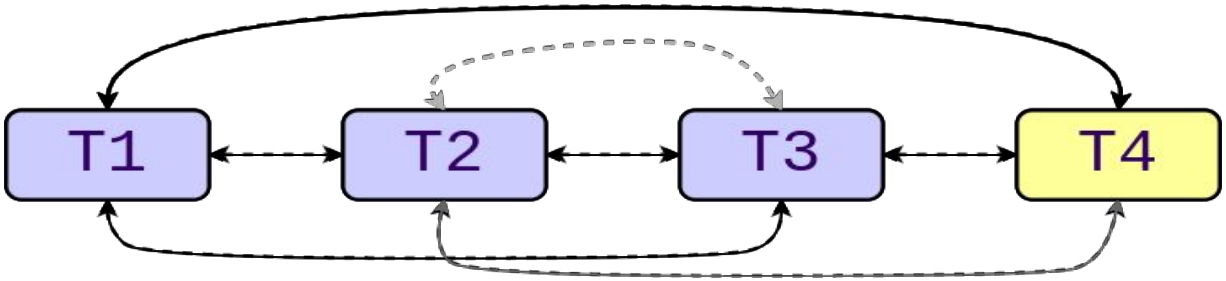
1. Sequence Based Models



2. Graph Based Models

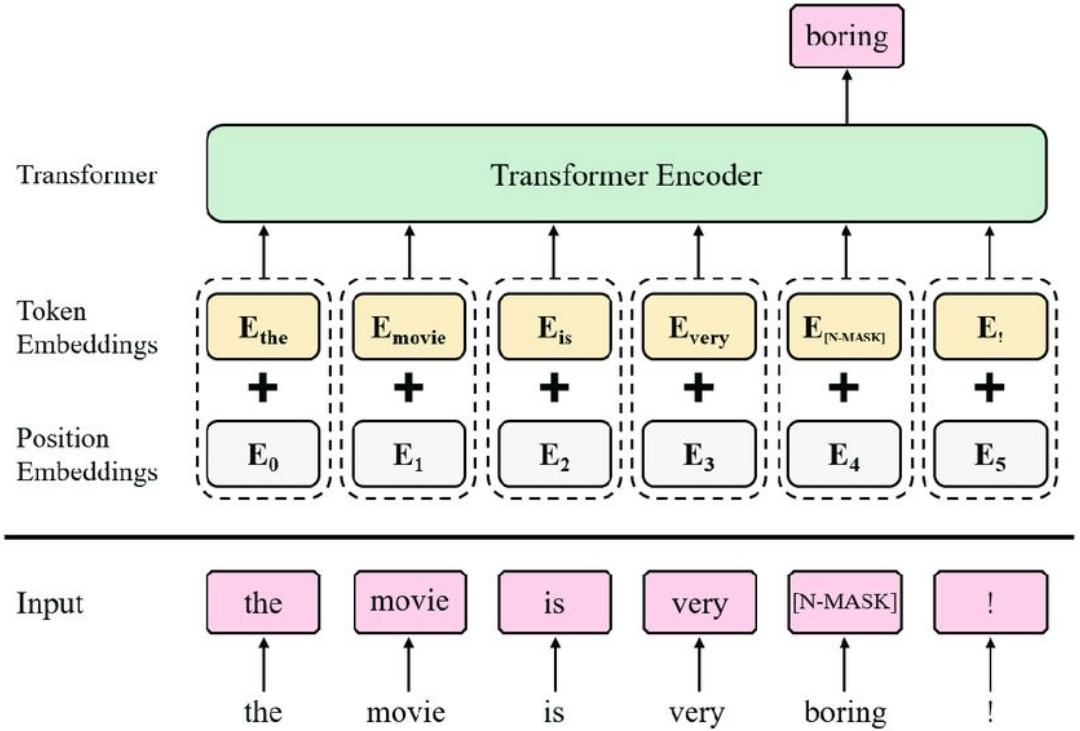


Transformer - A fantastic Idea



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.

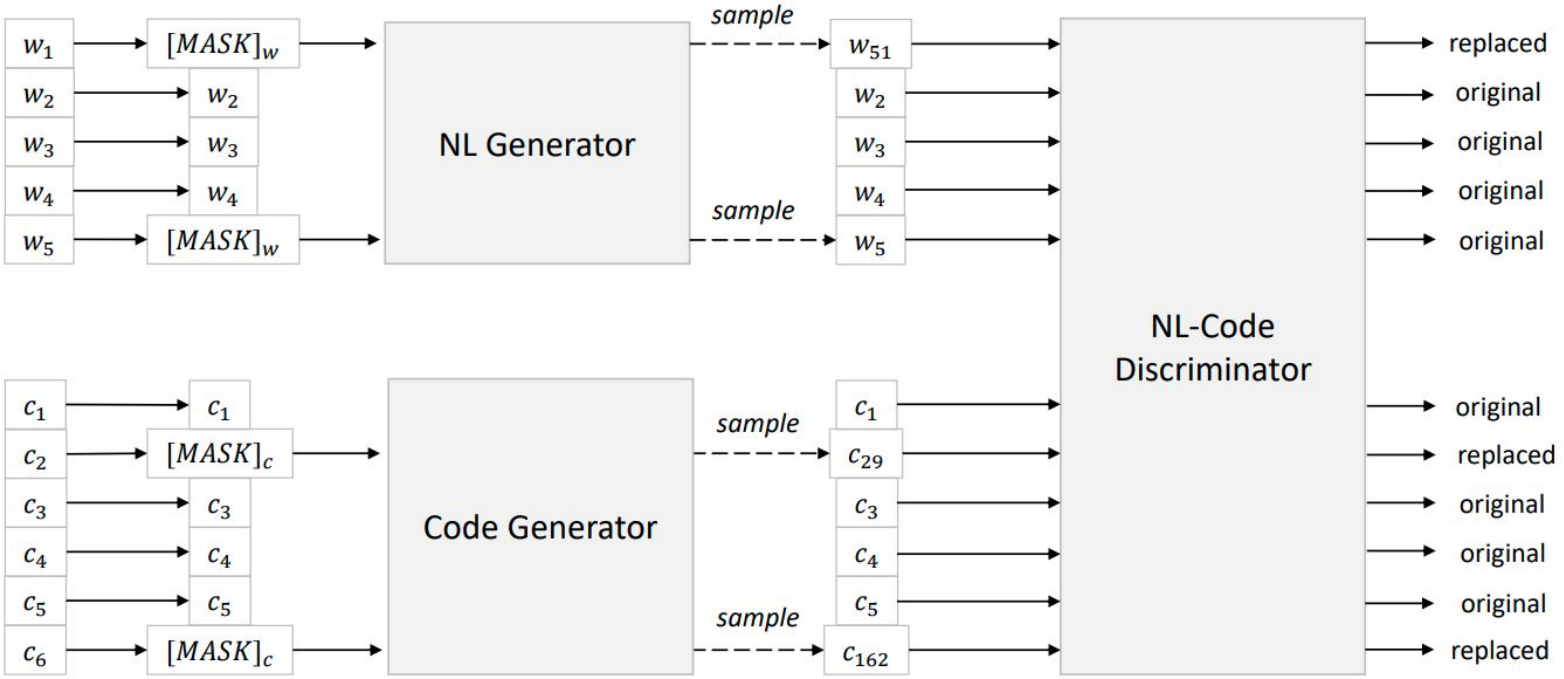
BERT-Pretrained Transformer



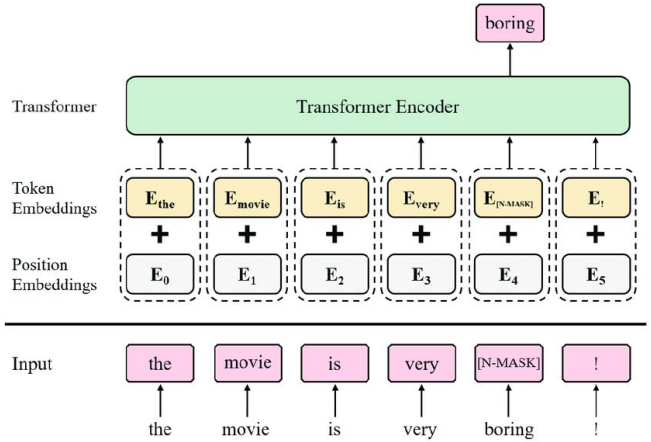
Pre-training:
Task agnostic Masked Language Model.

Fine Tuning:
Task Specific Objective.

CodeBERT - BERT for Code

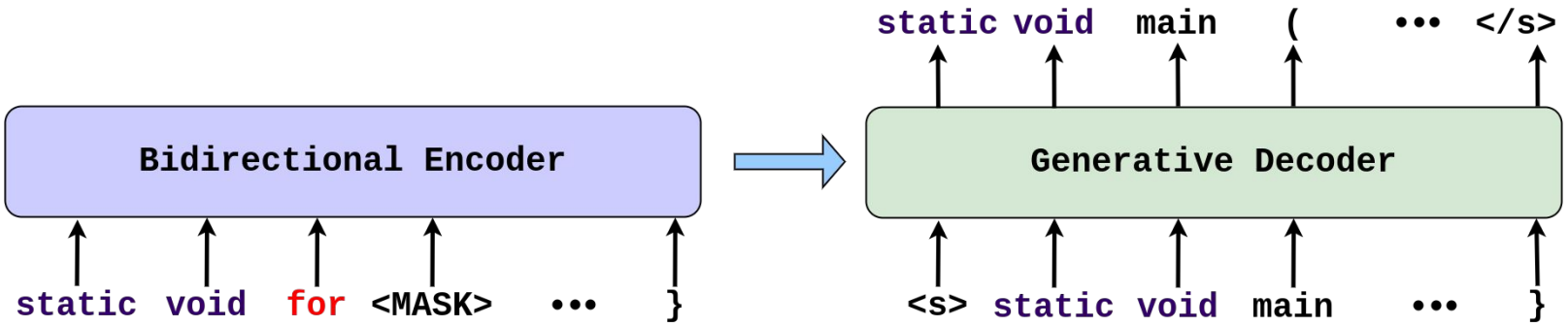


BERT - Any Problem?



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.

BART / PLBART (Denoising auto-encoding)



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.

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**
PLBART_{10K}	17.25	31.40	33.32
PLBART_{20K}	18.45	34.00	35.75
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**

Some Interesting examples of PLBART

Example 1 : ✔

Input Code : C#

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

Generated Code : Java

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

Take Away Points

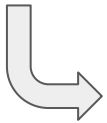
1. Machine learning in source code analysis showed a lot of promise over the years.
2. Source code exhibit different information through different input modalities, such as identifier names, syntax, semantic interaction between identifiers.
3. A good model for a particular task should exploit appropriate 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.

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


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

Potential Solutions - Code Editing

```
1. void isEmpty() {
2.     if (first == null){
3.         return true;
4.     }
5.     else{
6.         return false;
7.     }
8. }
```



```
1. void isEmpty() {
2.     return first == null;
3. }
```

A sample pattern

If code fragments matches
"if ($condition) {return true;} else {return false;}"
Replace with
"return $condition;"

That is just one pattern. How many pattern shall we write to give the developer a complete solution?

Questions?

Thanks!