On the robustness of code generation techniques: An empirical study on Github Copilot

Antonio Mastropaolo, *Luca Pascarella, Emanuela Guglielmi, Matteo Ciniselli, Simone Scalabrino, Rocco Oliveto and Gabriele Bavota*

Università della Svizzera italiana
Software Institute
SEART
UNIVERSITAS STUDIORUM MOLISII • SCIENTIARUM AUGMENTIS REIPUBLICAE UTILITATI • MCMLXXXII
erc
European Research Council
Established by the European Commission

“The focus of *software engineering* will move from writing code from scratch to reviewing code written and tested by AI”

—Thomas Zimmermann—

https://www.acm.org/articles/people-of-acm/2022/thomas-zimmermann

“The focus of *software engineering* will move from writing code from scratch to reviewing code written and tested by AI”

—Thomas Zimmermann—

https://www.acm.org/articles/people-of-acm/2022/thomas-zimmermann

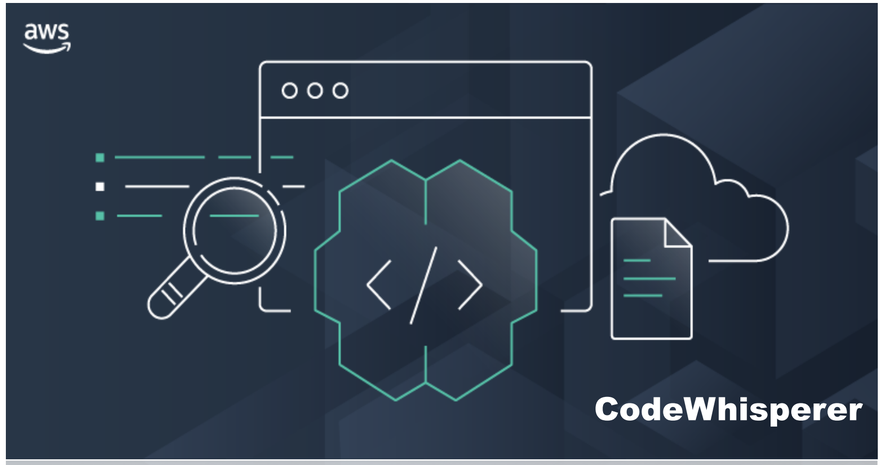
“The focus of *software engineering* will move from writing code from scratch to reviewing code written and tested by AI”

—Thomas Zimmermann—

https://www.acm.org/articles/people-of-acm/2022/thomas-zimmermann

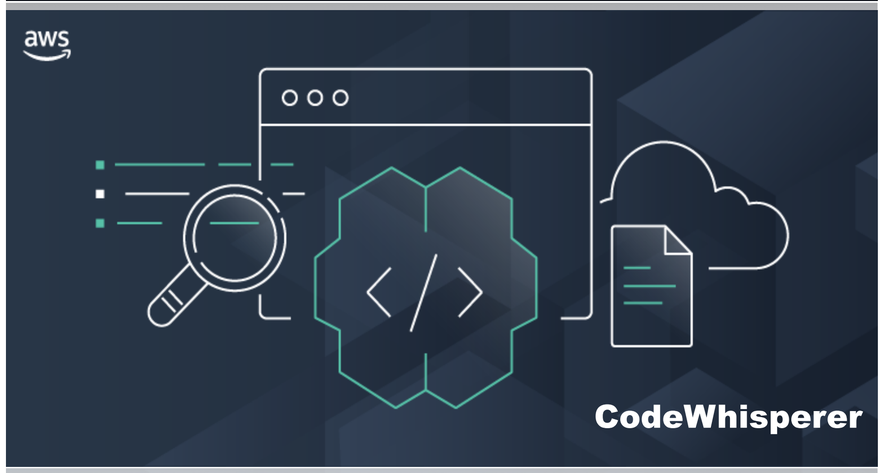
“The focus of *software engineering* will move from writing code from scratch to reviewing code written and tested by AI”

—Thomas Zimmermann—

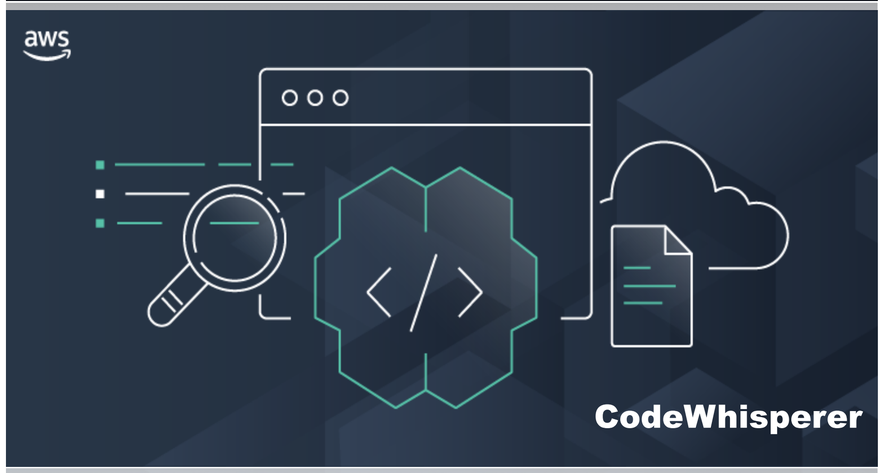


https://www.acm.org/articles/people-of-acm/2022/thomas-zimmermann


GitHub
Copilot


aws
CodeWhisperer


ChatGPT



Code Summarization Task

Summarize this method:

```
private static String toGroupName(String
                                  attribute){
    int pos = attribute.indexOf(';');
    if (pos > -1) {
        attribute = attribute
                        .substring(0, pos);
    }
    return attribute.replace("X", "XX")
                    .replace(“.", "X");
}
```

GitHub Copilot

```
// write the binary search algorithm

public int runBinarySearch(int[] sortedArray, int key, int low,
                           int high) {
```

GitHub Copilot

```
// write the binary search algorithm

public int runBinarySearch(int[] sortedArray, int key, int low,
                           int high) {
    int index = Integer.MAX_VALUE;
    while (low <= high) {
        int mid = low  + ((high - low) / 2);
        if (sortedArray[mid] < key) {
            low = mid + 1;
        } else if (sortedArray[mid] > key) {
            high = mid - 1;
        } else if (sortedArray[mid] == key) {
            index = mid;
            break;
        }
    }
    return index;
}
```

Original Code Description

```
write the binary search algorithm
```

Semantically Equivalent Code Descriptions

```
Write the binary search algorithm for
an array of Integer

Implements the binary search algorithm
for a vector of Integer
```

Original Code Description

```
write the binary search algorithm
```

Semantically Equivalent Code Descriptions

```
Write the binary search algorithm for
an array of Integer

Implements the binary search algorithm
for a vector of Integer
```

On the Robustness of Code Generation Techniques: An Empirical Study on GitHub Copilot

Antonio Mastropaolo*, Luca Pascarella*, Emanuela Guglielmi†, Matteo Ciniselli*
Simone Scalabrino†, Rocco Oliveto†, Gabriele Bavota*
*SEART @ Software Institute, Università della Svizzera italiana (USI), Switzerland
†University of Molise, Italy

***Abstract*—Software engineering research has always being concerned with the improvement of code completion approaches, which suggest the next tokens a developer will likely type while coding. The release of GitHub Copilot constitutes a big step forward, also because of its unprecedented ability to automatically generate even entire functions from their natural language description. While the usefulness of Copilot is evident, it is still unclear to what extent it is robust. Specifically, we do not know the extent to which semantic-preserving changes in the natural language description provided to the model have an effect on the generated code function. In this paper we present an empirical study in which we aim at understanding whether *different but semantically equivalent natural language descriptions result in the same recommended function*. A negative answer would pose questions on the robustness of deep learning (DL)-based code generators since it would imply that developers using different wordings to describe the same code would obtain different recommendations. We asked Copilot to automatically generate 892 Java methods starting from their original Javadoc description. Then, we generated different semantically equivalent descriptions for each method both manually and automatically, and we analyzed the extent to which predictions generated by Copilot changed. Our results show that modifying the description results in different code recommendations in ∼46% of cases. Also, differences in the semantically equivalent descriptions might impact the correctness of the generated code (±28%).**

***Index Terms*—Empirical Study, Recommender Systems**

I. Introduction

One of the long lasting dreams in software engineering research is the automated generation of source code. Towards this goal, several approaches have been proposed. The first attempts targeted the relatively simpler problem of code completion, that has been tackled exploiting historical information [50], coding patterns mined from software repositories [21], [42], [56], [10], [41], [45], [19] and, more recently, Deep Learning (DL) models [62], [27], [29], [8], [53], [15].

The release of *GitHub Copilot* [14] pushed the capabilities of these tools to whole new levels. The large-scale training performed on the OpenAI's Codex model allows Copilot to not limit its recommendations to few code tokens/statements the developer is likely to write: Copilot is able to automatically synthesize entire functions just starting from their signature and natural language descriptions.

This new generation of code recommender systems has the potential to change the way in which developers write code [18] and comes with a number of questions concerning how to effectively exploit them to maximize developers' productivity.

Intuitively, the ability of the developer to provide "proper" inputs to the model will become central to boost the effectiveness of its recommendations. In the concrete example of GitHub Copilot, the natural language description provided to the model to automatically generate a code function could substantially influence the model output. This means that two developers providing different natural language descriptions for the same function they would like to automatically generate could receive two different recommendations. While this would be fine in case the two descriptions are actually different in the semantics of what they describe, receiving different recommendations for *semantically equivalent natural language descriptions* would pose questions on the robustness and usability of DL-based code recommenders.

This is the main research question we investigate in this paper: We study the extent to which different semantically equivalent natural language descriptions of a function result in different recommendations (*i.e.,* different synthesized functions) by GitHub Copilot. The latter is selected as representative of DL-based code recommenders since it is the *de facto* state-of-the-art tool when it comes to code generation.

We collected from an initial set of 1,401 open source projects a set of 892 Java methods that are (i) accompanied by a Doc Comment for the Javadoc tool, and (ii) exercised by a test suite written by the project's contributors. Then, as done in the literature [23], [32], we considered the first sentence of the Doc Comments as a "natural language description" of the method. We refer to this sentence as the "*original*" description.

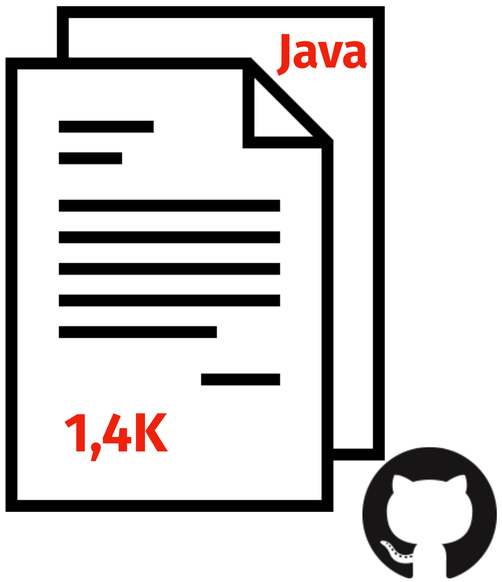
We preliminarily checked whether existing automated paraphrasing techniques are suitable for robustness testing, *i.e.,* if they can be used to create semantically equivalent descriptions of the methods to generate. We validated two state-of-the-art approaches in this scenario: PEGASUS [66], a DL-based paraphrasing tool, and Translation Pivoting (TP), a heuristic-based approach. We used both techniques to generate a paraphrase for each *original* description in our dataset. Then, we manually inspected the obtained paraphrases and classified them as semantically equivalent or not. We obtained positive results for both the approaches, with TP being the best performing one with 77% of valid paraphrases.

Then, to answer our main research question, we generated different paraphrases for each *original* description.

Experimental Design

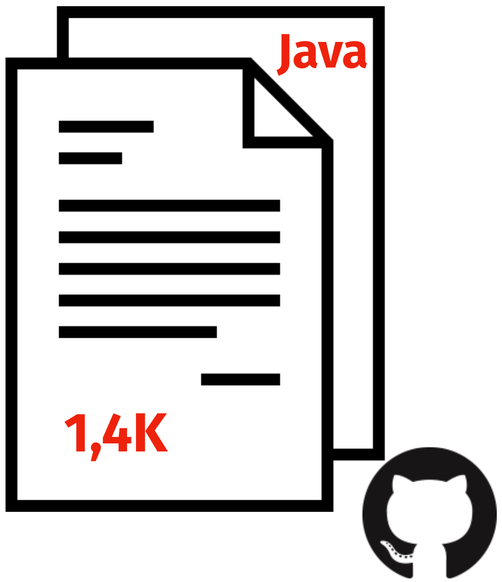
1

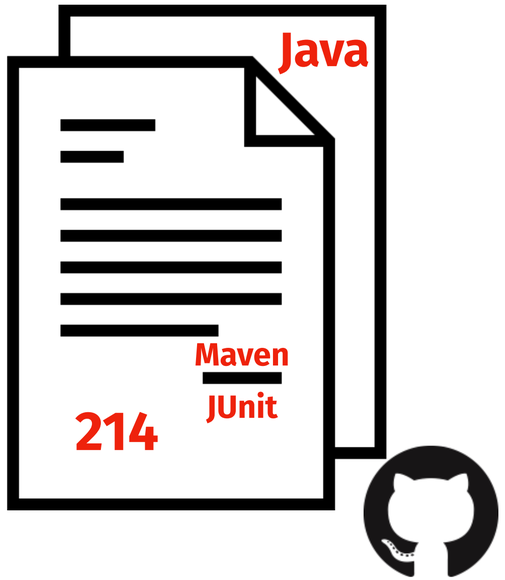
Dataset Construction



1 2 3 4

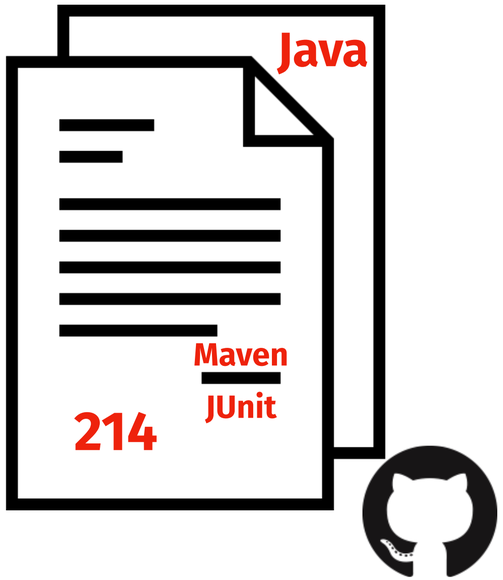
Dataset Construction





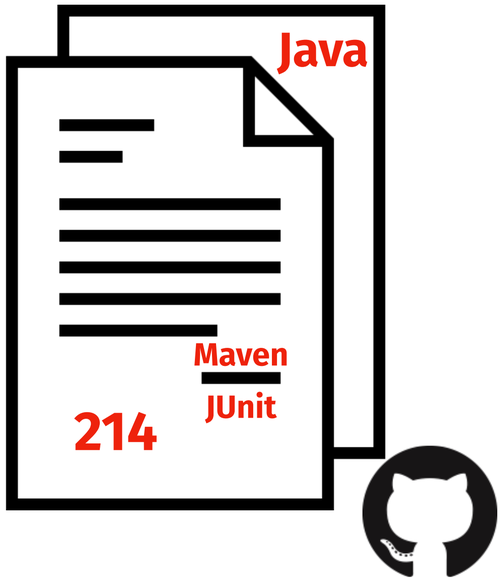
1 2 3 4

Dataset Construction




```java
/* Translates a attribute
name into a name suitable
for a named capturing
group. */

private static String
  toGroupName(String attribute){
  int pos = attribute.indexOf(';');
  if (pos > -1) {
      attribute =
        attribute.substring(0, pos);
    }
    return attribute.
       replace("X", "XX").
       replace(".", "X");
}
```

Dataset Construction



1 2 3 4

	Code Description	Method
1	*Translates an attribute name into a name suitable for a named capturing group.*	`private static String toGroupName(String attribute){ int pos = attribute.indexOf(';'); if (pos > -1) { attribute = attribute.substring(0, pos); } return attribute.replace("X", "XX") .replace(“.", "X"); }`
2	*Returns the time in millisecond until the next ticket.*	`public long timeout(){ if (tickets.isEmpty()) { return -1; } sortIfNeeded(); Ticket first = tickets.get(0); return first.start - now() + first.delay; }`
...		...

892

Experimental Design

2

Original Code Description

1	*Translates an attribute name into a name suitable for a named capturing group.*
2	*Returns the time in millisecond until the next ticket.*
...	

Paraphrased Code Description

1	*Given an attribute name as input, it translates an attribute name into a name suitable for a named capturing group.*
2	*The time is returned in millisecond until the next ticket.*
...	892

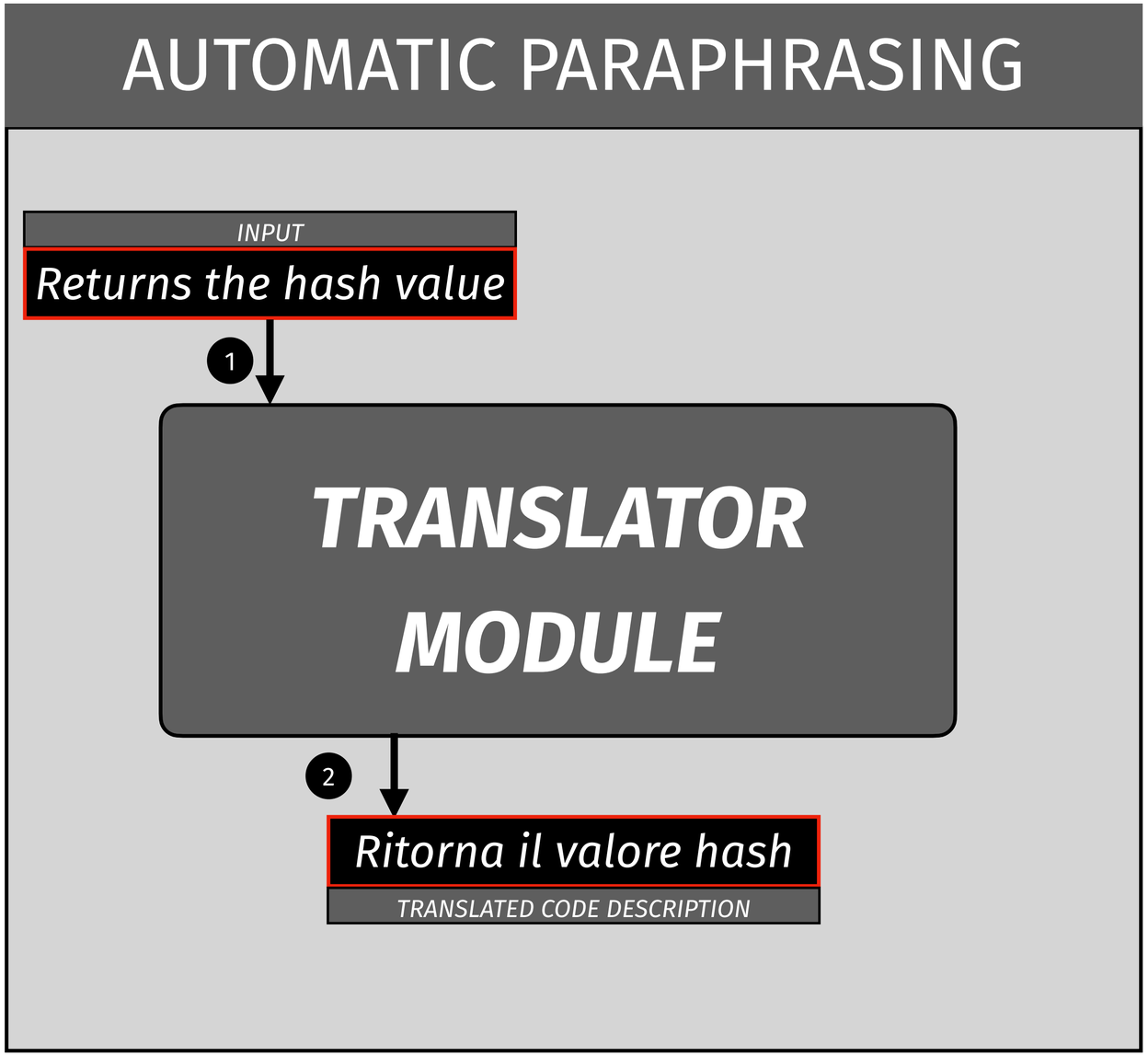
1 *Translation Pivoting*

2 *DL-based paraphrasing*

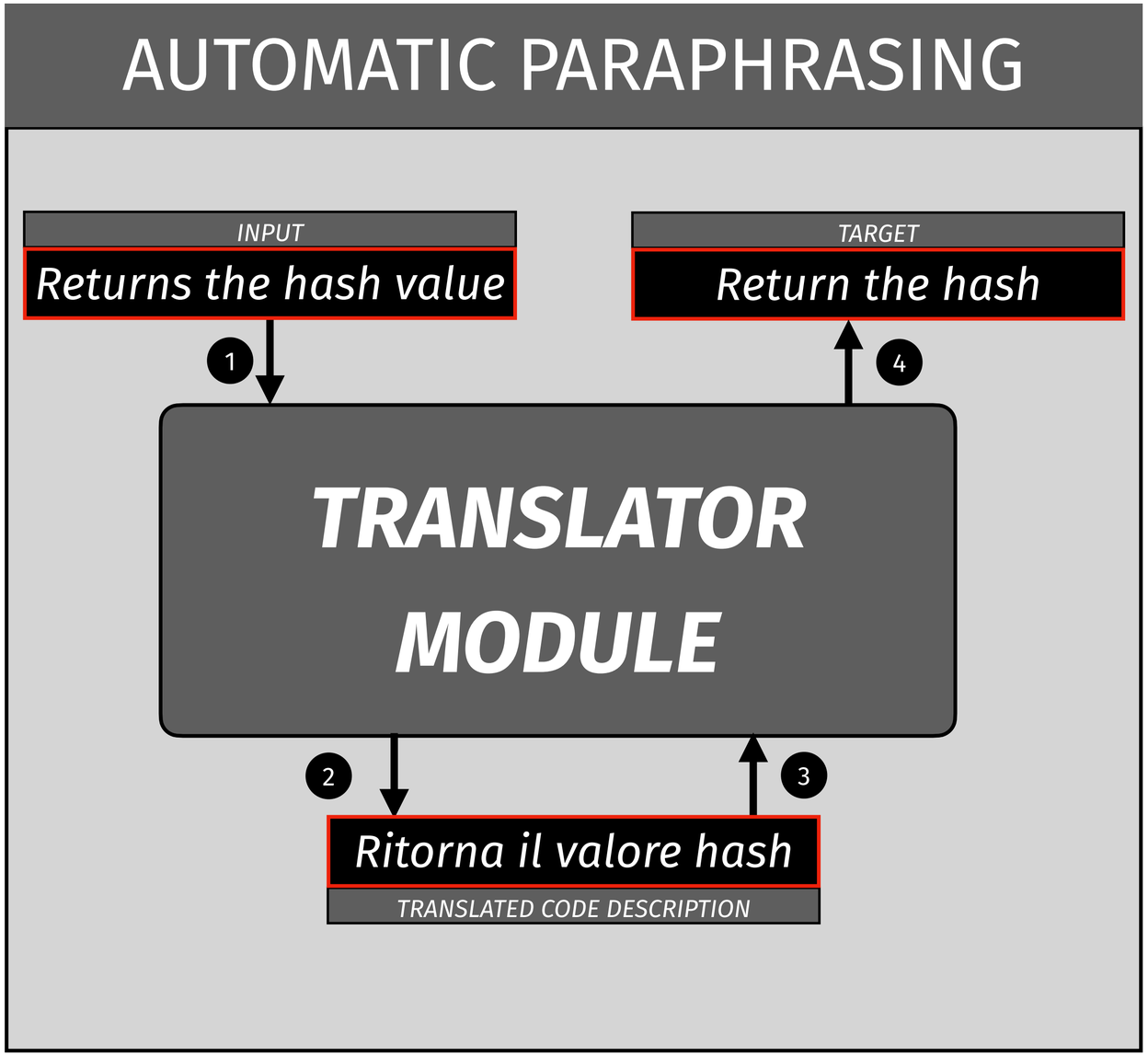
AUTOMATIC

3 *Manually Paraphrased*

Different, yet semantically equivalent

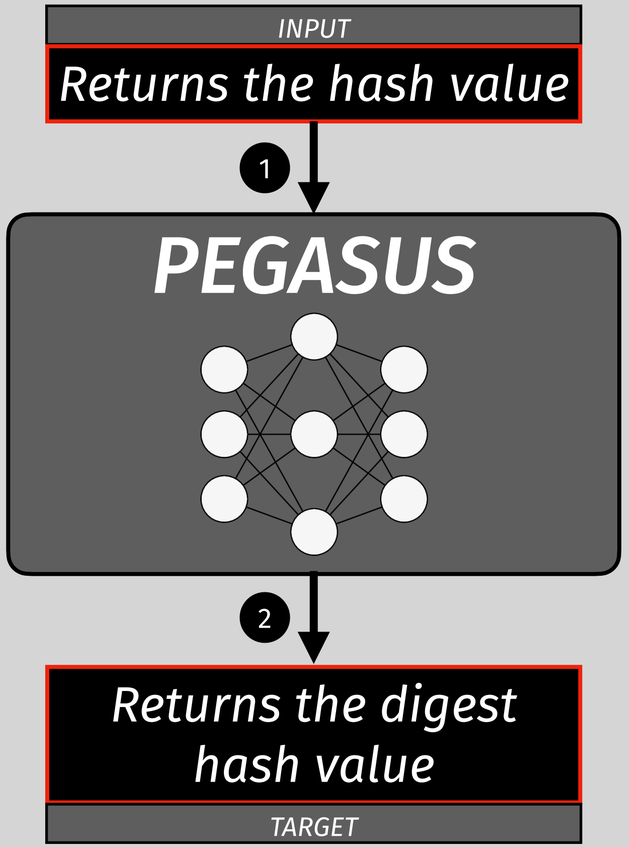


Translation Pivoting



Translation Pivoting

AUTOMATIC PARAPHRASING



DL-based Model

Experimental Design

3

```
Utility to determine if the specified mass is the major isotope for the given atomic number.
private boolean isMajorIsotope(int number, int mass){
    …
}
```

ORIGINAL

```
Given an atomic number, the utility can determine if a specific mass is the major isotope for that atomic number.
private boolean isMajorIsotope(int number, int mass){
    …
}
```

MANUALLY PARAPH.

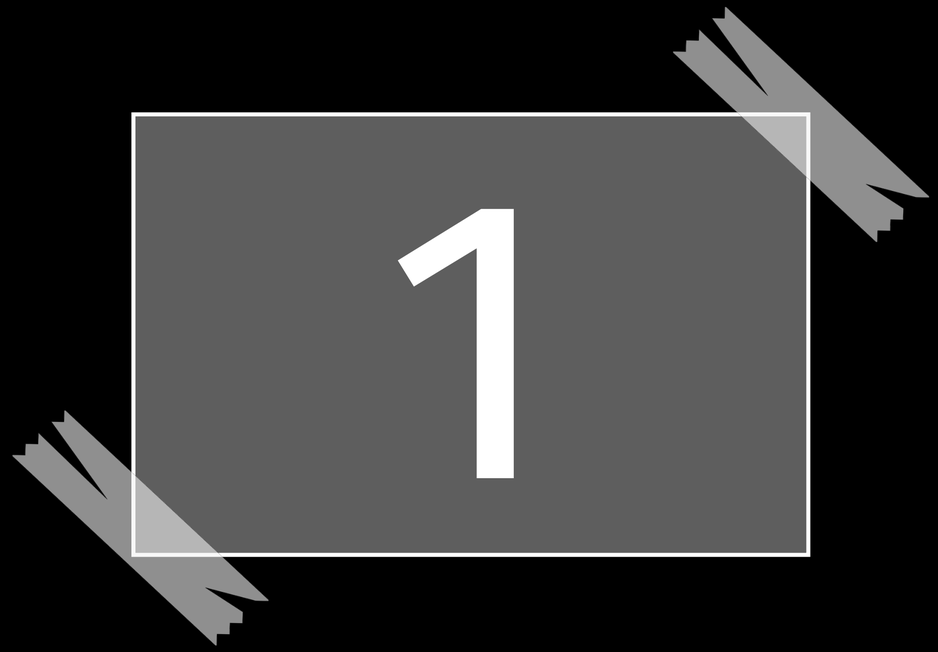
```
Utility for determining whether the specified mass is the primary isotope for the given atomic number.
private boolean isMajorIsotope(int number, int mass){
    …
}
```

TRANSL. PIVOTING

```
The mass is the major isotope for the number.
private boolean isMajorIsotope(int number, int mass){
    …
}
```

DL-BASED PARAPH.

Developers using different wordings to describe a needed method may end up with different recommendations, potentially resulting in the loss of ~28% correct recommendations



Handler used when a resource is represented by an Object that is a JSON element.

Starting from the array 'vs' inject the vertex 'v' in canonical position.

Converts a JSON workflow configuration to a workflow tool.

Find, within the trie symbol table, the longest prefix from position (i).

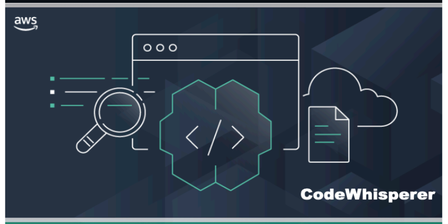
...

...

892

Up to what extent we can rely on automatic SOTA tools for generating different, but semantically equivalent code descriptions?

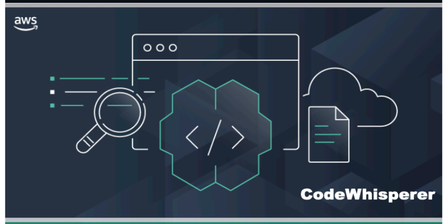
SOTA paraphrasing techniques can be used to test the robustness of DL-based code recommenders by generating semantically equivalent descriptions in up to *77% of cases*.



Code Summarization Task

Summarize this method:

```
private static String toGroupName(String
                                    attribute){
    int pos = attribute.indexOf(';');
    if (pos > -1) {
        attribute = attribute
                        .substring(0, pos);
    }
    return attribute.replace("X", "XX")
                    .replace(“.", "X");
}
```



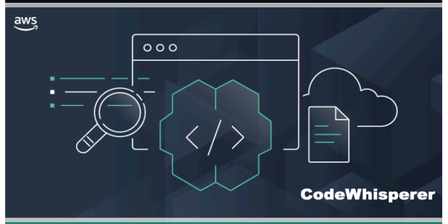
Code Summarization Task

Summarize this method:

```
private static String toGroupName(String
                                    attribute){
    int pos = attribute.indexOf(';');
    if (pos > -1) {
        attribute = attribute
                        .substring(0, pos);
    }
    return attribute.replace("X", "XX")
                    .replace(“.", "X");
}
```

GitHub Copilot

```
// write the binary search algorithm

public int runBinarySearchIteratively(int[] sortedArray, int key,
                                      int low, int high) {
    int index = Integer.MAX_VALUE;
    while (low <= high) {
        int mid = low  + ((high - low) / 2);
        if (sortedArray[mid] < key) {
            low = mid + 1;
        } else if (sortedArray[mid] > key) {
            high = mid - 1;
        } else if (sortedArray[mid] == key) {
            index = mid;
            break;
        }
    }
    return index;
}
```



Code Summarization Task

Summarize this method:

```
private static String toGroupName(String
                                    attribute){
    int pos = attribute.indexOf(';');
    if (pos > -1) {
        attribute = attribute
                        .substring(0, pos);
    }
    return attribute.replace("X", "XX")
                    .replace(".", "X");
}
```

GitHub Copilot

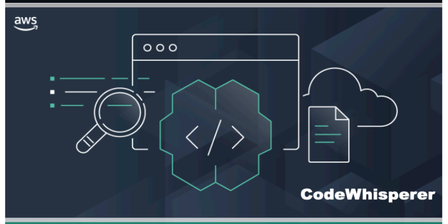
```
// write the binary search algorithm

public int runBinarySearchIteratively(int[] sortedArray, int key,
                                      int low, int high) {
    int index = Integer.MAX_VALUE;
    while (low <= high) {
        int mid = low  + ((high - low) / 2);
        if (sortedArray[mid] < key) {
            low = mid + 1;
        } else if (sortedArray[mid] > key) {
            high = mid - 1;
        } else if (sortedArray[mid] == key) {
            index = mid;
            break;
        }
    }
    return index;
}
```

```
Utility to determine if the specified mass is the major isotope for the given atomic number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
ORIGINAL

```
Given an atomic number, the utility can determine if a specific mass is the major isotope
for that atomic number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
MANUALLY PARAPH.

```
Utility for determining whether the specified mass is the primary isotope for the given
atomic number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
TRANSL. PIVOTING

```
The mass is the major isotope for the number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
DL-BASED PARAPH.



Code Summarization Task

Summarize this method:

```
private static String toGroupName(String
                                    attribute){
    int pos = attribute.indexOf(';');
    if (pos > -1) {
        attribute = attribute
                        .substring(0, pos);
    }
    return attribute.replace("X", "XX")
                    .replace(".", "X");
}
```

GitHub Copilot

```
// write the binary search algorithm

public int runBinarySearchIteratively(int[] sortedArray, int key,
                                      int low, int high) {
    int index = Integer.MAX_VALUE;
    while (low <= high) {
        int mid = low  + ((high - low) / 2);
        if (sortedArray[mid] < key) {
            low = mid + 1;
        } else if (sortedArray[mid] > key) {
            high = mid - 1;
        } else if (sortedArray[mid] == key) {
            index = mid;
            break;
        }
    }
    return index;
}
```

```
Utility to determine if the specified mass is the major isotope for the given atomic number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
ORIGINAL

```
Given an atomic number, the utility can determine if a specific mass is the major isotope
for that atomic number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
MANUALLY PARAPH.

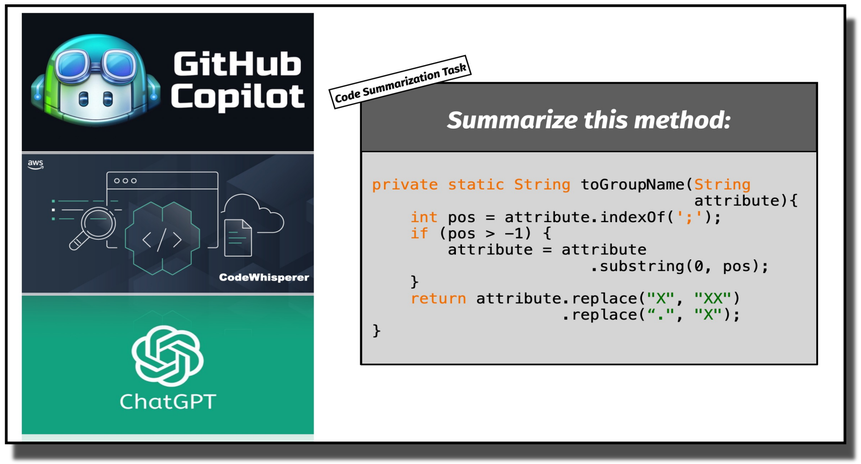
```
Utility for determining whether the specified mass is the primary isotope for the given
atomic number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
TRANSL. PIVOTING

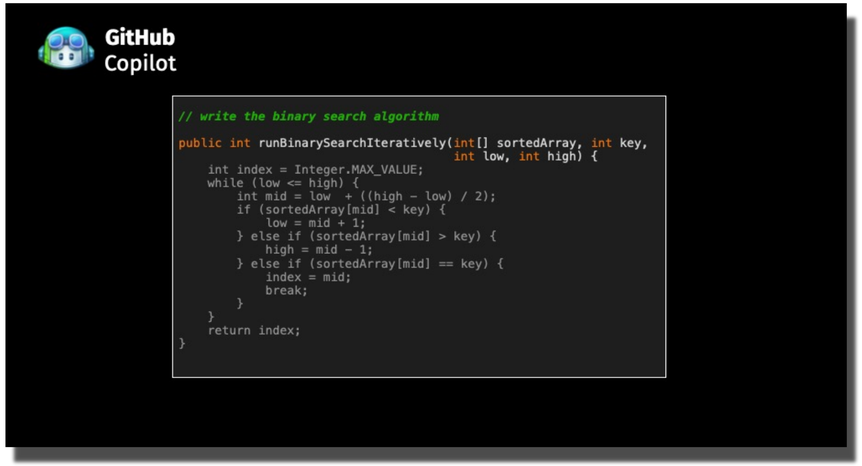
```
The mass is the major isotope for the number.
private boolean isMajorIsotope(int number, int mass){
    ""
}
```
DL-BASED PARAPH.

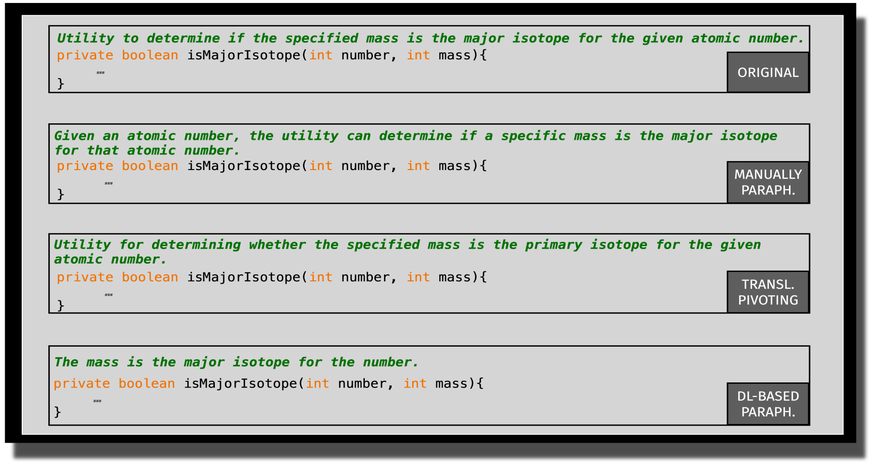
Developers using different wordings to describe a needed method may end up with different recommendations, potentially resulting in the loss of ~28% correct recommendations

FINDINGS

SOTA paraphrasing techniques can be used to test the robustness of DL-based code recommenders by generating semantically equivalent descriptions in up to *77% of cases*.







Developers using different wordings to describe a needed method may end up with different recommendations, potentially resulting in the loss of ~28% correct recommendations

FINDINGS

SOTA paraphrasing techniques can be used to test the robustness of DL-based code recommenders by generating semantically equivalent descriptions in up to *77% of cases*.

www.github.com/antonio-mastropaolo/robustness-copilot

SEART

Università della Svizzera italiana | Software Institute

International Conference on Software Engineering 2023

Stay in touch!

antonio.mastropaolo@usi.ch
www.antoniomastropaolo.com
AntonioMastro2