Report For CodeBLEU

Background

CodeBLEU is mainly composed of three parts: text n-gram, syntax-aware n-gram and semantic similarity matching. Here in the code realized the three parts separately, and the following result all based on these three indices.

Method

Use the code in CodeBLEU.py, and vary the value of n-gram.

Result Compare

The following table contains 5 LeetCode problems and table 1 is the result of *faultyJavaProgramsToCompare* and *correctJavaPrograms*. Table2 is the result of AI-*GeneratedCode* and *correctJavaPrograms*. The result is the score of 3 indices.

	LC-1	LC-2	LC-3	LC-4	LC-5
n-gram(n=4)	0.9790	0.9639	0.9857	0.9775	1.0
AST n-gram(n=3)	1.0	1.0	1.0	1.0	1.0
semantic similarity	1.0	1.0	1.0	1.0	1.0

Table.1 Result of faultyCompare/Correct

	LC-1	LC-2	LC-3	LC-4	LC-5
n-gram(n=4)	0.4769	0.2778	0.3669	06675	0.6698
AST n-gram(n=3)	0.7230	0.2605	0.5447	0.8263	0.8373
semantic similarity	0.7996	0.6370	0.7045	0.8559	0.8800

Table.2 Result of Generate/Correct

	LC-1	LC-2	LC-3	LC-4	LC-5
n-gram(n=2)	0.9832	0.9744	0.9922	0.9840	0.9547
n-gram(n=3)	0.9810	0.9692	0.9887	0.9808	0.9481
n-gram(n=4)	0.9790	0.9639	0.9857	0.9775	0.9415
n-gram(n=5)	0.9771	0.9585	0.9828	0.9743	0.9348
n-gram(n=6)	0.9752	0.9532	0.9800	0.9715	0.9297

Table.3 Result of Compare/Correct in different n-gram

LC-1: basic-calculator; LC-2: count-the-number-of-special-characters-ii; LC-3: minimum-cost-good-caption; LC-4: robot-collisions; LC-5: sum-of-largest-prime-substrings

Conclusion

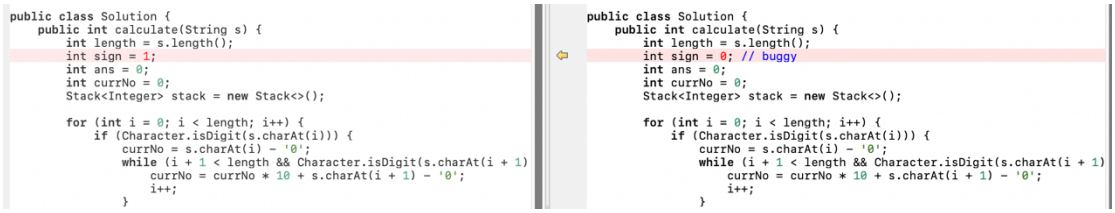
As shown above, the similarity indices for n-gram metrics are low between the generated code and the correct code, since n-gram compares the code character-by-character, the similarity score is low and cannot distinguish whether different codes

implement the same function. But for AST n-gram and semantic similarity, the values are higher which shows the AST n-gram and semantic similarity can, to some extent, determine whether the semantics of the code are consistent.

As for Compare/Correct code, n-gram's value is not 1 which means n-gram can show the difference between these two codes, and the larger the n, the coarser the granularity(Table3), the lower the fault tolerance, and the more sensitive it is to local changes.

As for AST n-gram and semantic similarity it all shows the two codes have same meaning. AST only extract Declaration/ Statement/ Expression/ Type/ Annotation/ Auxiliary / Container Nodes, thus in the codes, most other difference won't be detected by the index. And below is a detailed analysis of each question:

- LC-1: The AST is unaware of concrete variable values



```
public class Solution {
    public int calculate(String s) {
        int length = s.length();
        int sign = 1;
        int ans = 0;
        int currNo = 0;
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < length; i++) {
            if (Character.isDigit(s.charAt(i))) {
                currNo = s.charAt(i) - '0';
                while (i + 1 < length && Character.isDigit(s.charAt(i + 1))
                    currNo = currNo * 10 + s.charAt(i + 1) - '0';
                    i++;
                }
```

```
public class Solution {
    public int calculate(String s) {
        int length = s.length();
        int sign = 0; // buggy
        int ans = 0;
        int currNo = 0;
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < length; i++) {
            if (Character.isDigit(s.charAt(i))) {
                currNo = s.charAt(i) - '0';
                while (i + 1 < length && Character.isDigit(s.charAt(i + 1))
                    currNo = currNo * 10 + s.charAt(i + 1) - '0';
                    i++;
                }
```

Pic.1 LC-1

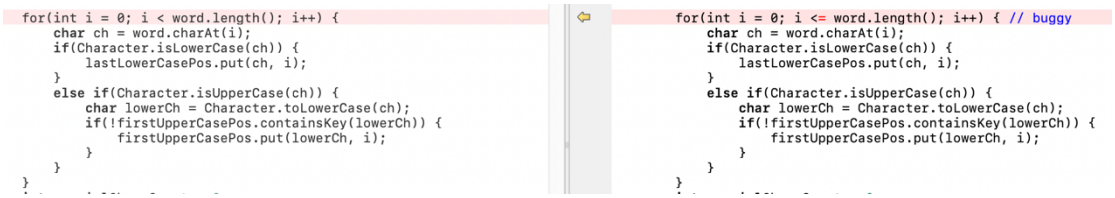
- LC-2: The AST does not capture the specific symbolic logic.

```
public class Solution {
    public String minCostGoodCaption(String caption) {
        if (caption.length() < 3) {
            return "";
        }
        char[] arr = caption.toCharArray();
        State[] dp = new State[arr.length];
        State state = go(dp, arr, 0);
        char[] res = new char[arr.length];
```

```
public class Solution {
    public String minCostGoodCaption(String caption) {
        if (!(caption.length() < 3)) { // buggy
            return "";
        }
        char[] arr = caption.toCharArray();
        State[] dp = new State[arr.length];
        State state = go(dp, arr, 0);
        char[] res = new char[arr.length];
```

Pic.2 LC-2

- LC-3: The AST does not capture the specific symbolic logic.

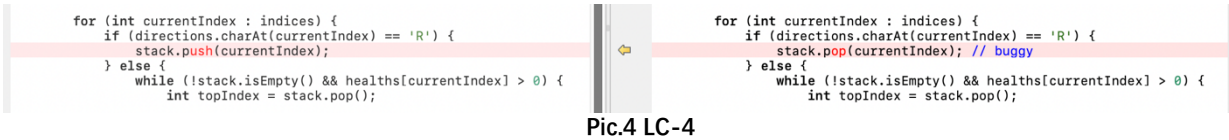


```
for(int i = 0; i < word.length(); i++) {
    char ch = word.charAt(i);
    if(Character.isLowerCase(ch)) {
        lastLowerCasePos.put(ch, i);
    }
    else if(Character.isUpperCase(ch)) {
        char lowerCh = Character.toLowerCase(ch);
        if(!firstUpperCasePos.containsKey(lowerCh)) {
            firstUpperCasePos.put(lowerCh, i);
        }
    }
}
```

```
for(int i = 0; i <= word.length(); i++) { // buggy
    char ch = word.charAt(i);
    if(Character.isLowerCase(ch)) {
        lastLowerCasePos.put(ch, i);
    }
    else if(Character.isUpperCase(ch)) {
        char lowerCh = Character.toLowerCase(ch);
        if(!firstUpperCasePos.containsKey(lowerCh)) {
            firstUpperCasePos.put(lowerCh, i);
        }
    }
}
```

Pic.3 LC-3

- LC-4: The AST is only responsible for syntactic structure; it does not perceive runtime semantics.

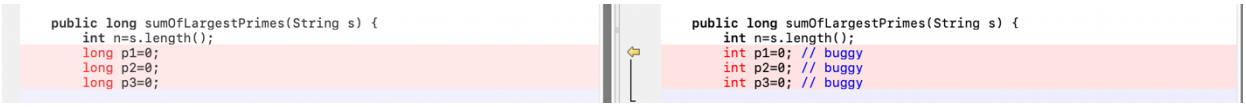


```
for (int currentIndex : indices) {
    if (directions.charAt(currentIndex) == 'R') {
        stack.push(currentIndex);
    } else {
        while (!stack.isEmpty() && healths[currentIndex] > 0) {
            int topIndex = stack.pop();
```

```
for (int currentIndex : indices) {
    if (directions.charAt(currentIndex) == 'R') {
        stack.pop(currentIndex); // buggy
    } else {
        while (!stack.isEmpty() && healths[currentIndex] > 0) {
            int topIndex = stack.pop();
```

Pic.4 LC-4

- LC-5: As expected, the variable similarity is not 1 — the two sets of variables are not identical. However, analysis shows that when Jaccard similarity is used, the variable list is turned into a set first, which merges duplicate variable names; it does not compare the variables one-by-one. Meanwhile, by examining the AST nodes we found that each node only records the **type** of the symbol; the actual value (or subtype) of that type is **not** compared. Consequently, the node-level similarity also comes out as 1.



Pic.5 LC-5

In summary, the experiments and analyses above show that when two pieces of code differ greatly in overall structure, the AST can still extract their structural trees, and the resulting similarity score reflects—at least to some extent—whether the two programs perform the same function. However, when the differences are small and lie in minor logical details, n-gram comparison is required to detect them. Therefore, the experiments above reveal that CodeBLEU can give only a coarse indication of code similarity and fails to capture fine-grained logical differences between programs.