An Empirical Investigation into Learning Bug-Fixing Patches in the Wild via Neural Machine Translation

Michele Tufano, Cody Watson, Gabriele Bavota,
Massimiliano Di Penta, Martin White, Denys Poshyvanyk

Can you translate
buggy code
into fixed code?

Can you translate buggy code into fixed code?

Google

Translate

Buggy | English | Spanish | Detect language

```
public void addElement ( Element <?> elem) {
     myList.add(elem);
}
```

Fixed | Dutch | Spanish | Translate

```
public void addElement ( Element <?> elem) {
    if (! myList.contains(elem))
         myList.add(elem);
}
```

Why?

Automated Program Repair is (arguably) one of the most exciting research problem in SE.

//TODO - <insert example of bug which costed a lot of money>

//TODO - <sentence about testing and fixing being expensive>

How?

Via Neural Machine Translation by
Learning from past mistakes (historical bug-fixes)

How?

Via Neural Machine Translation by
Learning from past mistakes (historical bug-fixes)

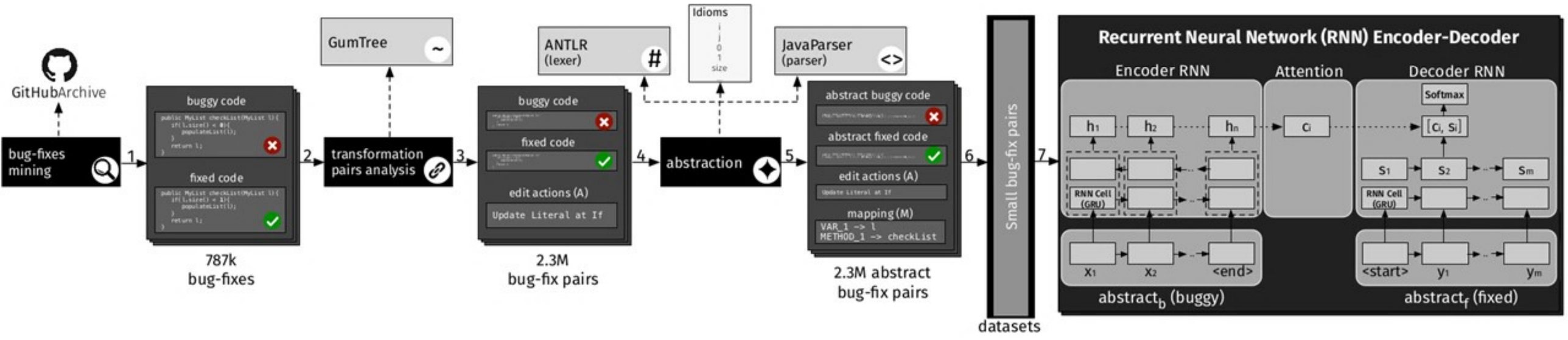
GitHub

March 2011 - October 2017

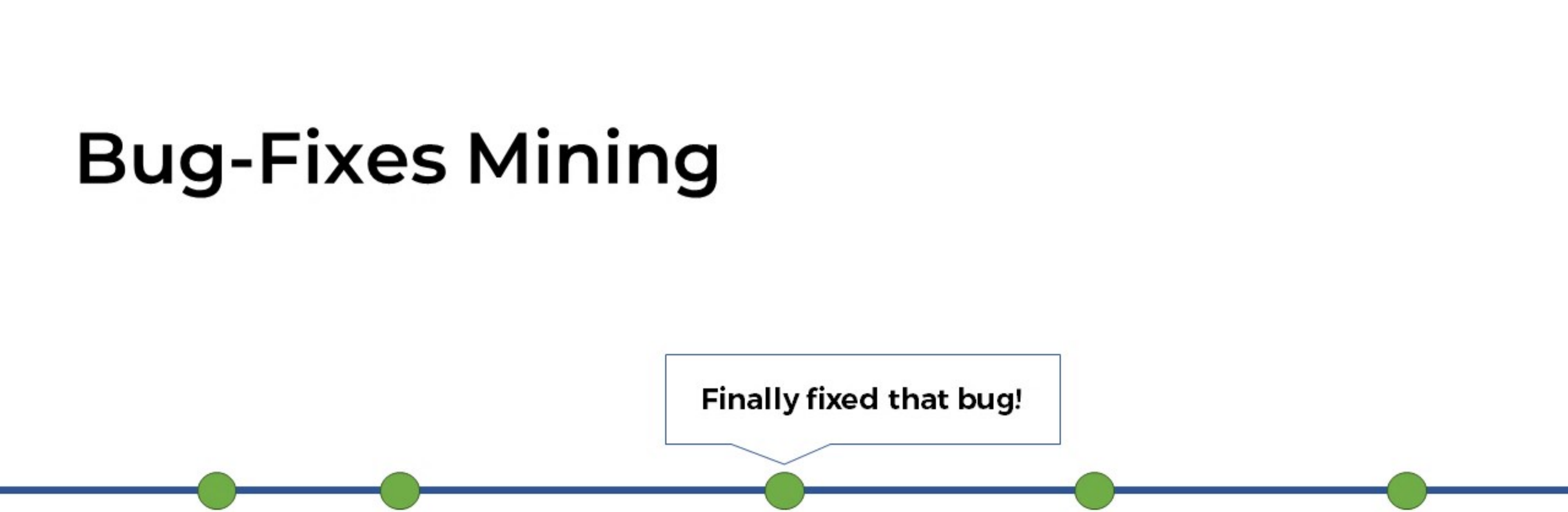
$\sim 10M$illion Bug-fixing commits

Overview

1. Bug-fixes mining
2. Transformation Pairs extraction
3. Code Abstraction
4. NMT (Encoder-Decoder) training



Bug-Fixes Mining

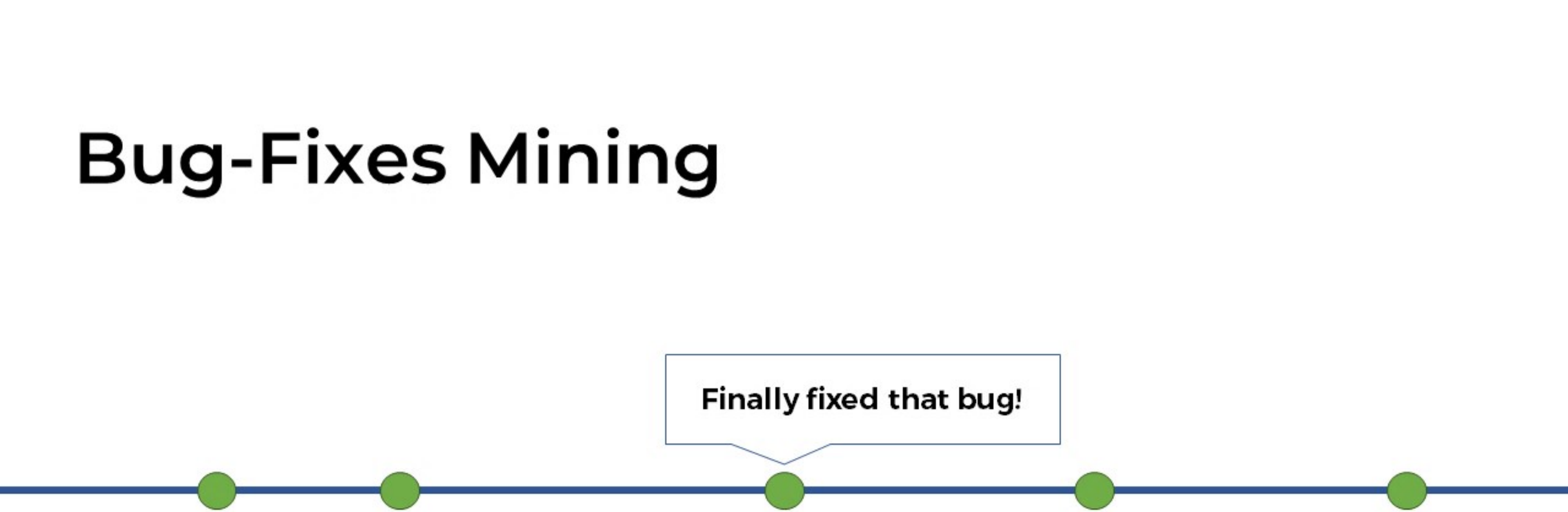


Archive

Regex for Comments

```
(“fix” or “solve” ) AND
(“bug” or “issue” or “problem” or “error” )
```

Bug-Fixes Mining



Archive

Regex for Comments

```
(“fix” or “solve” ) AND
(“bug” or “issue” or “problem” or “error” )
```

Total Commits

10,056,052

Bug-Fixes Mining

Finally fixed that bug!

Archive

Regex for Comments

```
(“fix” or “solve” ) AND
(“bug” or “issue” or “problem” or “error” )
```

Total Commits

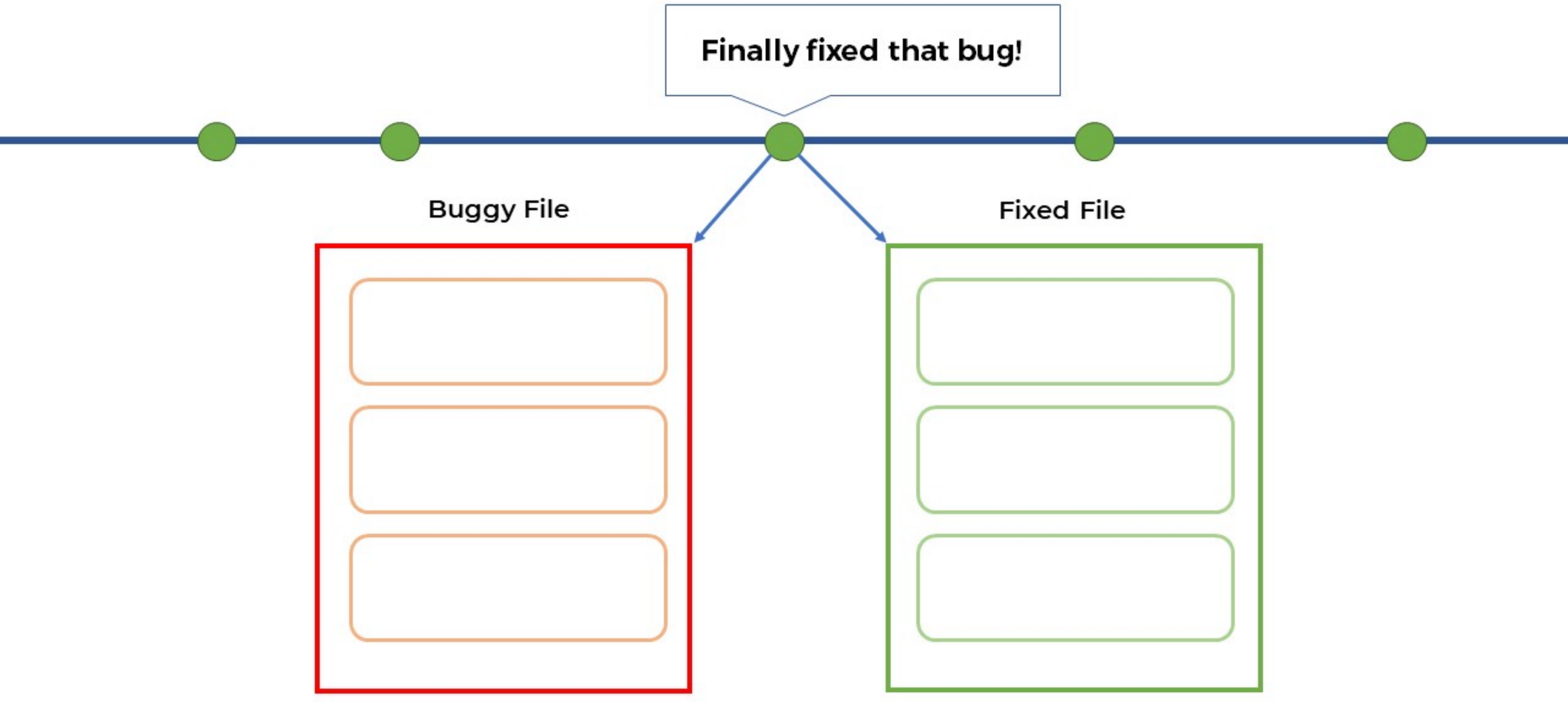
10,056,052

Java Commits

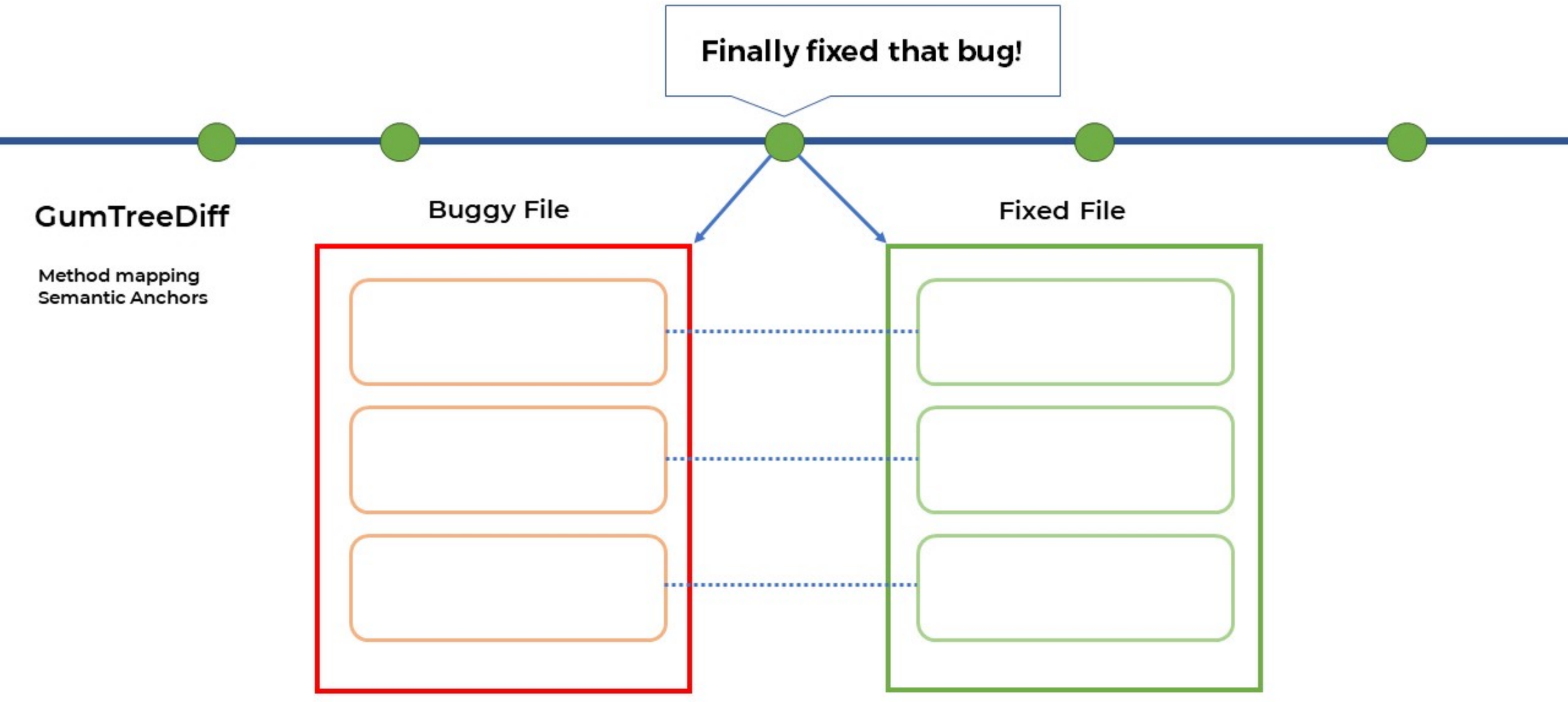
No more than 5 files

787,178

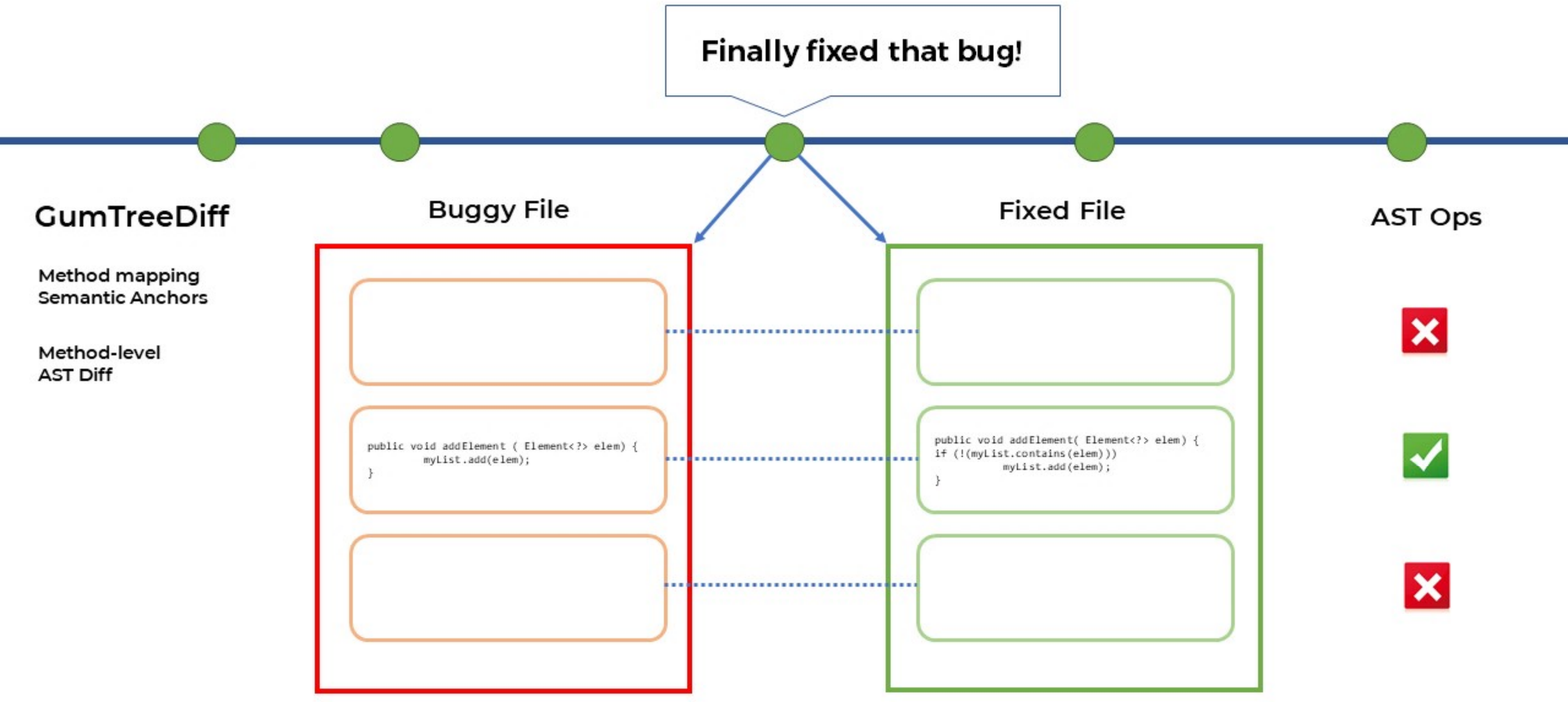
Pair Extraction



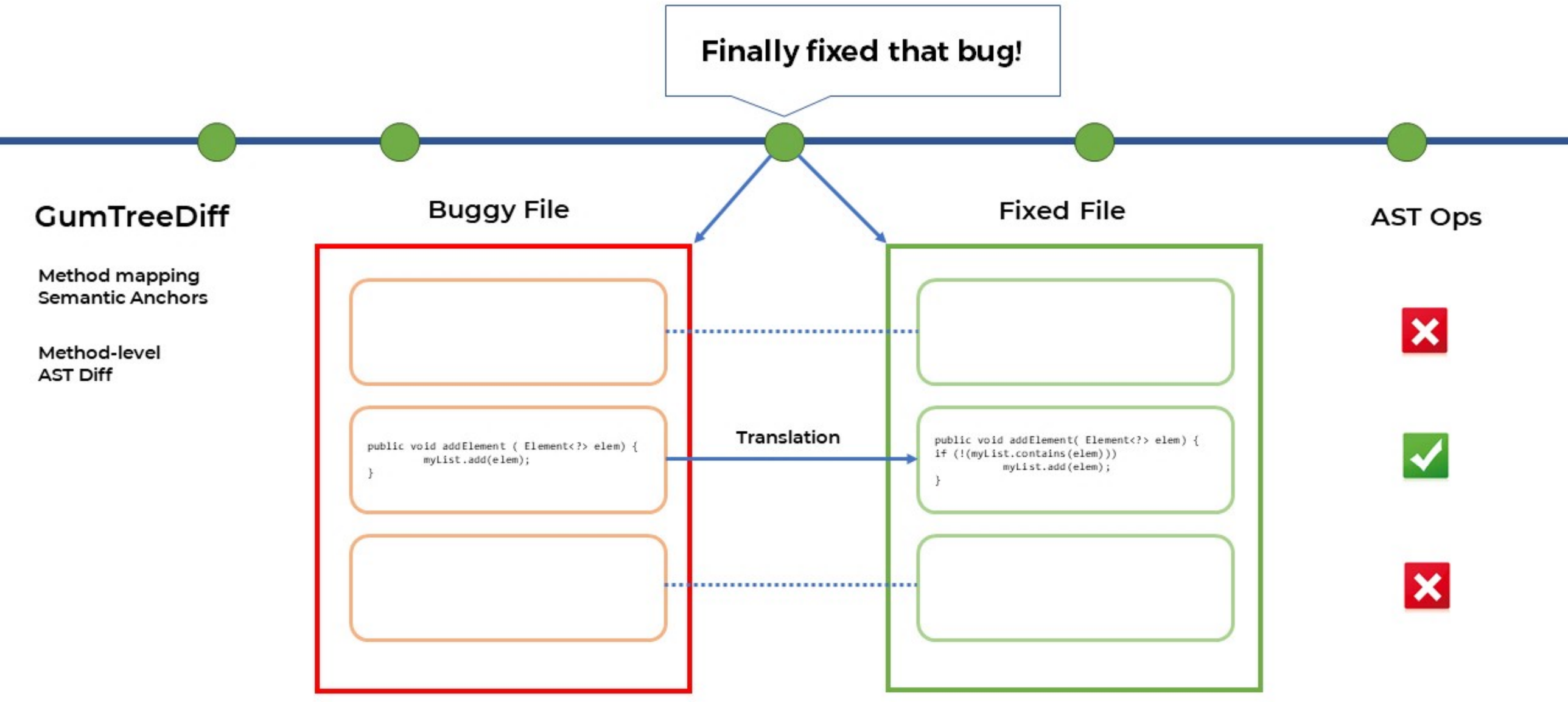
Pair Extraction



Pair Extraction



Pair Extraction



Code Abstraction

Goal: reduce Vocabulary

Source Code

```
public void addElement ( Element <?> elem) { if ( myList.size() > 0) { myList.add(elem); } }
```

Code Abstraction

Goal: reduce Vocabulary

Source Code

```
public void addElement ( Element <?> elem) { if ( myList.size() > 0) { myList.add(elem); } }
```

Abstracted code

```
public void        (       <?>      ) { if (           () >  ) {          (     ); } }
```

- Java Keywords and separators

Code Abstraction

Goal: reduce Vocabulary

Source Code

```
public void addElement ( Element <?> elem) { if ( myList.size() > 0) { myList.add(elem); } }
```

Abstracted code

```
public void          (        <?>      ) { if (      .size() > 0) {      .add(      ); } }
```

- Java Keywords and separators
- Idioms: frequent identifiers and literals (e.g. size, add, 0)

Code Abstraction

Goal: reduce Vocabulary

Source Code

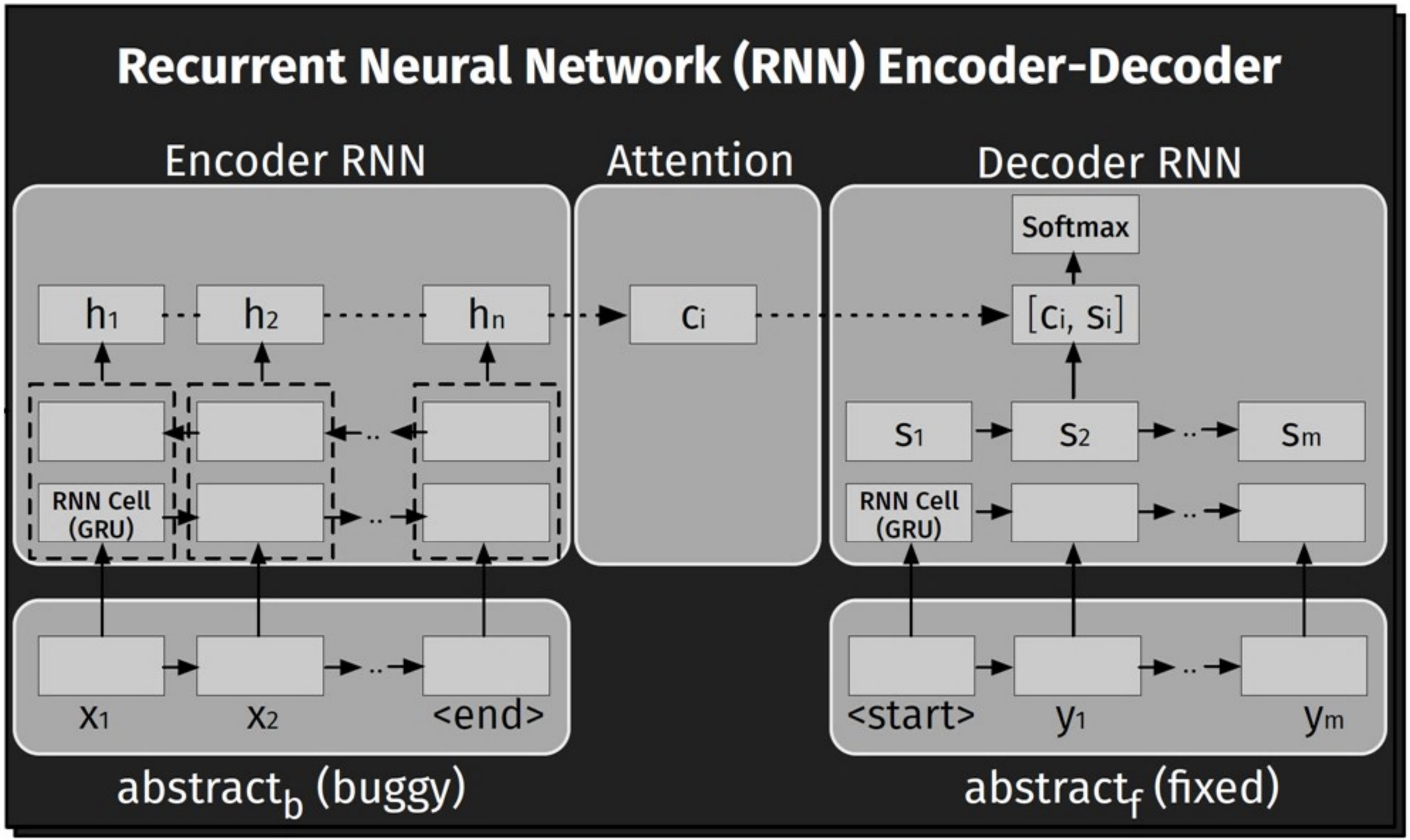
```java
public void addElement ( Element <?> elem) { if ( myList.size() > 0) { myList.add(elem); } }
```

Abstracted code

```java
public void METHOD_1 ( TYPE_1 <?> VAR_1) { if ( VAR_2.size() > 0) { VAR_2.add(VAR_1); } }
```

- Java Keywords and separators
- Idioms: frequent identifiers and literals (e.g. size, add, 0)
- IDs : replace identifiers and literals with typified IDs (e.g., METHOD, TYPE, VAR, INT, STRING, etc.)

Learning Fixes



Learning Fixes

Hyperparameters

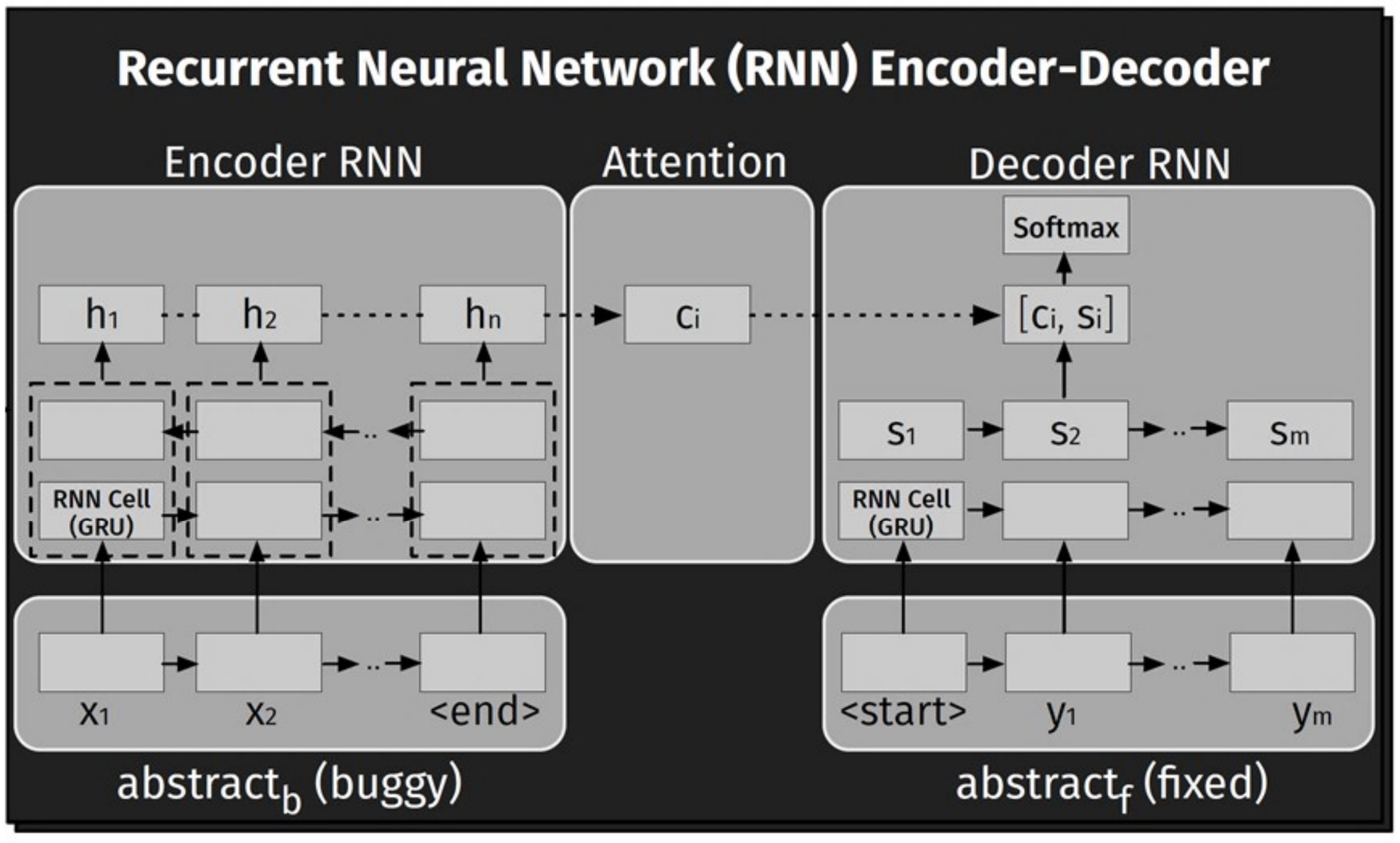
10 configurations

RNN Cells
- LSMT
- GRU

Layers
- 1
- 2
- 4

Units
- 256
- 512

Embedding Size
- 256
- 512



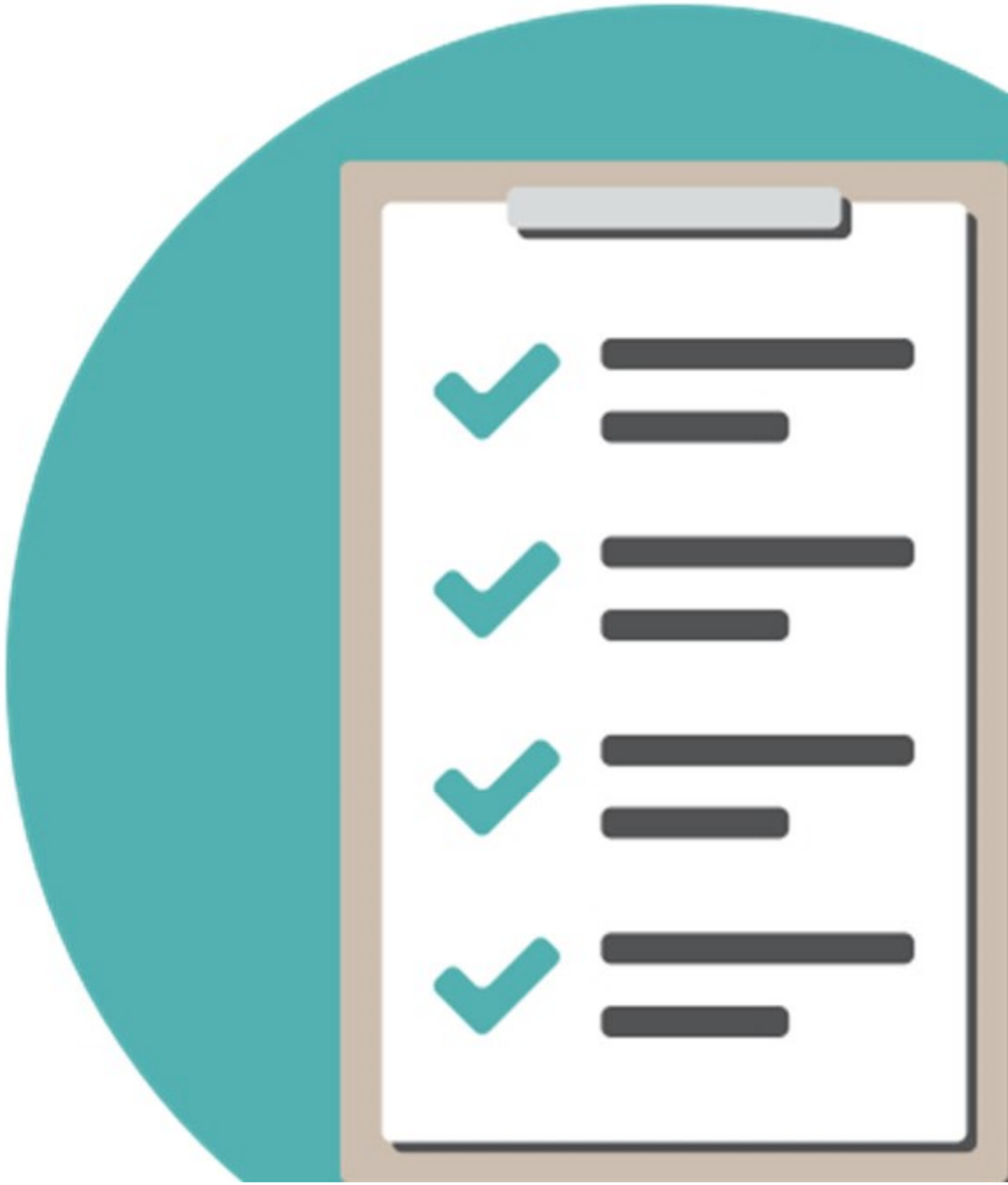
Evaluation

Small Methods
No longer than 50 tokens

Dataset: 58,350 methods

- 80% Training
- 10% Validation
- 10% Test

No duplicates
Unique at source and abstracted code level



Buggy Code

```
public void addElement ( Element <?> elem) { myList.add(elem); }
```

Buggy Code

```java
public void addElement ( Element <?> elem) { myList.add(elem); }
```

Abstracted Buggy Code

```java
public void METHOD_1 ( TYPE_1 <?> VAR_1) { VAR_2.add(VAR_1); }
```

Mapping

ID	Value
METHOD_1	addElement
TYPE_1	Element
VAR_1	elem
VAR_2	myList

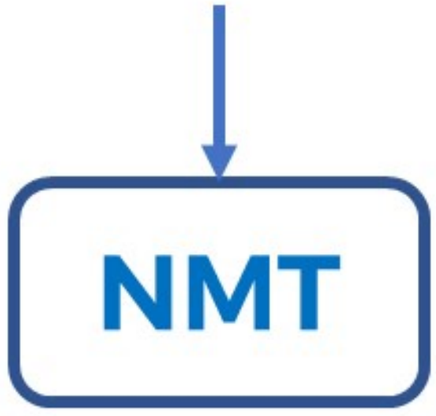
Buggy Code

```
public void addElement ( Element <?> elem) { myList.add(elem); }
```

Abstracted Buggy Code

```
public void METHOD_1 ( TYPE_1 <?> VAR_1) { VAR_2.add(VAR_1); }
```

Neural Machine Translation



Mapping

ID	Value
METHOD_1	addElement
TYPE_1	Element
VAR_1	elem
VAR_2	myList

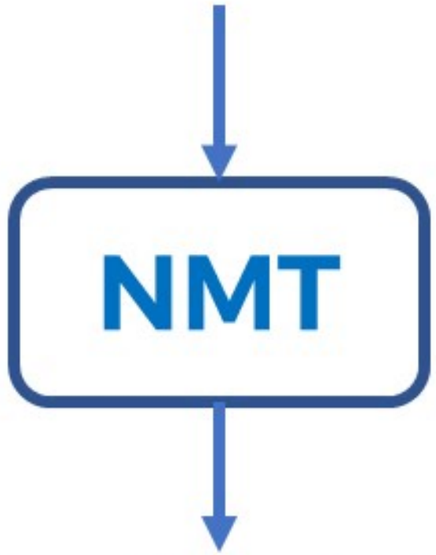
Buggy Code

```
public void addElement ( Element <?> elem) { myList.add(elem); }
```

Abstracted Buggy Code

```
public void METHOD_1 ( TYPE_1 <?> VAR_1) { VAR_2.add(VAR_1); }
```

Neural Machine Translation



Abstracted Fixed Code

```
public void METHOD_1 ( TYPE_1 <?> VAR_1) { if (! VAR_2.contains(VAR_1)) VAR_2.add(VAR_1); }
```

Mapping

ID	Value
METHOD_1	addElement
TYPE_1	Element
VAR_1	elem
VAR_2	myList

Buggy Code

```java
public void addElement ( Element <?> elem) { myList.add(elem); }
```

Abstracted Buggy Code

```java
public void METHOD_1 ( TYPE_1 <?> VAR_1) { VAR_2.add(VAR_1); }
```

Neural Machine Translation

NMT

Abstracted Fixed Code

```java
public void METHOD_1 ( TYPE_1 <?> VAR_1) { if (! VAR_2.contains(VAR_1)) VAR_2.add(VAR_1); }
```

Mapping

ID	Value
METHOD_1	addElement
TYPE_1	Element
VAR_1	elem
VAR_2	myList

But can you generate real source code?

But can you generate real source code?

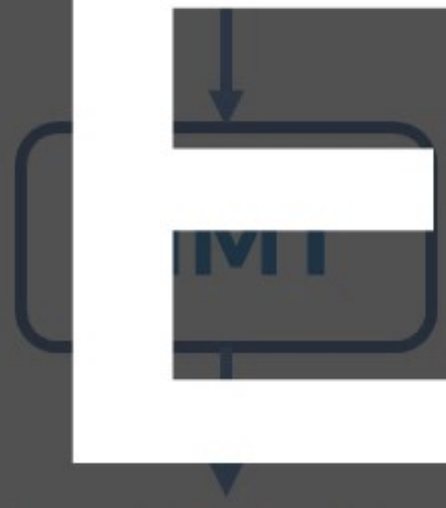
Buggy Code

```
public void addElement ( Element <?> elem) { myList.add(elem); }
```

Abstracted Buggy Code

```
public void METHOD_1 ( TYPE_1 <?> VAR_1) { VAR_2.add(VAR_1); }
```

Neural Machine Translation



Abstracted Fixed Code

```
public void METHOD_1 ( TYPE_1 <?> VAR_1) { if (! VAR_2.contains(VAR_1)) VAR_2.add(VAR_1); }
```

Mapping

ID	Value
METHOD_1	addElement
TYPE_1	Element
VAR_1	elem
VAR_2	myList

YES!

Enlarged font for Reviewers

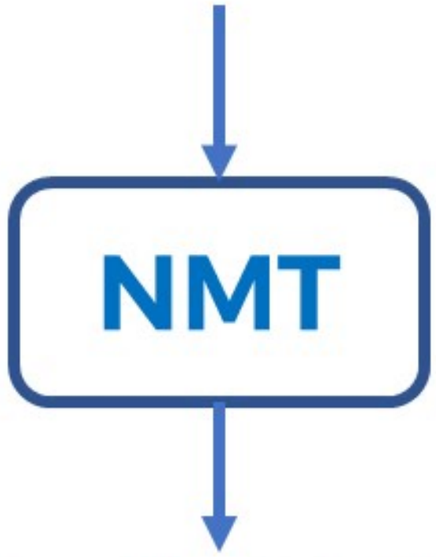
Buggy Code

```
public void addElement ( Element <?> elem) { myList.add(elem); }
```

Abstracted Buggy Code

```
public void METHOD_1 ( TYPE_1 <?> VAR_1) { VAR_2.add(VAR_1); }
```

Neural Machine Translation



Abstracted Fixed Code

```
public void METHOD_1 ( TYPE_1 <?> VAR_1) { if (! VAR_2.contains(VAR_1)) VAR_2.add(VAR_1); }
```

Fixed Code

```
public void addElement ( Element <?> elem) { if (! myList.contains(elem)) myList.add(elem); }
```

Mapping

ID	Value
METHOD_1	addElement
TYPE_1	Element
VAR_1	elem
VAR_2	myList

Results

538 bug-fixes*

9.22% test set

*Unique at source and abstract code level.
*Never seen in training nor validation set.

Results

538 bug-fixes*

9.22% test set

*Unique at source and abstract code level.
*Never seen in training nor validation set.

1 One Single Patch Attempt per bug

Can you generate multiple candidate patches?

50 different translations?

50 different translations?

Can you generate multiple candidate patches?

With more candidates...
can you fix 30-50% of the bugs?

50 different translations?

Can you generate multiple candidate patches?

With more candidates...
can you fix 30-50% of the bugs?

What type of AST operations?

50 different translations?

Can you generate multiple candidate patches?

With more candidates...
can you fix 30-50% of the bugs?

What type of AST operations?

Longer methods?

50 different translations?

Can you generate multiple candidate patches?

DATA!!

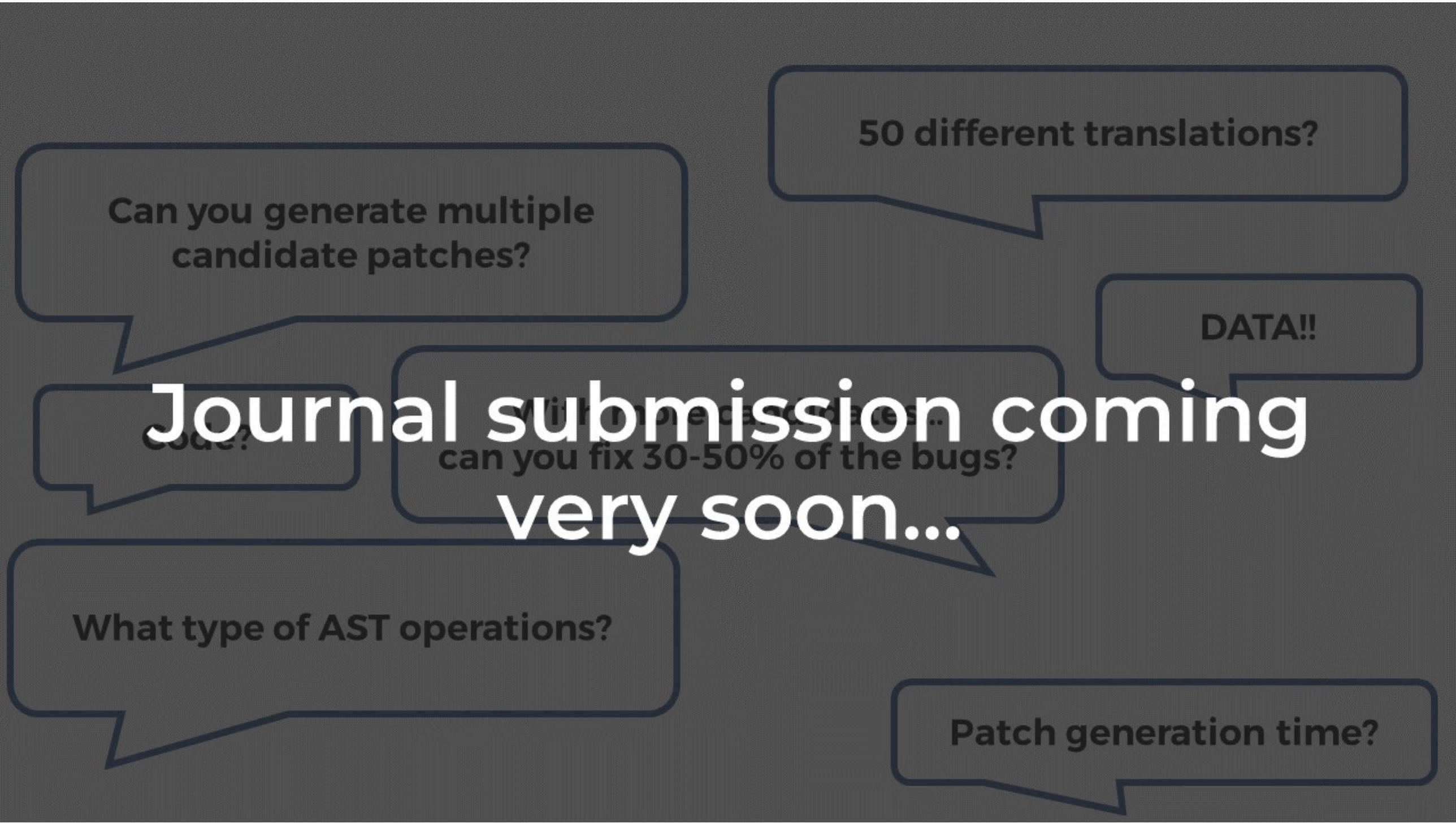
Code?

With more candidates...
can you fix 30-50% of the bugs?

What type of AST operations?

Generation time?

Longer methods?


50 different translations?
Can you generate multiple candidate patches?
DATA!!
can you fix 30-50% of the bugs?
What type of AST operations?
Patch generation time?
Journal submission coming very soon...

Thanks!

Questions?

Michele Tufano

@tufanomichele

http://www.cs.wm.edu/~mtufano/

An Empirical Investigation into
Learning Bug-Fixing Patches
in the Wild via Neural Machine Translation