University of Molise

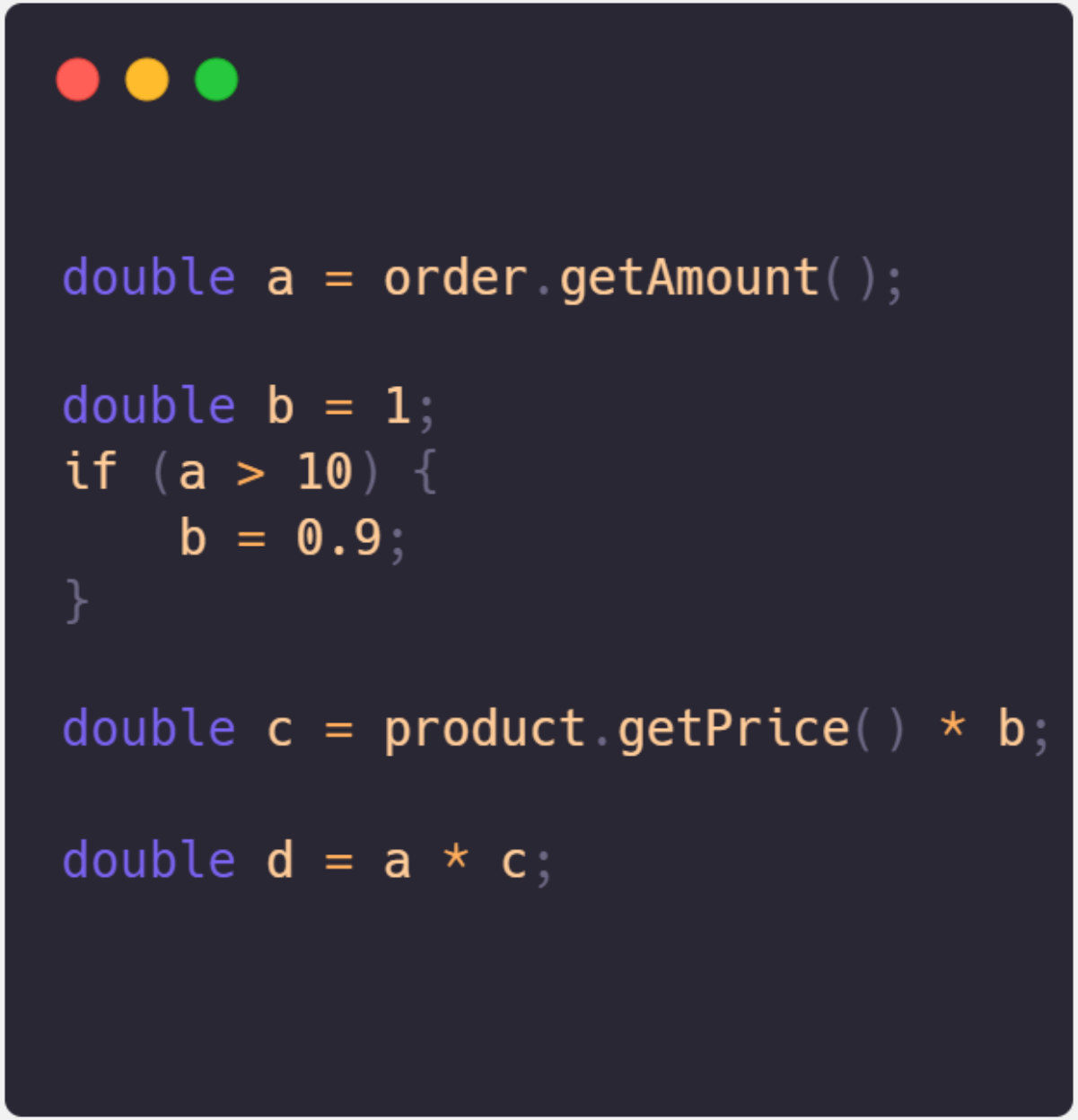
An Empirical Exploration of Deep Learning Techniques for Automated Enhancement of Code Readability in Large-Scale Software Repositories

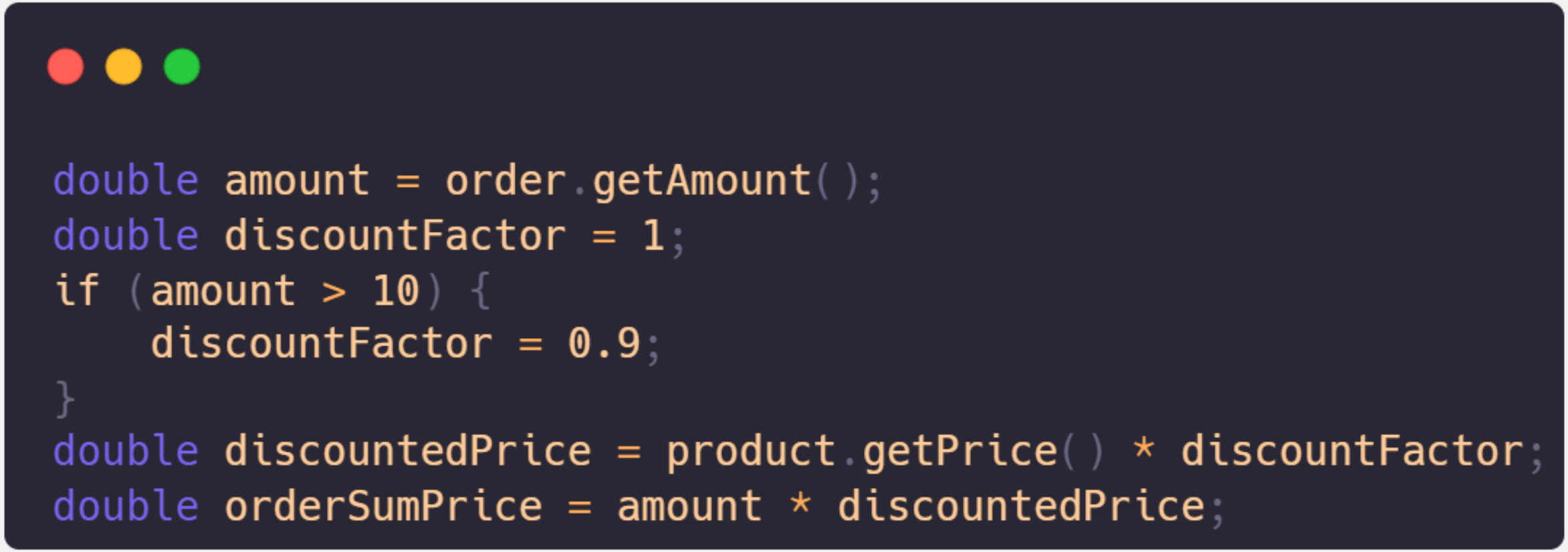
Student:
Marco Omicini

Supervisors:
Prof. Simone Scalabrino,
Dr. Antonio Vitale

2022/2023


Readability


double a = order.getAmount();
double b = 1;
if (a > 10) {
b = 0.9;
}
double c = product.getPrice() * b;
double d = a * c;


double amount = order.getAmount( );
double discountFactor = 1;
if (amount > 10) {
discountFactor = 0.9;
}
double discountedPrice = product.getPrice( ) * discountFactor;
double orderSumPrice = amount * discountedPrice;

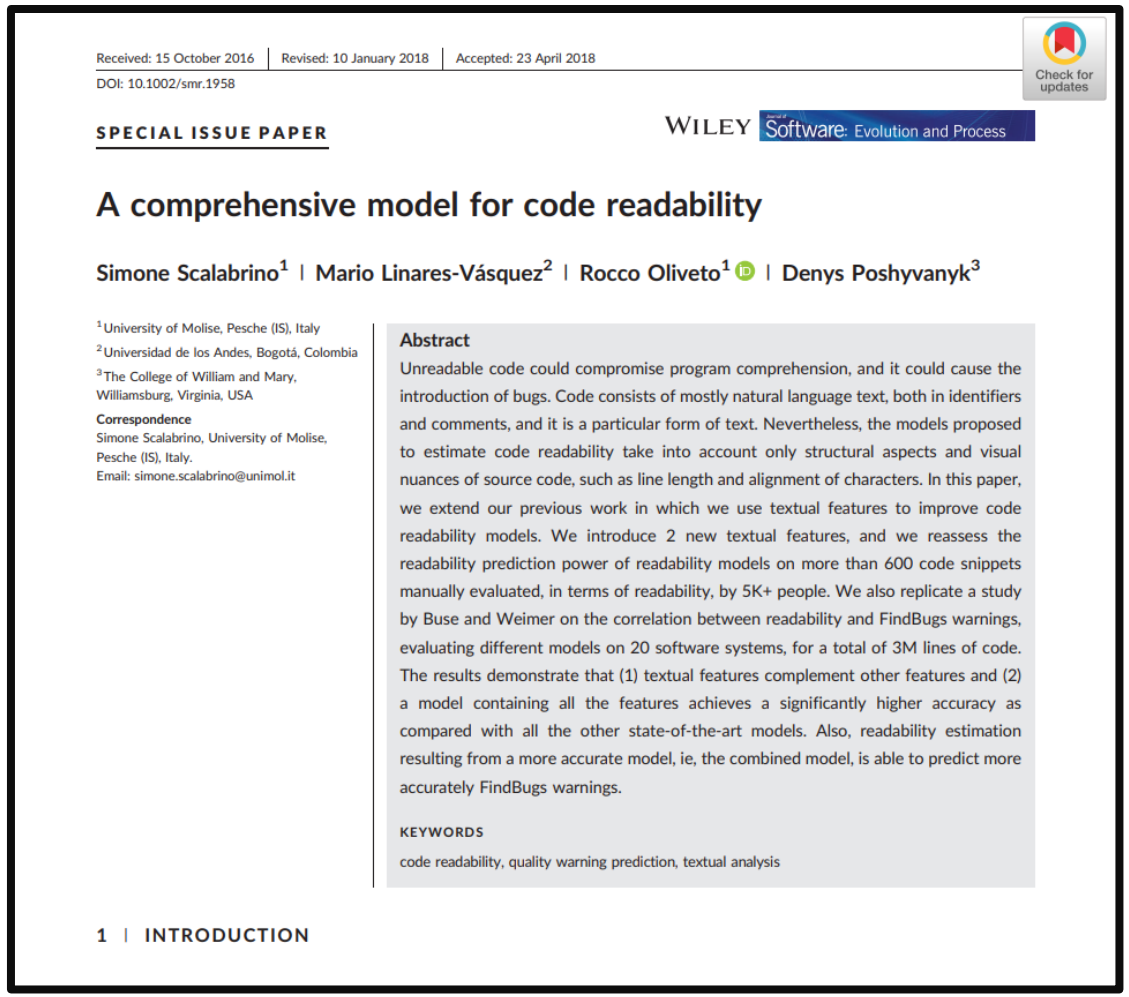
Refactoring

Technical Debt

Automation

State of the *Art*

Automatically assess code readability



Received: 15 October 2016 | Revised: 10 January 2018 | Accepted: 23 April 2018
DOI: 10.1002/smr.1958

Check for updates

SPECIAL ISSUE PAPER

WILEY Software: Evolution and Process

A comprehensive model for code readability

Simone Scalabrino[1] | Mario Linares-Vásquez[2] | Rocco Oliveto[1] | Denys Poshyvanyk[3]

[1] University of Molise, Pesche (IS), Italy
[2] Universidad de los Andes, Bogotá, Colombia
[3] The College of William and Mary, Williamsburg, Virginia, USA

Correspondence
Simone Scalabrino, University of Molise, Pesche (IS), Italy.
Email: simone.scalabrino@unimol.it

Abstract

Unreadable code could compromise program comprehension, and it could cause the introduction of bugs. Code consists of mostly natural language text, both in identifiers and comments, and it is a particular form of text. Nevertheless, the models proposed to estimate code readability take into account only structural aspects and visual nuances of source code, such as line length and alignment of characters. In this paper, we extend our previous work in which we use textual features to improve code readability models. We introduce 2 new textual features, and we reassess the readability prediction power of readability models on more than 600 code snippets manually evaluated, in terms of readability, by 5K+ people. We also replicate a study by Buse and Weimer on the correlation between readability and FindBugs warnings, evaluating different models on 20 software systems, for a total of 3M lines of code. The results demonstrate that (1) textual features complement other features and (2) a model containing all the features achieves a significantly higher accuracy as compared with all the other state-of-the-art models. Also, readability estimation resulting from a more accurate model, ie, the combined model, is able to predict more accurately FindBugs warnings.

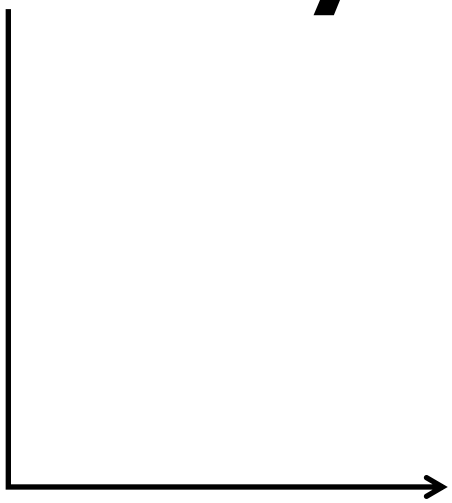
KEYWORDS

code readability, quality warning prediction, textual analysis

1 | INTRODUCTION

Scalabrino et. al

Automatically improve code readability



Using Deep Learning to Automatically Improve Code Readability

Antonio Vitale, Valentina Piantadosi, Simone Scalabrino, Rocco Oliveto
STAKE Lab @ University of Molise, Italy

***Abstract*—Reading source code occupies most of developer's daily activities. Any maintenance and evolution task requires developers to read and understand the code they are going to modify. For this reason, previous research focused on the definition of techniques to automatically assess the readability of a given snippet. However, when many unreadable code sections are detected, developers might be required to manually modify them all to improve their readability. While existing approaches aim at solving specific readability-related issues, such as improving variable names or fixing styling issues, there is still no approach to automatically suggest which actions should be taken to improve code readability.**

In this paper, we define the first holistic readability-improving approach. As a first contribution, we introduce a methodology for automatically identifying readability-improving commits, and we use it to build a large dataset of 122k commits by mining the whole revision history of all the projects hosted on GitHub between 2015 and 2022. We show that such a methodology has ∼86% accuracy. As a second contribution, we train and test the T5 model to emulate what developers did to improve readability. We show that our model achieves a perfect prediction accuracy between 21% and 28%. The results of a manual evaluation we performed on 500 predictions shows that when the model does not change the behavior of the input and it applies changes (34% of the cases), in the large majority of the cases (79.4%) it allows to improve code readability.

***Index Terms*—code readability, large language models, t5**

I. Introduction

presented approaches aimed at improving aspects naturally related to code readability, like styling [25] and naming [1], [24], [46]. Still, to the best of our knowledge, no previous work introduced an approach specifically aimed at improving code readability as a whole.

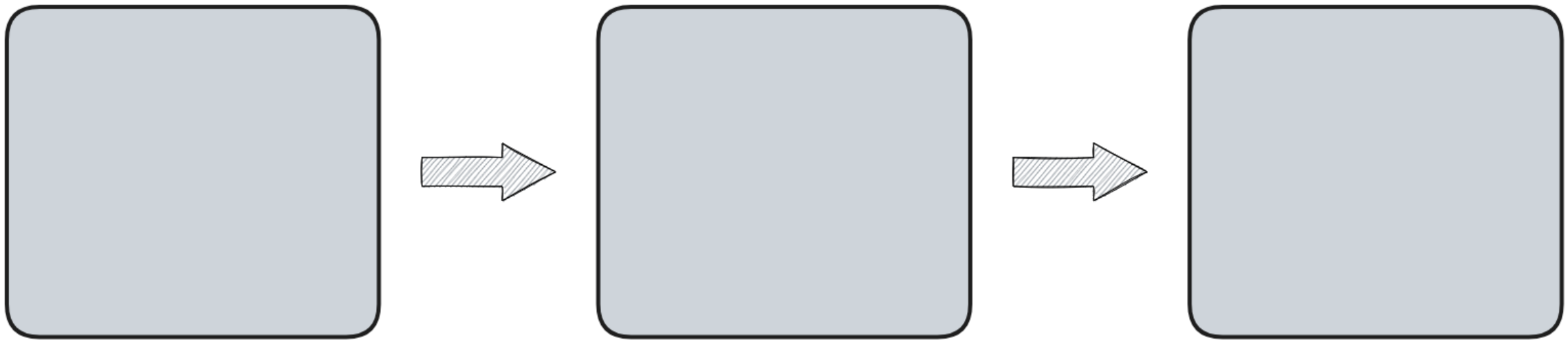
Ideally, such an approach would take as input a snippet s with readability-related issues and return a new version of s (s^*) without such issues. A viable solution is using Deep Learning (DL) and, specifically, Large Language Models (LLMs). LLMs have been successfully adopted for several coding task, like automated bug fixing and generation of assertion statements for test cases [28]. To use such approaches, however, plenty of pairs (s, s^*) are needed to fine tune the model. Manually defining an adequately large dataset is not an option because of the colossal amount of work required.

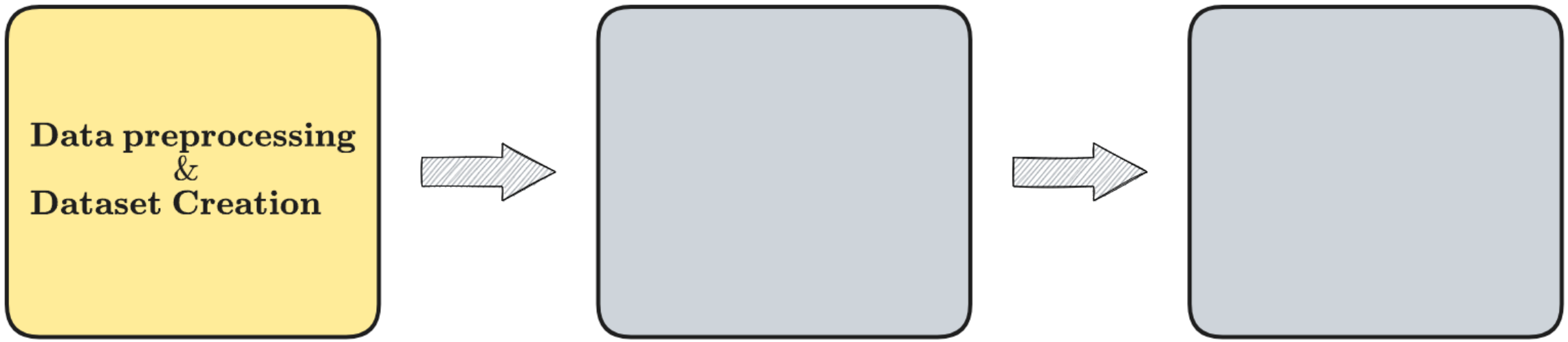
In this regard, software repositories might be a valuable source of information. Developers improve code readability in the evolution of open-source software projects all the time. Let us consider as an example commit `3acfc16` from the GitHub repository `RoboJackets/roboracing-software` [41]. The commit message, "Made the code readable," clearly states the developer's intention. Automatically collecting such commits would allow us to fine tune a LLM able to automatically improve code readability. Still, this is not an easy task. A simple keyword-based approach would not work, because the

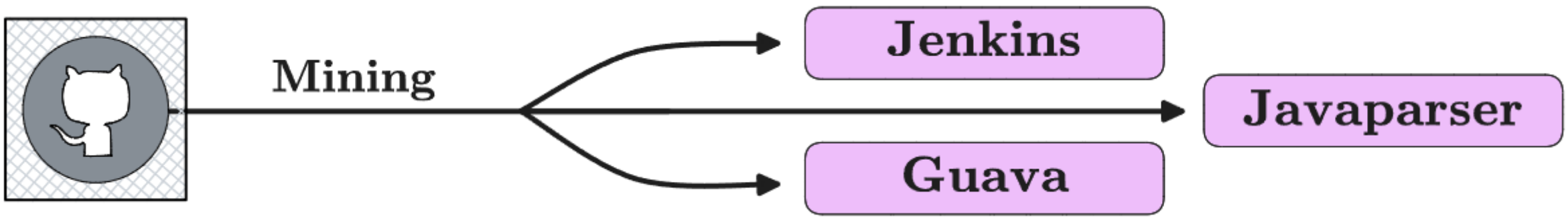
Vitale et. al

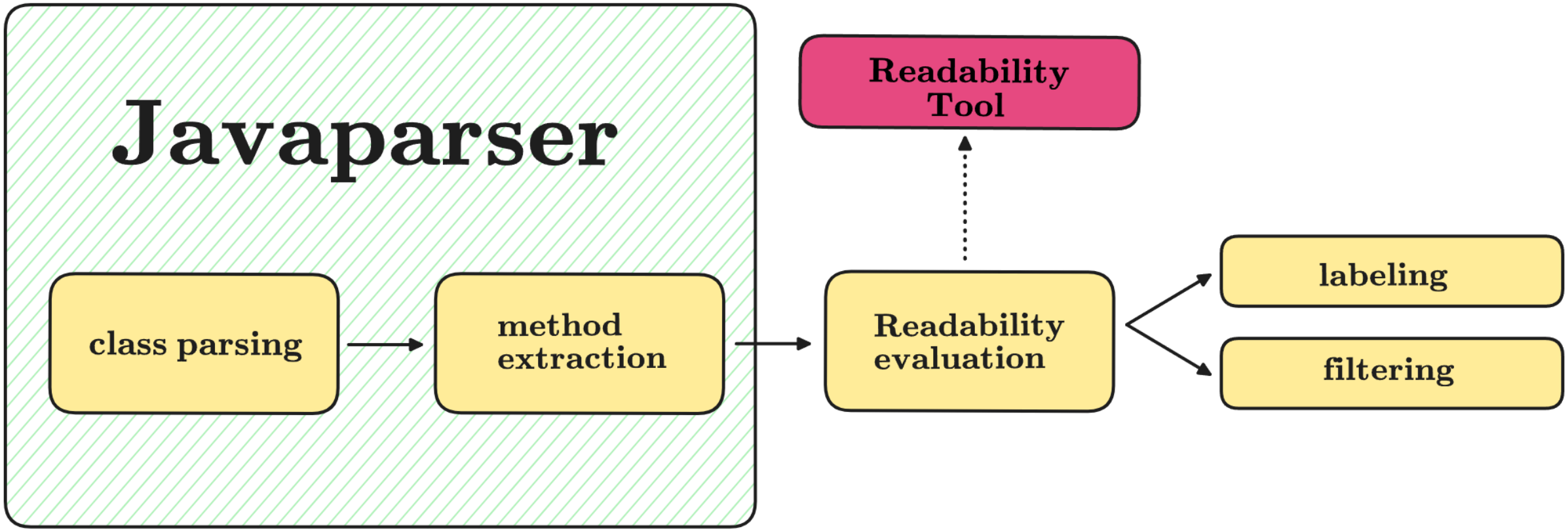

GOALS

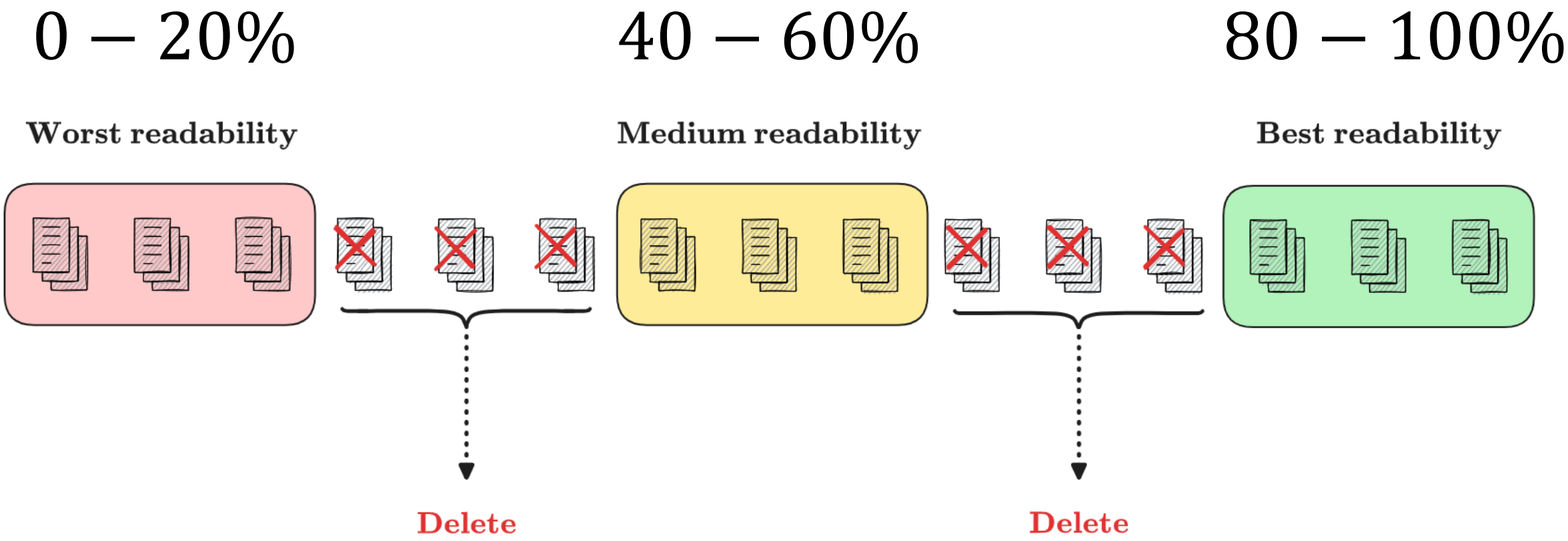
Experimental Procedure




Data preprocessing
&
Dataset Creation


Mining
Jenkins
Guava
Javaparser


Javaparser
class parsing
method extraction
Readability Tool
Readability evaluation
labeling
filtering


0 – 20%
40 – 60%
80 – 100%
Worst readability
Medium readability
Best readability
Delete
Delete

LISTING 3.2: Method's Information


```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```

LISTING 3.2: Method's Information


```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```

LISTING 3.2: Method's Information


```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```

LISTING 3.2: Method's Information


```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```

LISTING 3.2: Method's Information


```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```

LISTING 3.2: Method's Information

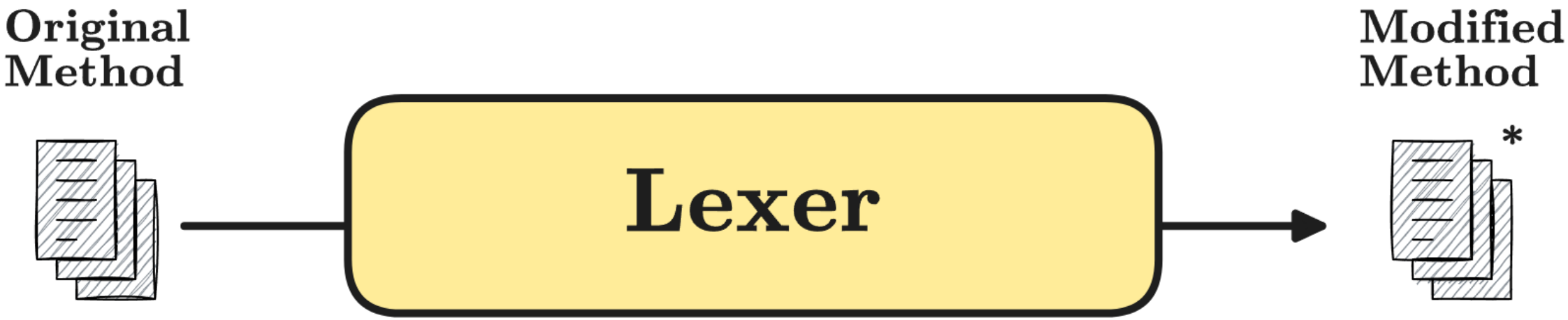

```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```

LISTING 3.2: Method's Information


```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```

LISTING 3.2: Method's Information

```
{
  "name": "orNull",
  "method": "\t\t@Override\n\t\t@CheckForNull\n\t\tpublic T
     orNull(){\n\t\t    return null;\n\t\t}",
  "startLine": 64,
  "endLine": 68,
  "classPath": "guava/android/guava/src/com/google/common/
     base/Absent.java",
  "readabilityScore": 0.8923128247261047,
  "label": "NONE"
}
```


Original
Method
Lexer
Modified
Method
*

Original Method	Tokenized Method
1 public void generate() { 2 final int size = 0; 3 final String alphabet = ""; 4 final String digits = ""; 5 }	1 public $whitespace$ void $whitespace$ generate ( ) $whitespace$ { $newline$ $indentation$ final $whitespace$ int $whitespace$ size $whitespace$ = $whitespace$ $number$; $newline$ $indentation$ final $whitespace$ string $whitespace$ alphabet $whitespace$ = $whitespace$ $string$; $newline$ $indentation$ final $whitespace$ string $whitespace$ digits $whitespace$ = $whitespace$ $string$; $whitespace$

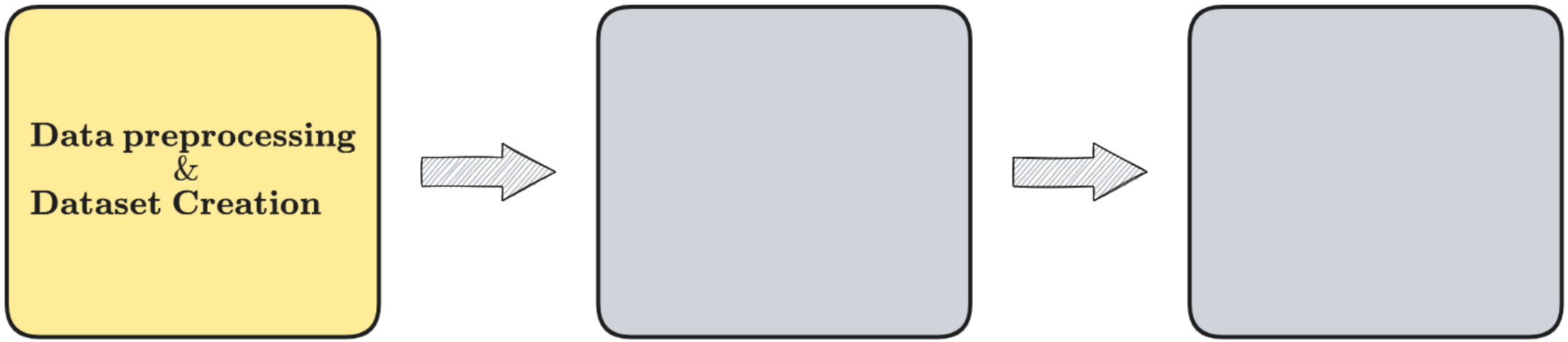
Original Method	Tokenized Method
1 public void generate() { 2 final int size = 0; 3 final String alphabet = ""; 4 final String digits = ""; 5 }	1 public $whitespace$ void $whitespace$ generate ( ) $whitespace$ { $newline$ $indentation$ final $whitespace$ int $whitespace$ size $whitespace$ = $whitespace$ $number$; $newline$ $indentation$ final $whitespace$ string $whitespace$ alphabet $whitespace$ = $whitespace$ $string$; $newline$ $indentation$ final $whitespace$ string $whitespace$ digits $whitespace$ = $whitespace$ $string$; $whitespace$

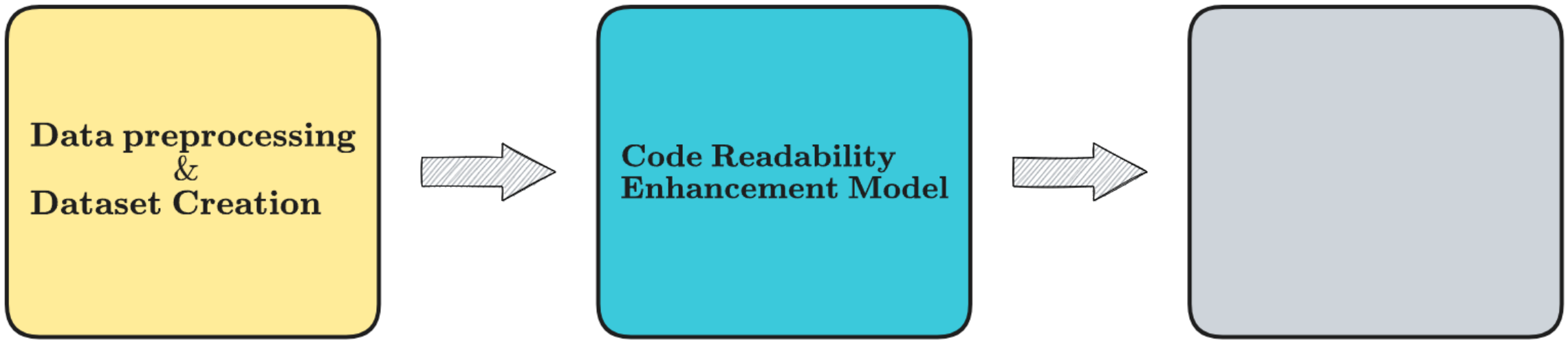
Original Method	Tokenized Method
1 public void generate() { 2 ····final int size = 0; 3 ····final String alphabet = ""; 4 ····final String digits = ""; 5 }	1 public $whitespace$ void $whitespace$ generate ( ) $whitespace$ { $newline$ $indentation$ final $whitespace$ int $whitespace$ size $whitespace$ = $whitespace$ $number$; $newline$ $indentation$ final $whitespace$ string $whitespace$ alphabet $whitespace$ = $whitespace$ $string$; $newline$ $indentation$ final $whitespace$ string $whitespace$ digits $whitespace$ = $whitespace$ $string$; $whitespace$

Original Method	Tokenized Method
1 public void generate() { 2 final int size = 0; 3 final String alphabet = ""; 4 final String digits = ""; 5 }	1 public $whitespace$ void $whitespace$ generate ( ) $whitespace$ { $newline$ $indentation$ final $whitespace$ int $whitespace$ size $whitespace$ = $whitespace$ $number$; $newline$ $indentation$ final $whitespace$ string $whitespace$ alphabet $whitespace$ = $whitespace$ $string$; $newline$ $indentation$ final $whitespace$ string $whitespace$ digits $whitespace$ = $whitespace$ $string$; $whitespace$

Original Method	Tokenized Method
1 public void generate() { 2 final int size = 0; 3 final String alphabet = ""; 4 final String digits = ""; 5 }	1 public $whitespace$ void $whitespace$ generate ( ) $whitespace$ { $newline$ $indentation$ final $whitespace$ int $whitespace$ size $whitespace$ = $whitespace$ $number$; $newline$ $indentation$ final $whitespace$ string $whitespace$ alphabet $whitespace$ = $whitespace$ $string$; $newline$ $indentation$ final $whitespace$ string $whitespace$ digits $whitespace$ = $whitespace$ $string$; $whitespace$

Original Method	Tokenized Method
1 public void generate() { 2 final int size = 0; 3 final String alphabet = ""; 4 final String digits = ""; 5 }	1 public $whitespace$ void $whitespace$ generate ( ) $whitespace$ { $newline$ $indentation$ final $whitespace$ int $whitespace$ size $whitespace$ = $whitespace$ $number$; $newline$ $indentation$ final $whitespace$ string $whitespace$ alphabet $whitespace$ = $whitespace$ $string$; $newline$ $indentation$ final $whitespace$ string $whitespace$ digits $whitespace$ = $whitespace$ $string$; $whitespace$


Data preprocessing
&
Dataset Creation


Data preprocessing
&
Dataset Creation
Code Readability
Enhancement Model

Problems

Lower case output

```
ArrayList<String> myList ≠ arraylist<string> mylist
```

Lower case output

```
ArrayList<String> myList ≠ arraylist<string> mylist
```

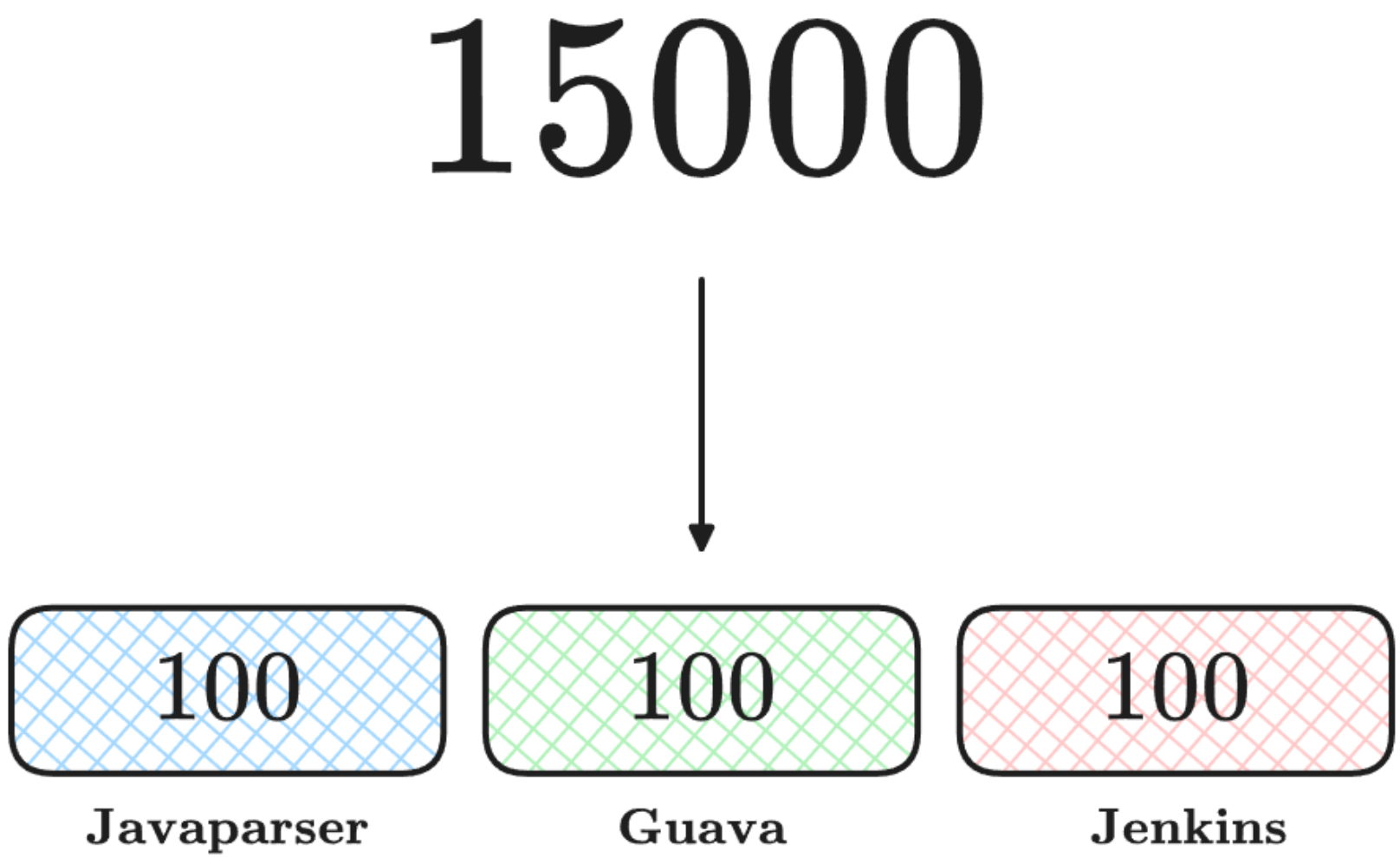
Partially detokenized

```
System.out.println("Hello") ≠ System.out.println($string$)
```

Partially detokenized

```
System.out.println("Hello") ≠ System.out.println($string$)
```


Manual
Evaluation


15000
100
100
100
Javaparser
Guava
Jenkins

Input

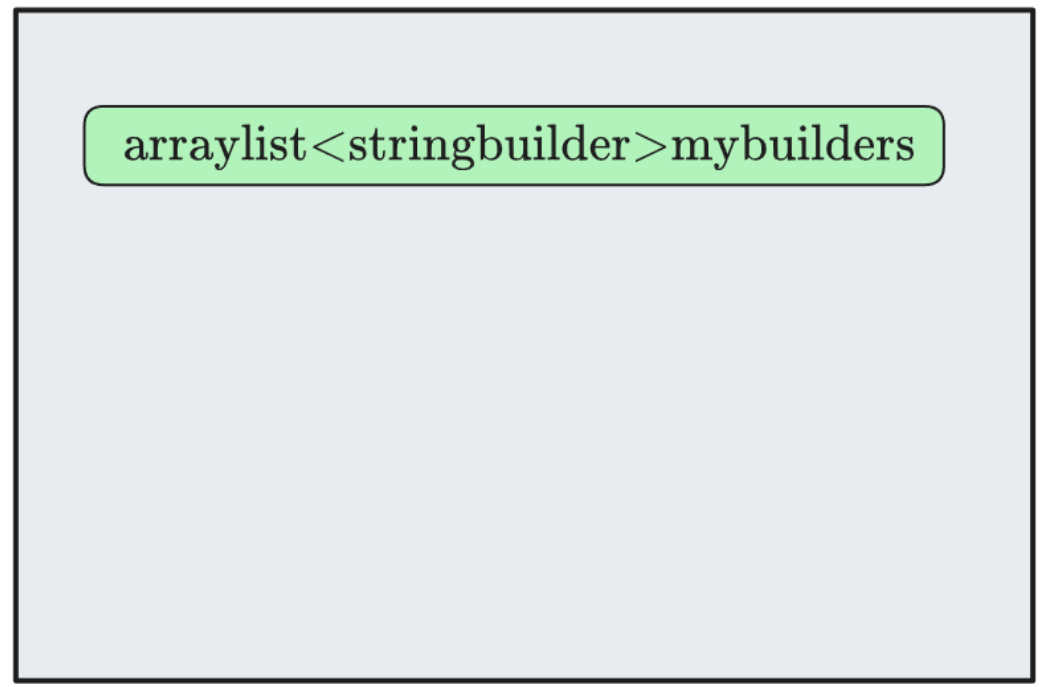
arraylist<stringbuilder>mybuilders

Binary tree

Input

arraylist<stringbuilder>mybuilders

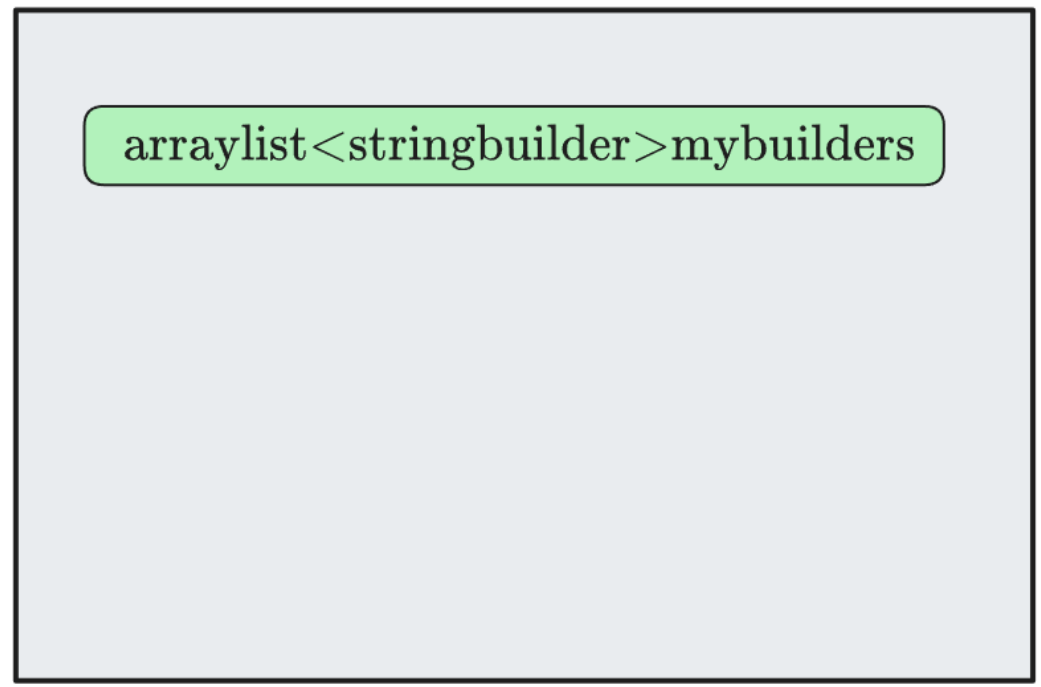
Binary tree



Input

arraylist<stringbuilder>mybuilders

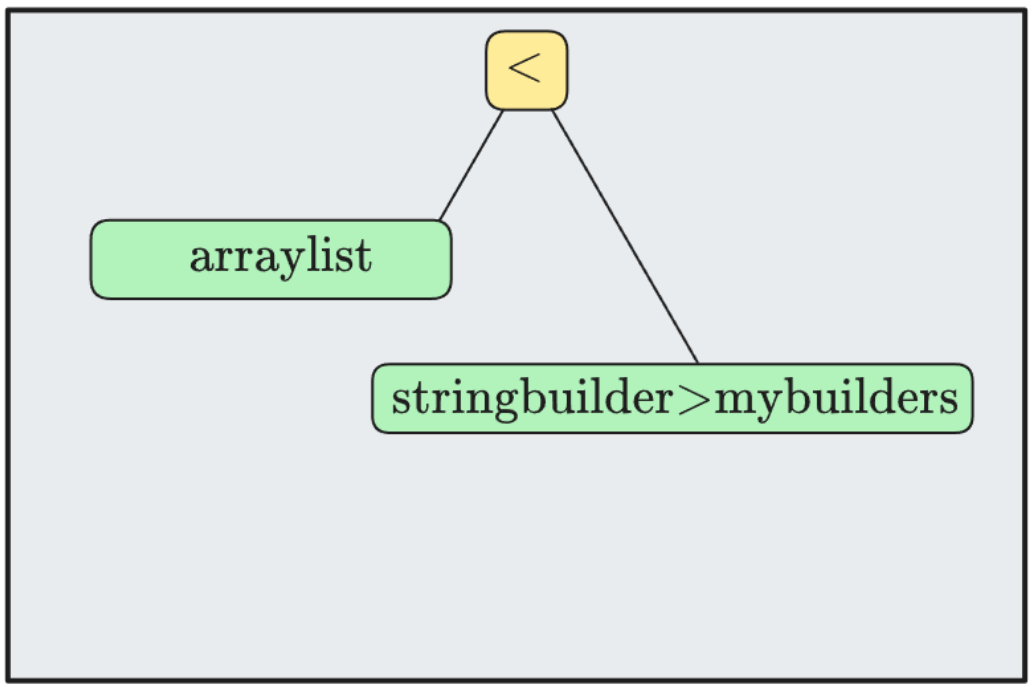
Binary tree



Input

arraylist<stringbuilder>mybuilders

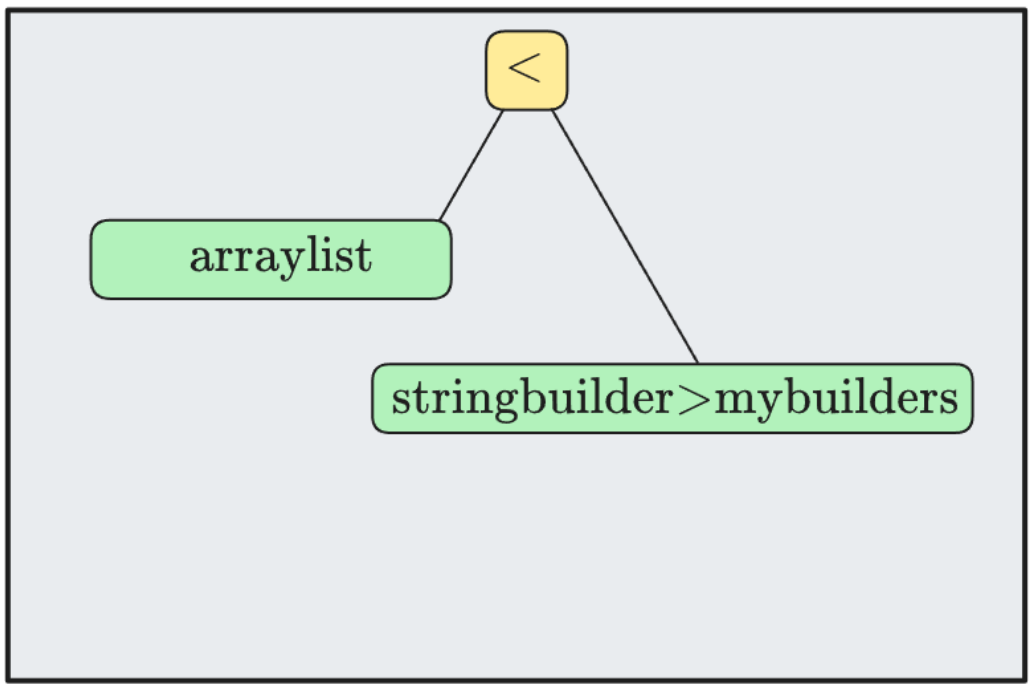
Binary tree



Input

arraylist<stringbuilder>mybuilders

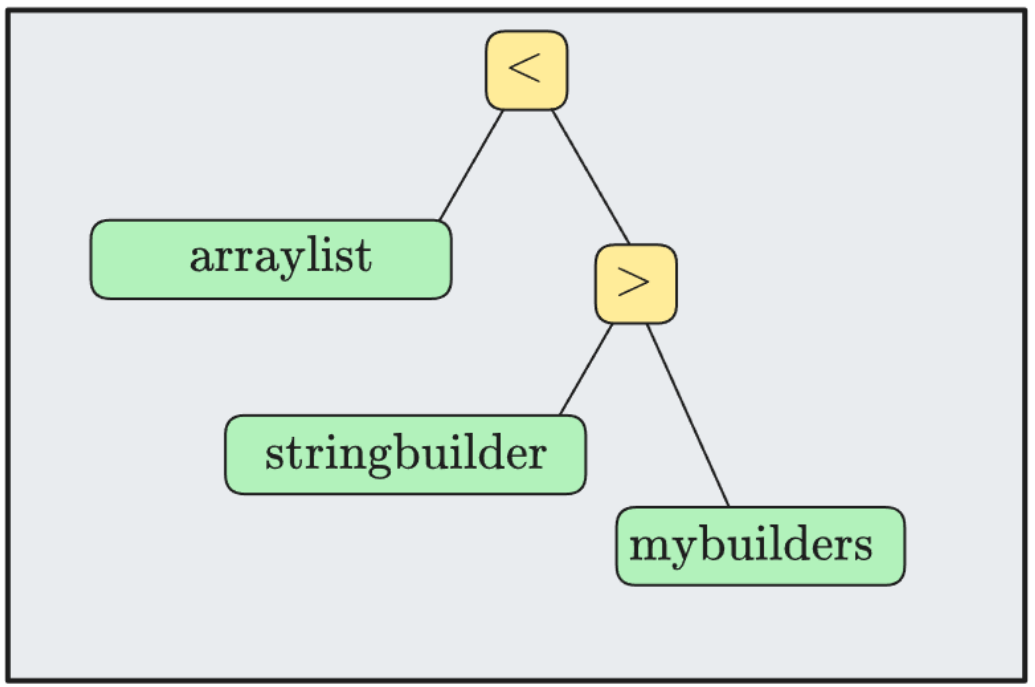
Binary tree



Input

arraylist<stringbuilder>mybuilders

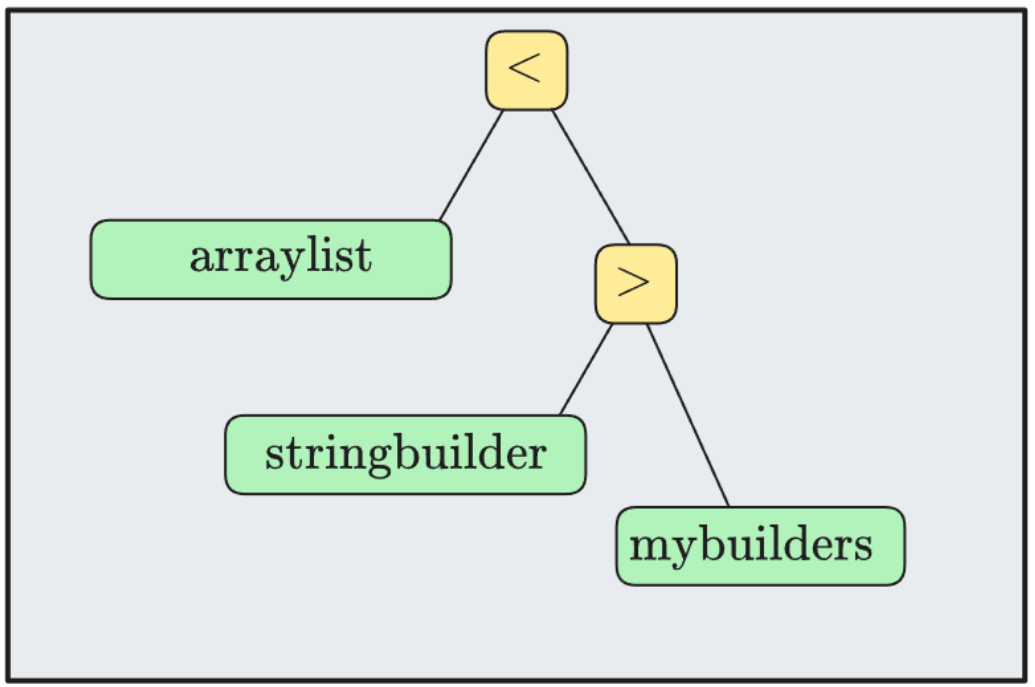
Binary tree

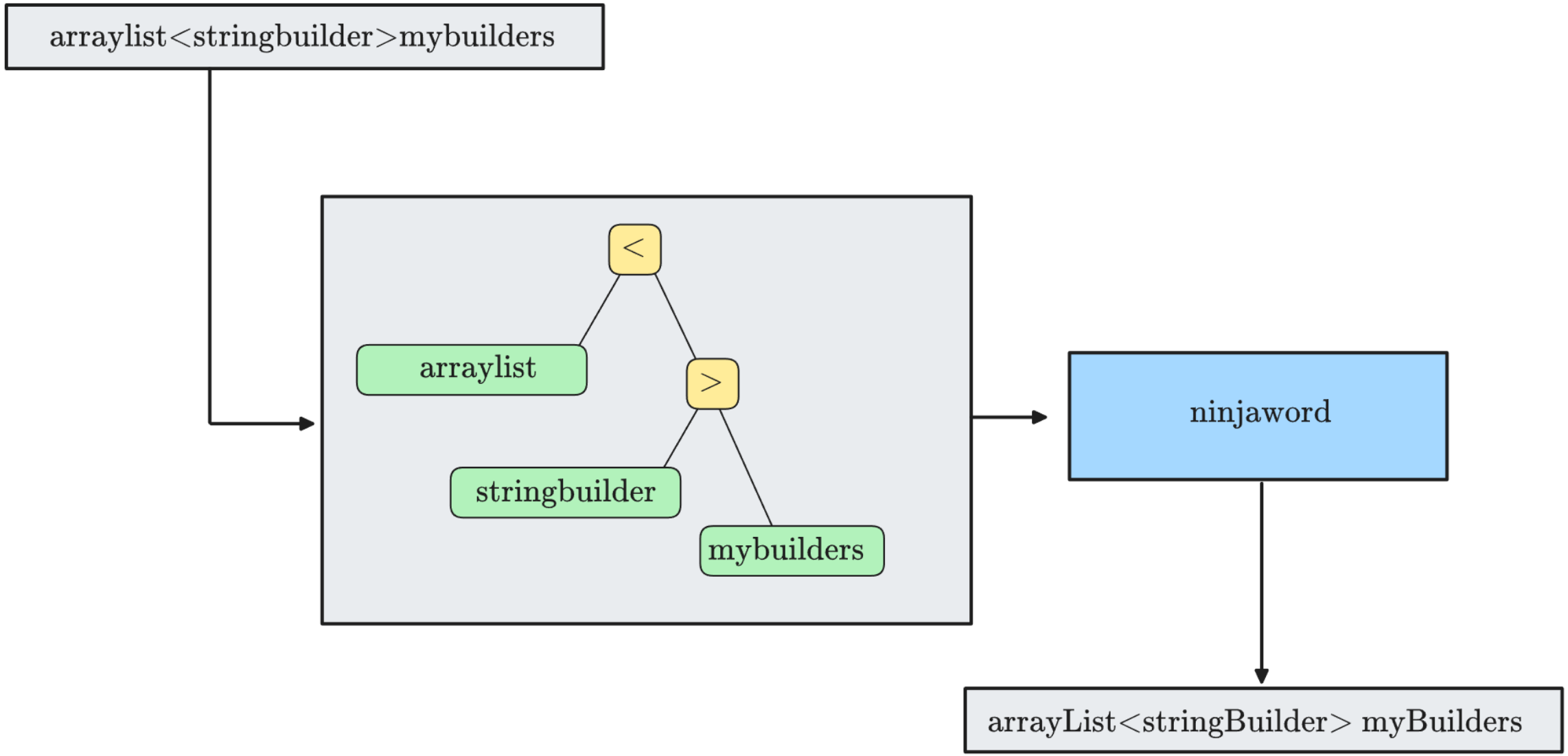


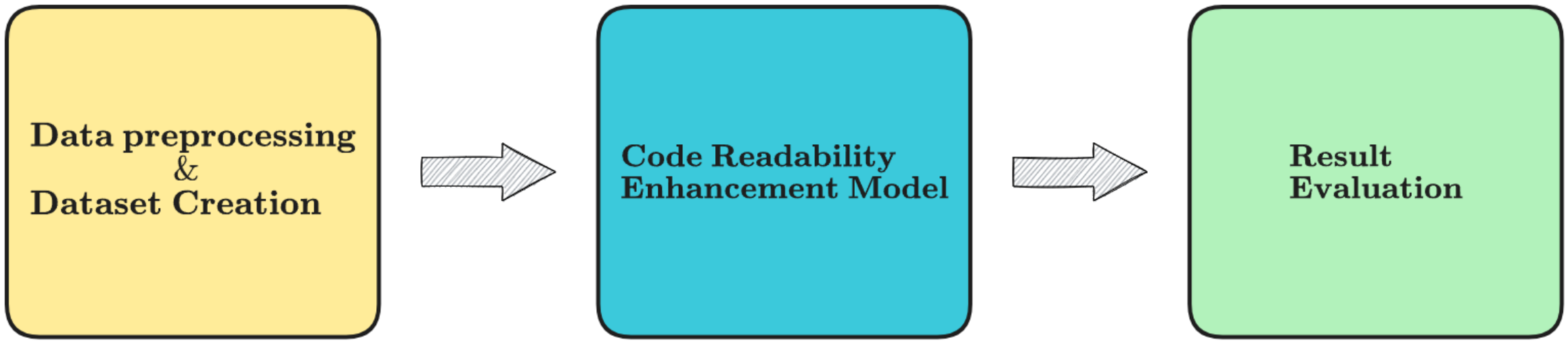
Input

arraylist<stringbuilder>mybuilders

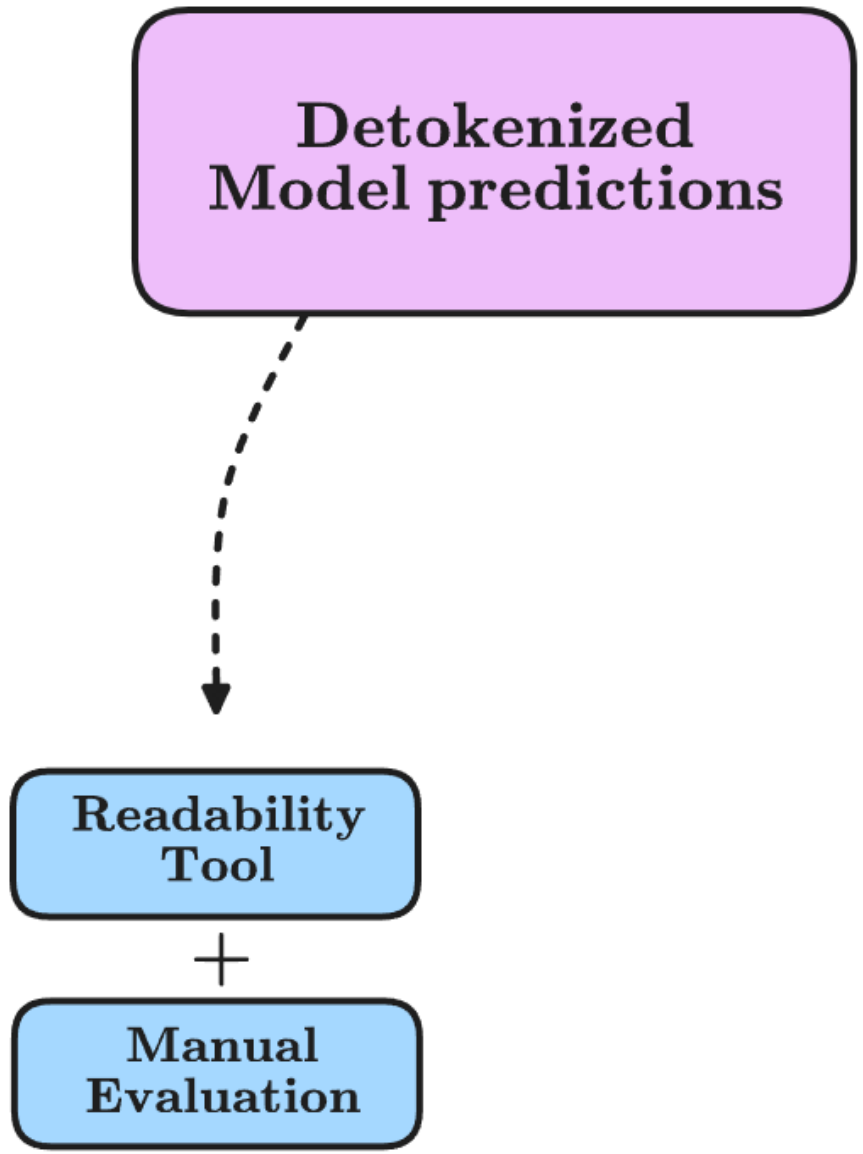
Binary tree

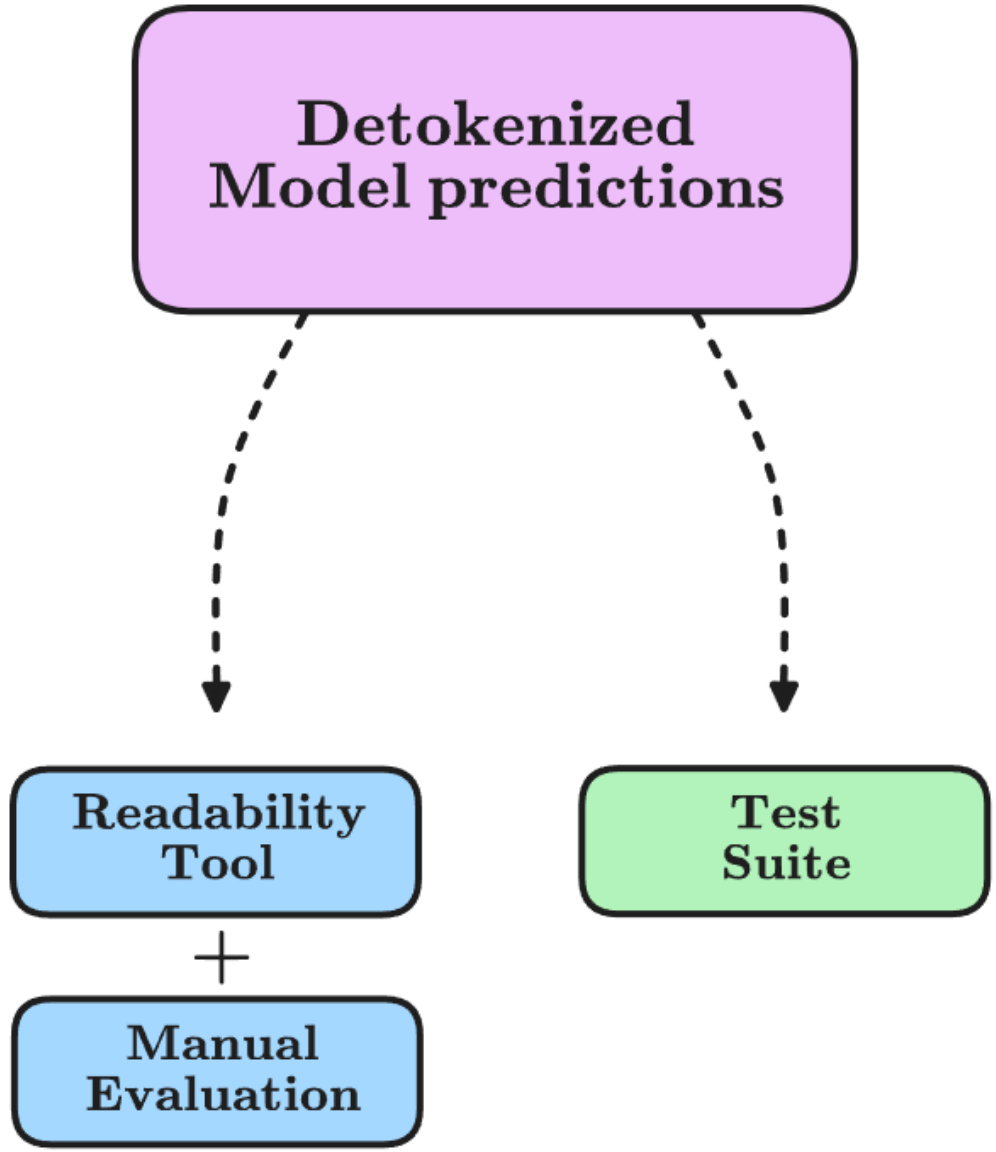



arraylist<stringbuilder>mybuilders
<
arraylist
>
stringbuilder
mybuilders
ninjaword
arrayList<stringBuilder> myBuilders


Data preprocessing
&
Dataset Creation
Code Readability
Enhancement Model
Result
Evaluation

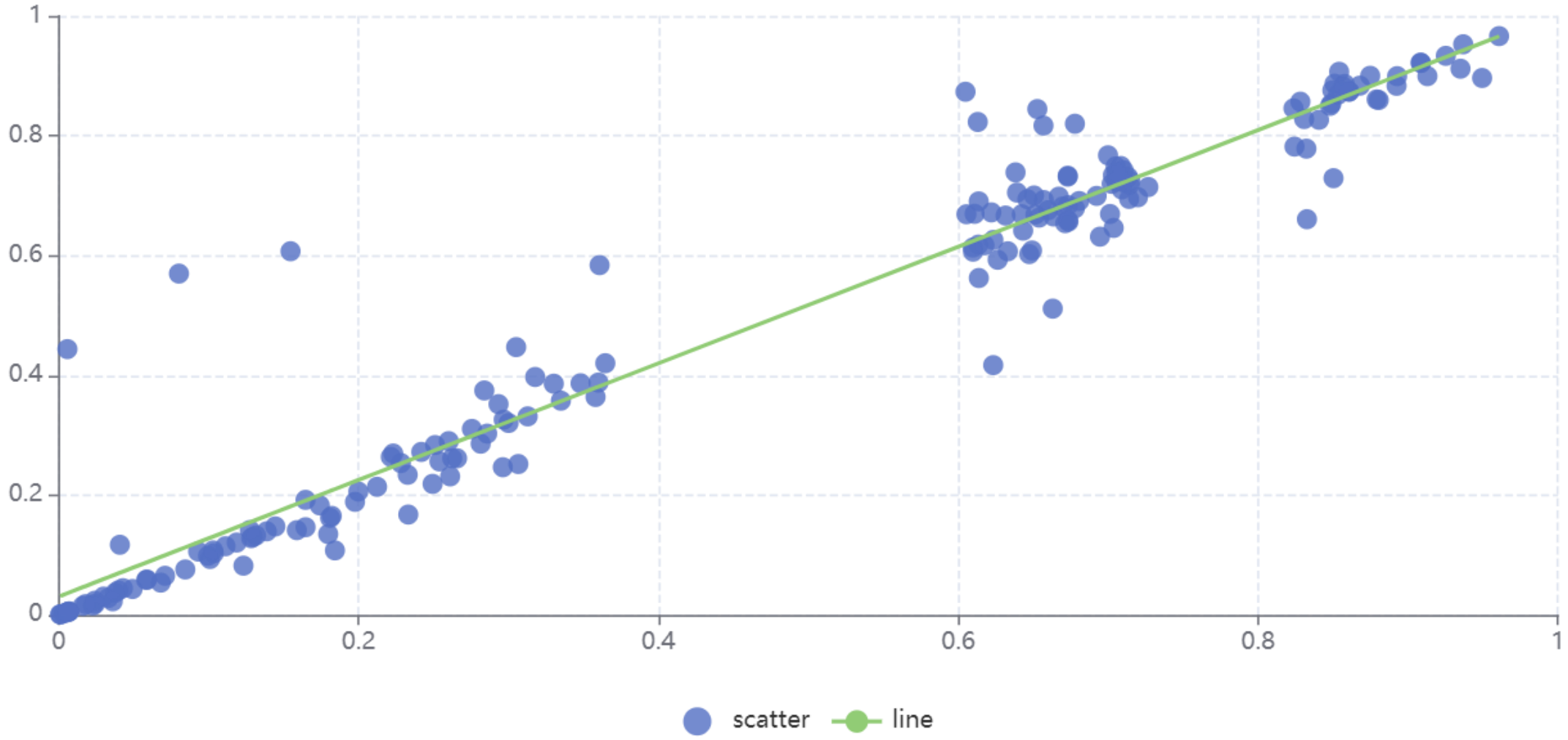
Detokenized Model predictions


Detokenized
Model predictions
Readability
Tool
+
Manual
Evaluation

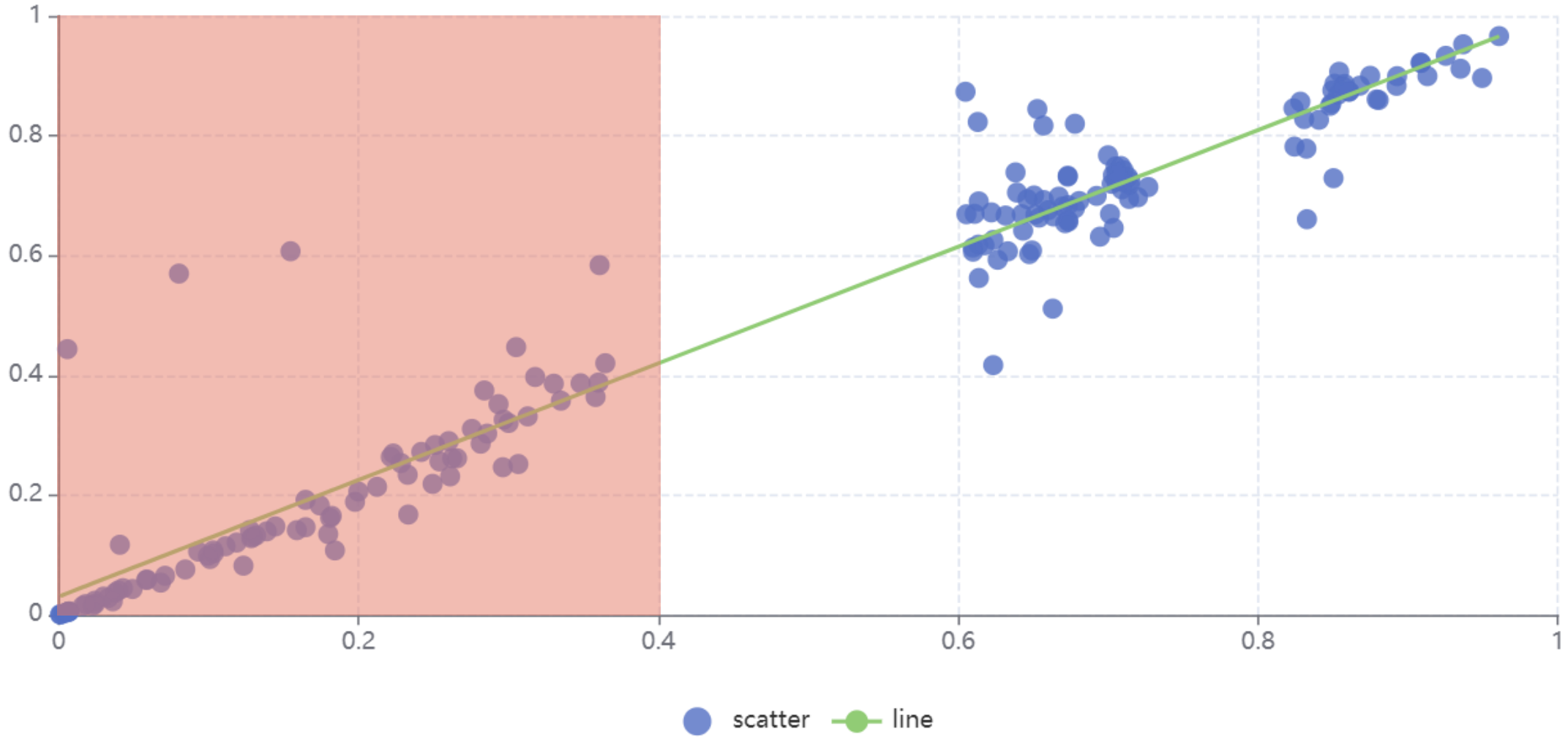

Detokenized
Model predictions
Readability
Tool
+
Manual
Evaluation
Test
Suite

RQ1: What is the ***impact*** of applying ***automatic*** code readability enhancement considering snippets with varying ***levels*** of readability, namely low, medium and high?

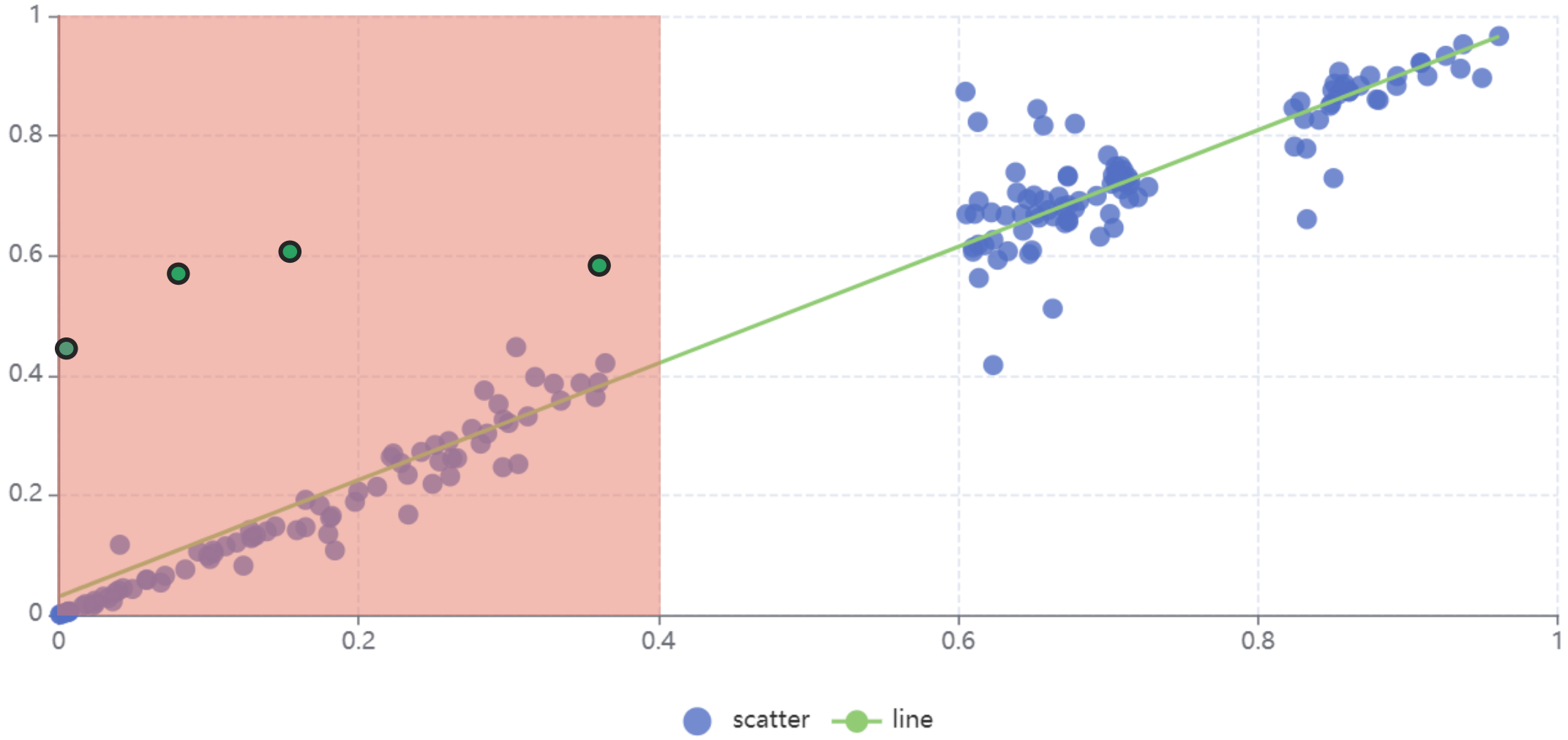
Readability Results



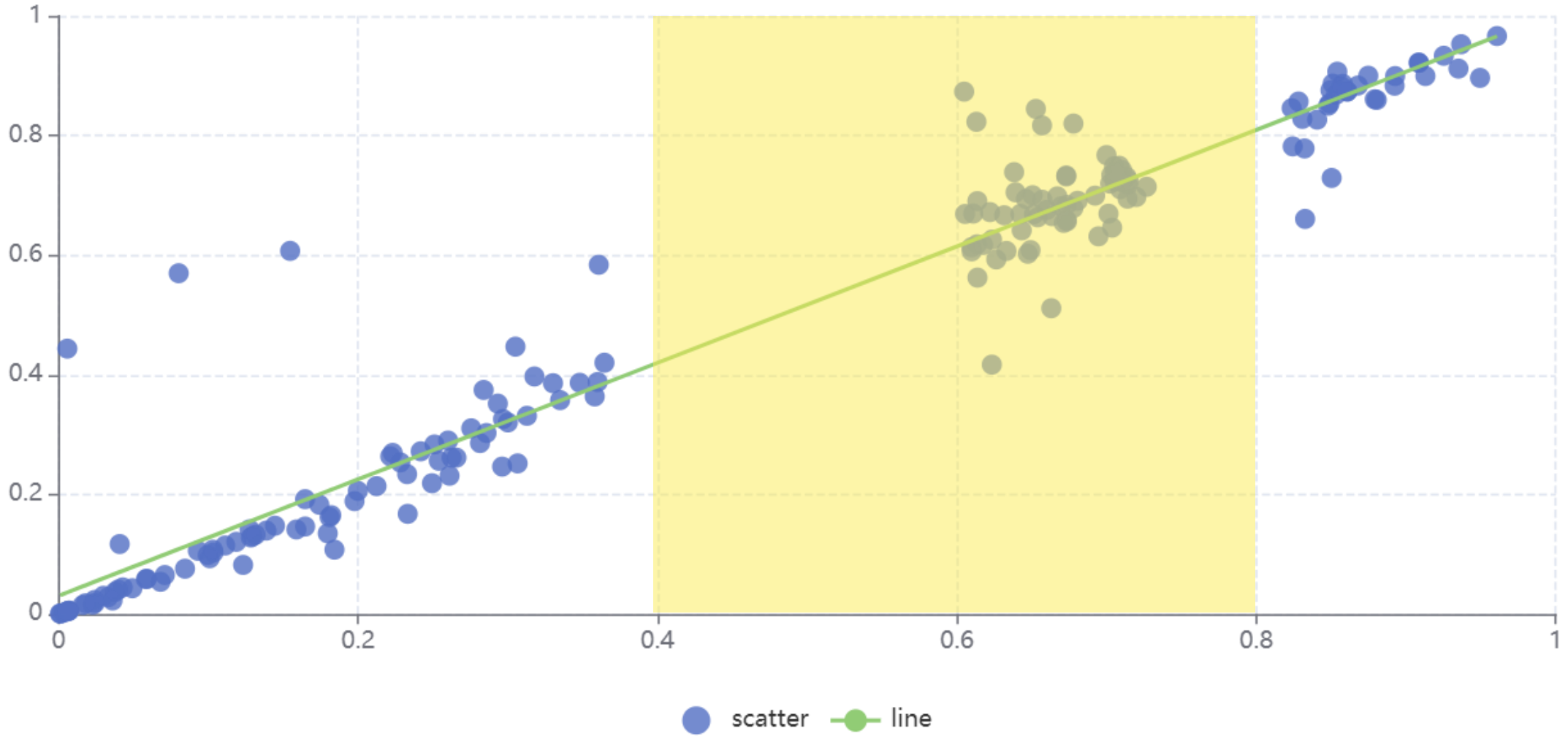
Readability Results



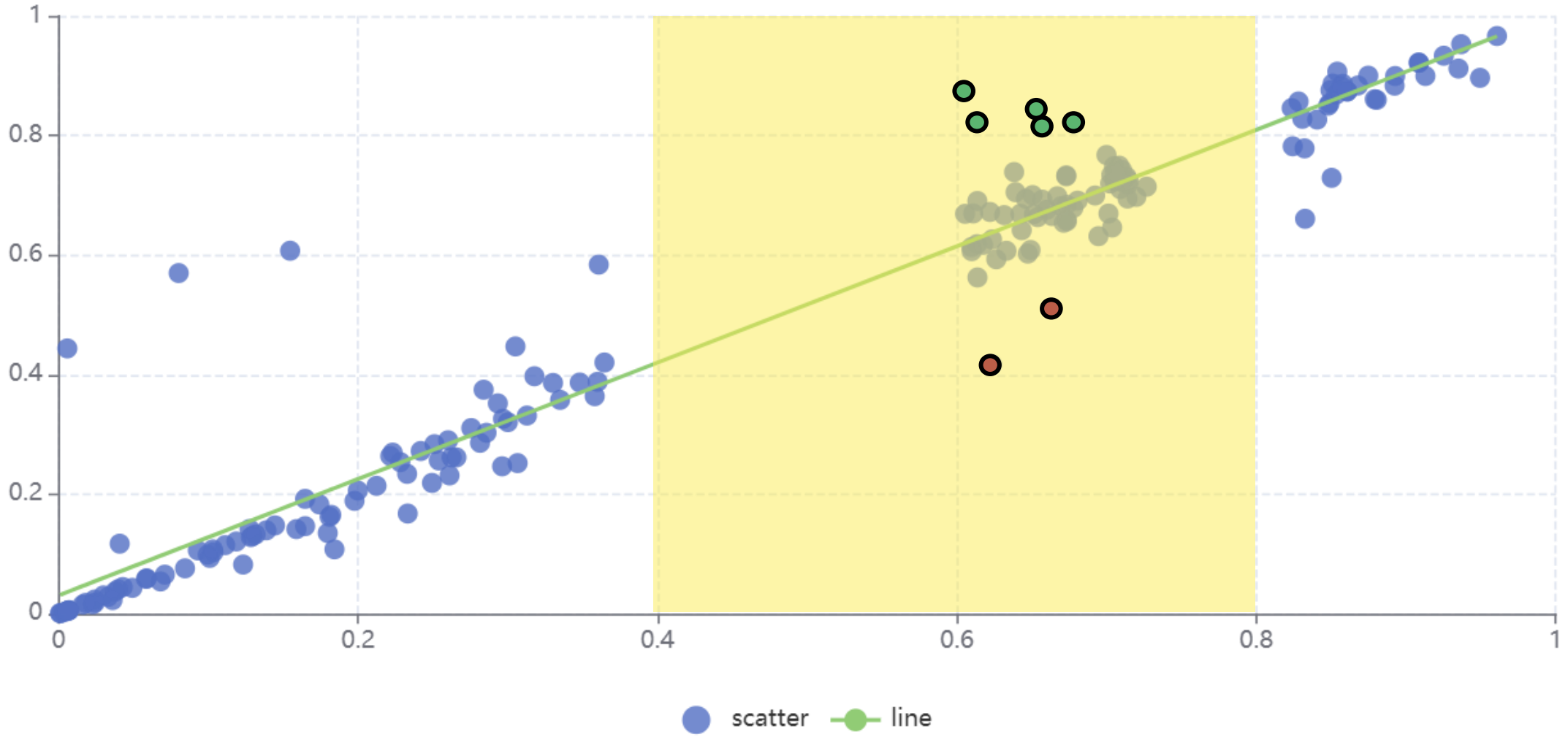
Readability Results



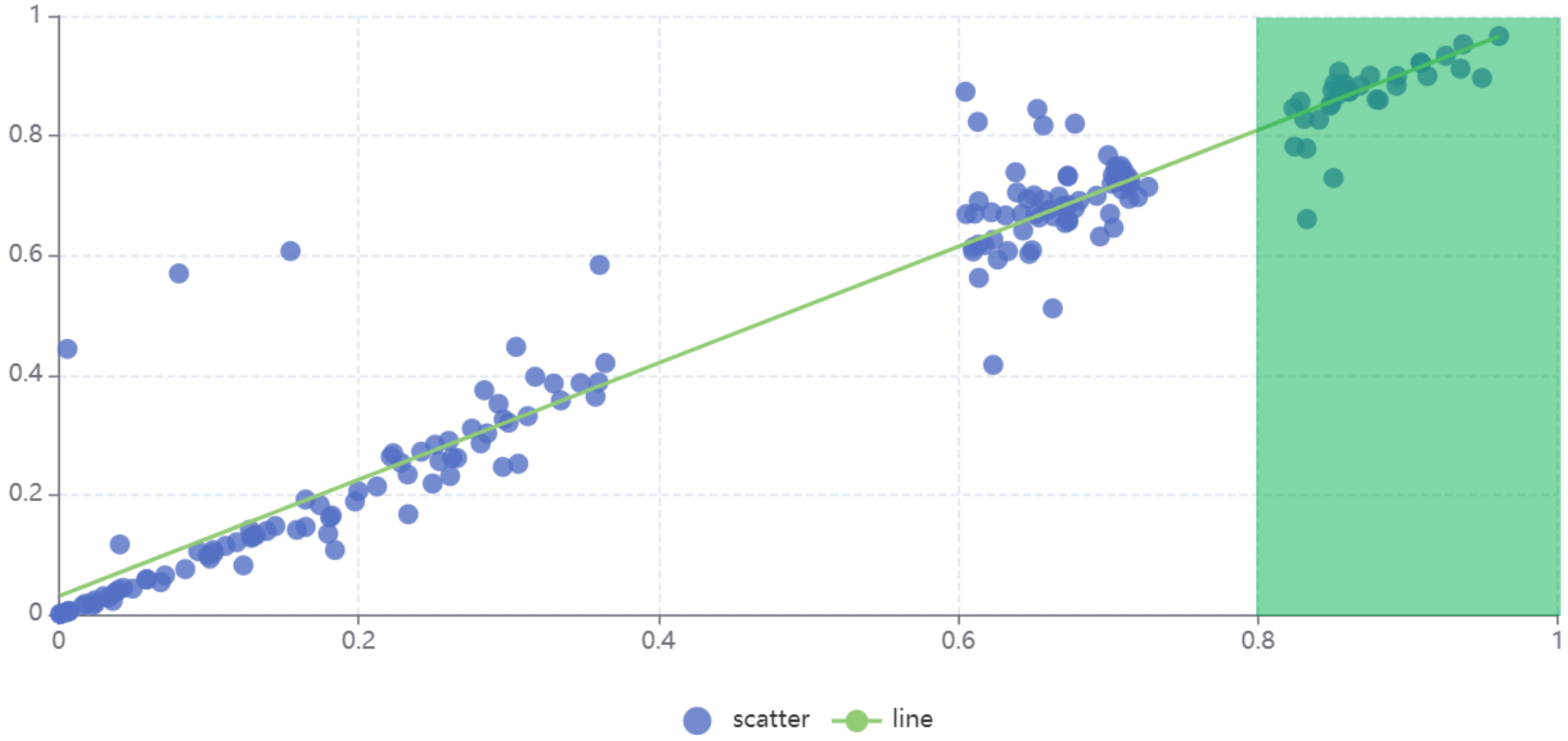
Readability Results



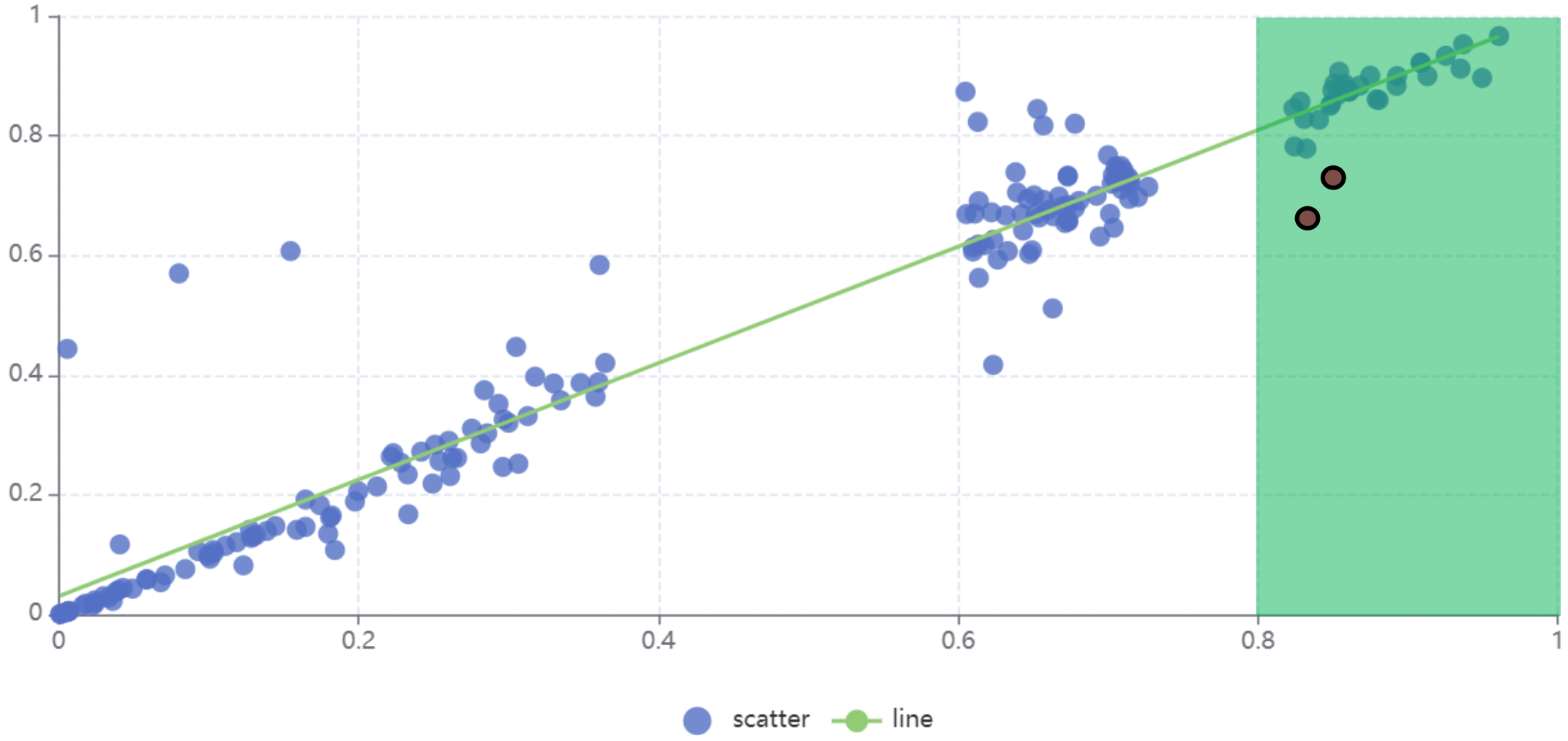
Readability Results



Readability Results



Readability Results



Readability Results

	Low	Medium	High
Increment	8	3	0
Decrement	3	1	1

Readability Results

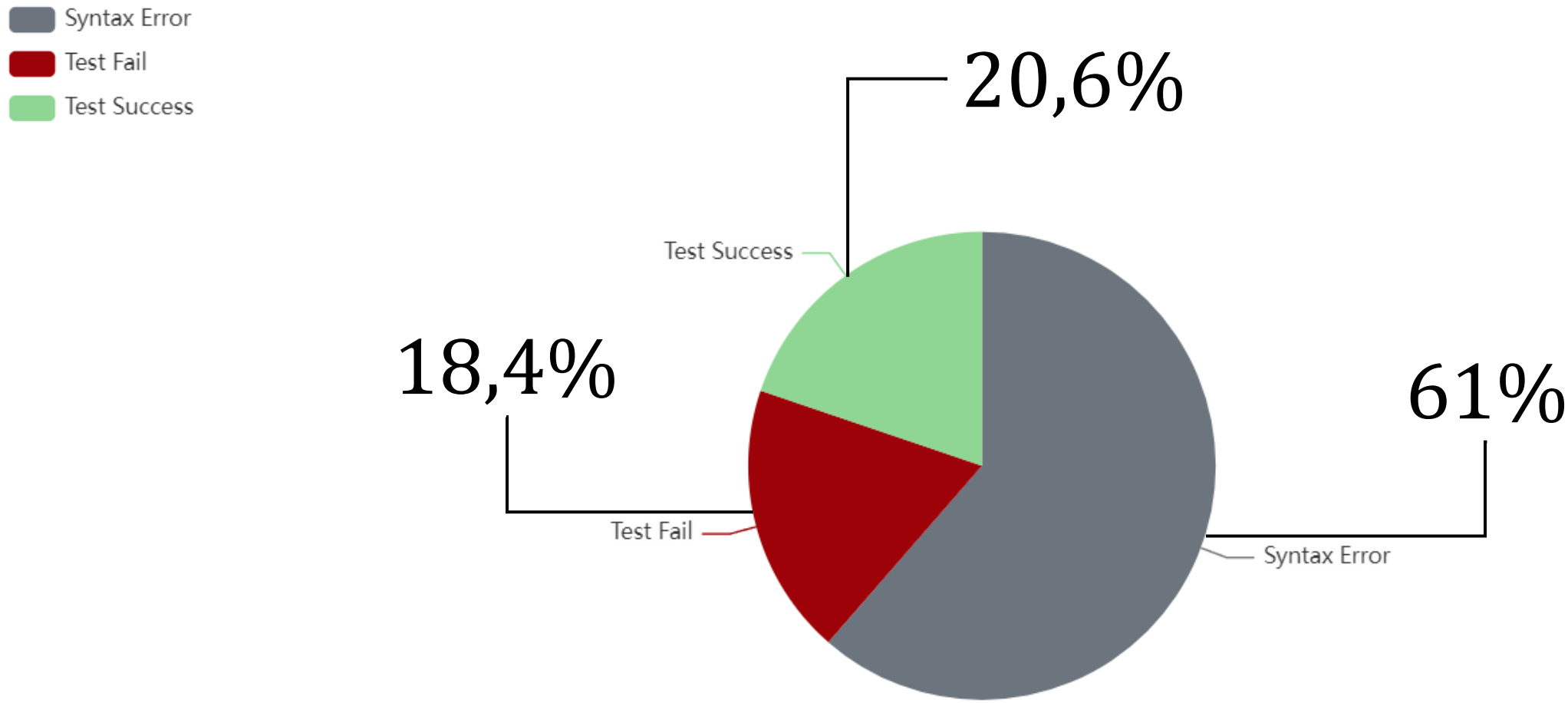
	Low	Medium	High
Increment	8	3	0
Decrement	3	1	1

Readability Results

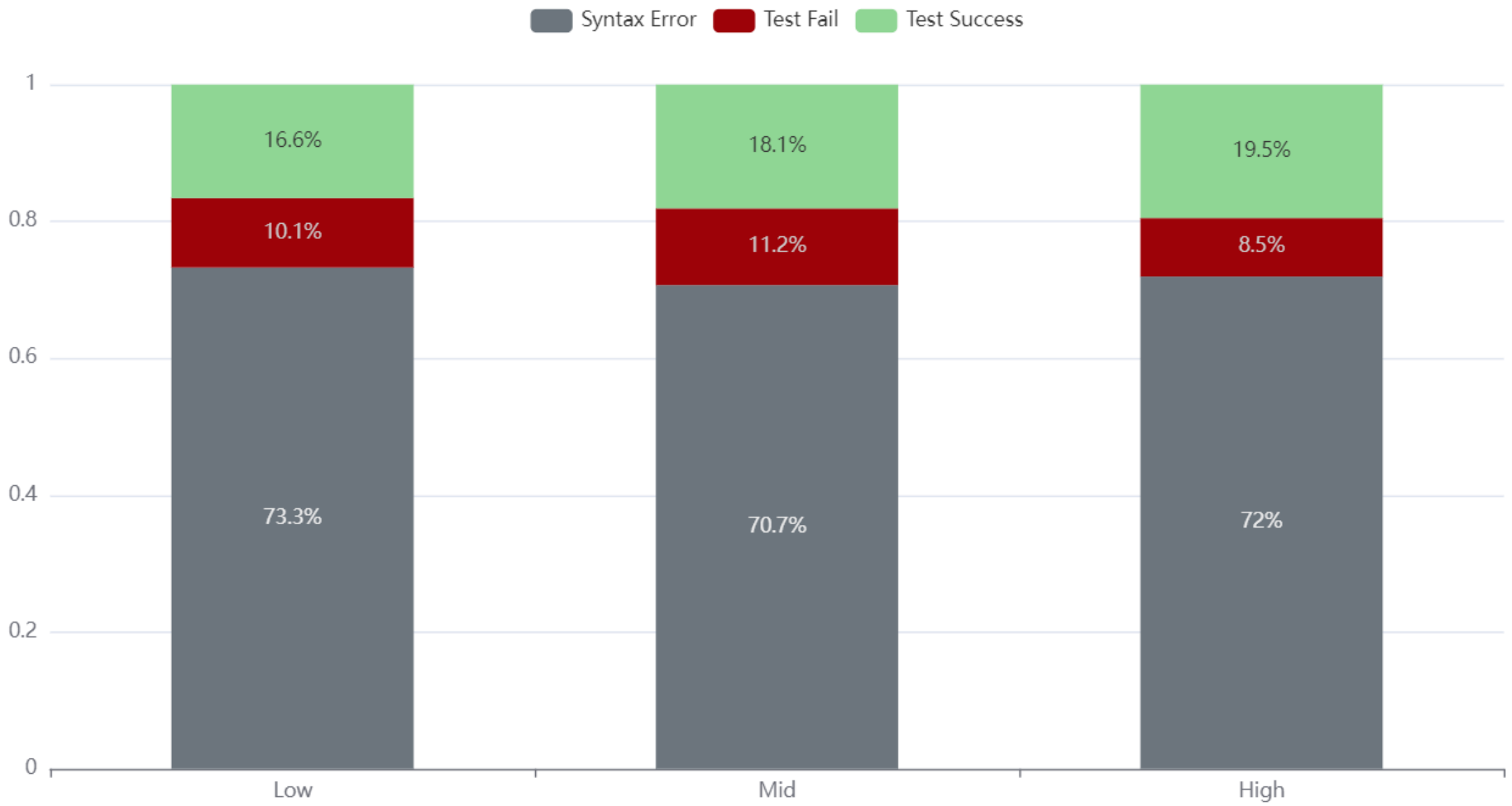
	Low	Medium	High
Increment	8	3	0
Decrement	3	1	1

RQ2: Does readability enhancement through deep learning models ***change code behaviour*** ?

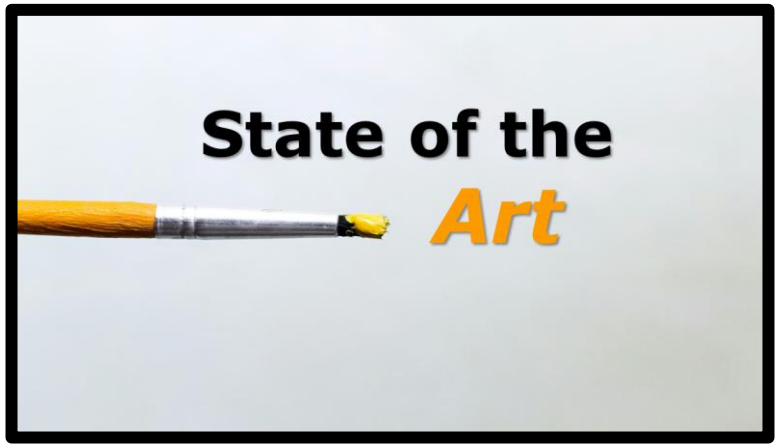
Correctness Results

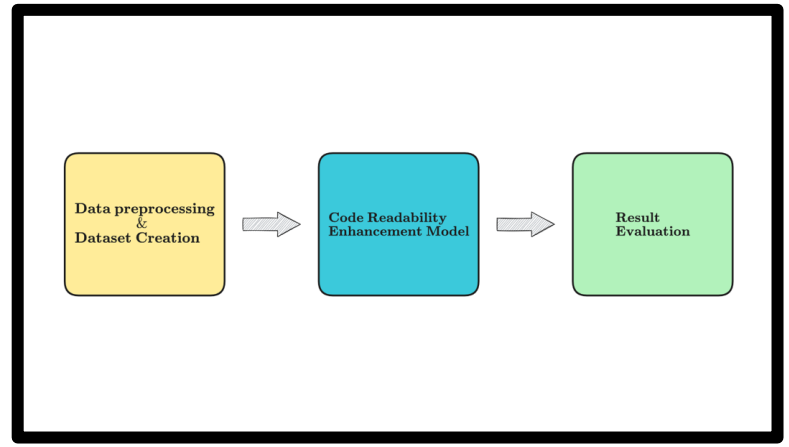


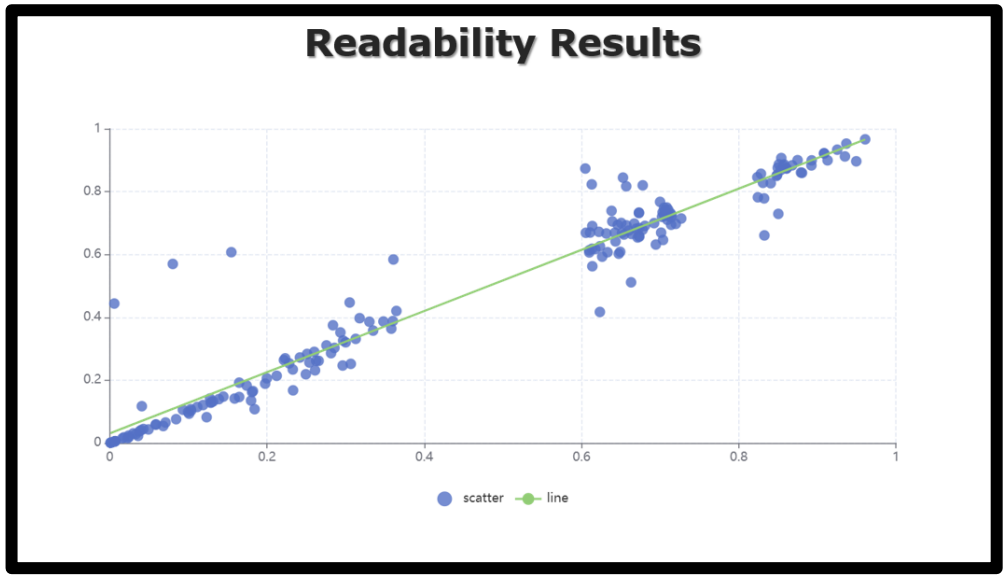
Correctness Results

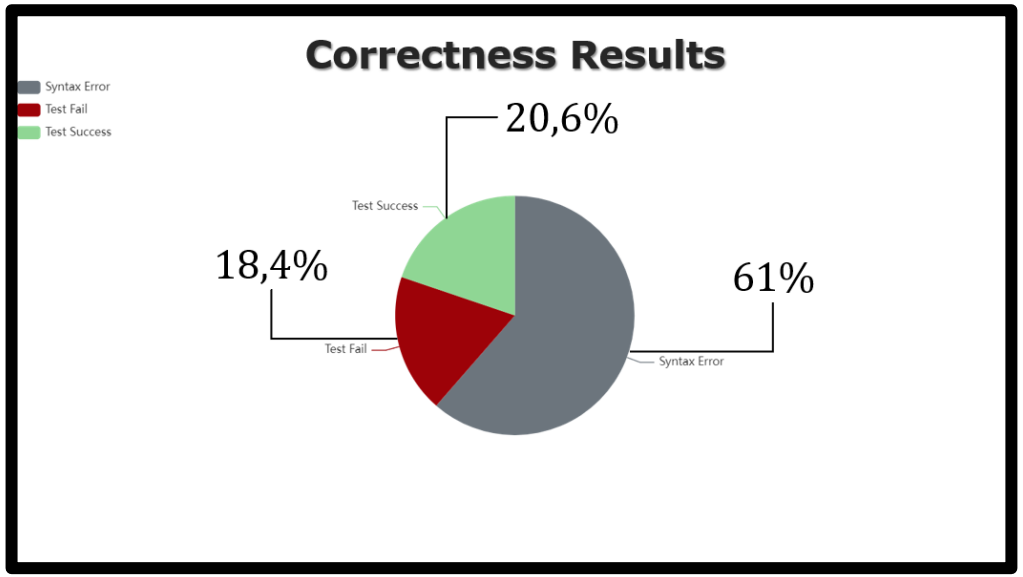



Technical
Debt


State of the
Art


Data preprocessing
&
Dataset Creation
Code Readability
Enhancement Model
Result
Evaluation


Readability Results
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
scatter
line


Correctness Results
Syntax Error
Test Fail
Test Success
20,6%
Test Success
18,4%
61%
Test Fail
Syntax Error

Grazie per l’attenzione