

## B Sample Inputs and Outputs

To illustrate the output of each prompting strategy, this section provides a concrete example based on the ‘[plusOut](#)’ problem from the CodeWorkout website. The outputs shown were generated by the *anthropic/claude-sonnet-4* model and are representative of the results from our experiments.

### B.1 Problem Description (Input)

The following problem description was provided to the model as input for all strategies. The task is to implement a Java method named ‘plusOut’.

Write a function in Java that implements the following logic: Given a string `str` and a non-empty word, return a version of the original string where all chars have been replaced by pluses (+), except for appearances of the word which are preserved unchanged.

Examples:

`plusOut("12xy34", "xy") -> "++xy++"`

`plusOut("12xy34", "1") -> "1++++"`

### B.2 Input-Output (IO) Prompting Output

Given the prompt in Appendix A.1, the model produced the following submissions. Note the absence of explicit reasoning for the introduced errors.

Submission 1:

```
```java
public String plusOut(String str, String word) {
    String result = "";
    for (int i = 0; i < str.length(); i++) {
        if (i <= str.length() - word.length() && str.substring(i, i +
word.length()).equals(word)) {
            result += word;
            i += word.length() - 1;
        } else {
            result += "+";
        }
    }
    return result;
}
```
```

Submission 2:

```
```java
public String plusOut(String str, String word) {
    String result = str;
    for (int i = 0; i < str.length(); i++) {
```

```

833     if (str.substring(i, Math.min(i + word.length(), str.length())).equals(word))
834     {
835         i += word.length() - 1;
836     } else {
837         result = result.substring(0, i) + "+" + result.substring(i + 1);
838     }
839 }
840 return result;
841 }
842 ...

```

Submission 3:

```

846 ```java
847 public String plusOut(String str, String word) {
848     String result = "";
849     int i = 0;
850     while (i < str.length()) {
851         if (str.startsWith(word, i)) {
852             result += word;
853             i += word.length();
854         } else {
855             result += "+";
856             i++;
857         }
858     }
859     return result;
860 }
861 ...

```

Submission 4:

```

867 ```java
868 public String plusOut(String str, String word) {
869     boolean[] preserve = new boolean[str.length()];
870
871     for (int i = 0; i < str.length() - word.length(); i++) {
872         if (str.substring(i, i + word.length()).equals(word)) {
873             for (int j = i; j < i + word.length(); j++) {
874                 preserve[j] = true;
875             }
876         }
877     }
878
879     String result = "";
880     for (int i = 0; i < str.length(); i++) {
881         if (preserve[i]) {

```

```

885         result += str.charAt(i);
886     } else {
887         result += "+";
888     }
889 }
890 }
891 return result;
892 }
893 ---
894
895 Submission 5:
896 ```java
897 public String plusOut(String str, String word) {
898     String result = "";
899     for (int i = 0; i < str.length(); i++) {
900         boolean found = false;
901         if (i + word.length() <= str.length()) {
902             String sub = str.substring(i, i + word.length());
903             if (sub.equals(word)) {
904                 result += word;
905                 i += word.length() - 1;
906                 found = true;
907             }
908         }
909     }
910     if (!found) {
911         result += "+";
912     }
913 }
914 }
915 return result;
916 }
917 ---
918
919 Submission 6:
920 ```java
921 public String plusOut(String str, String word) {
922     String temp = str.replace(word, "PLACEHOLDER");
923     String result = "";
924     for (int i = 0; i < temp.length(); i++) {
925         if (temp.charAt(i) == 'P' && temp.substring(i).startsWith("PLACEHOLDER")) {
926             result += word;
927             i += "PLACEHOLDER".length() - 1;
928         } else {
929             result += "+";
930         }
931     }
932 }
933 }
934 return result;
935 }
936

```

```

937 }
938 ---
939
940 Submission 7:
941 ```java
942 public String plusOut(String str, String word) {
943     String result = "";
944     int wordIndex = 0;
945
946     for (int i = 0; i < str.length(); i++) {
947         if (str.charAt(i) == word.charAt(wordIndex)) {
948             wordIndex++;
949             if (wordIndex == word.length()) {
950                 result = result.substring(0, result.length() - word.length() + 1) +
951                 word;
952                 wordIndex = 0;
953             } else {
954                 result += "+";
955             }
956         } else {
957             wordIndex = 0;
958             result += "+";
959         }
960     }
961     return result;
962 }
963 ---
964
965 Submission 8:
966 ```java
967 public String plusOut(String str, String word) {
968     String result = "";
969     for (int i = 0; i < str.length(); i++) {
970         if (i + word.length() < str.length() && str.substring(i, i +
971         word.length()).equals(word)) {
972             result += word;
973             i += word.length() - 1;
974         } else {
975             result += "+";
976         }
977     }
978     return result;
979 }
980 ---
981
982
983
984
985
986
987
988

```

### B.3 Chain-of-Thought (CoT) Prompting Output

Given the prompt in Appendix A.2, the model first provided reasoning for each intended error before generating the code.

```

Submission 1:
/*Reasoning: This approach tries to iterate through the string and check if the
current position starts with the target word. However, it fails to properly
handle overlapping matches and doesn't correctly preserve the word when found. */
```java
public String plusOut(String str, String word) {
    StringBuilder result = new StringBuilder();
    for (int i = 0; i < str.length(); i++) {
        if (i <= str.length() - word.length() && str.substring(i, i +
word.length()).equals(word)) {
            result.append(word);
        } else {
            result.append('+');
        }
    }
    return result.toString();
}
```

Submission 2:
/*Reasoning: This solution attempts to replace all characters with pluses first, then
restore the word occurrences. The logical error is in the replacement logic where
it doesn't properly account for the length difference after replacements.*/
```java
public String plusOut(String str, String word) {
    String result = str.replaceAll(".", "+");
    int index = 0;
    while ((index = str.indexOf(word, index)) != -1) {
        result = result.substring(0, index) + word + result.substring(index +
word.length());
        index++;
    }
    return result;
}
```

Submission 3:
/*Reasoning: This approach uses a character array and tries to mark positions that
should remain unchanged. The error lies in the increment logic where it advances
by only 1 position even after finding a complete word match.*/
```java
public String plusOut(String str, String word) {
    char[] chars = str.toCharArray();

```

```

1041     for (int i = 0; i < chars.length; i++) {
1042         if (i <= str.length() - word.length() && str.substring(i, i +
1043 word.length()).equals(word)) {
1044             i += word.length() - 1;
1045         } else {
1046             chars[i] = '+';
1047         }
1048     }
1049     return new String(chars);
1050 }
1051 ...
1052
1053 Submission 4:
1054 /*Reasoning: This solution tries to build the result by checking each position for
1055 word matches. The logical flaw is that it doesn't skip the remaining characters
1056 of a matched word, causing partial overwriting.*/
1057
1058 ```java
1059 public String plusOut(String str, String word) {
1060     StringBuilder result = new StringBuilder();
1061     int i = 0;
1062     while (i < str.length()) {
1063         if (str.startsWith(word, i)) {
1064             result.append(word);
1065             i += word.length();
1066         } else {
1067             result.append('+');
1068             i++;
1069         }
1070     }
1071     return result.toString();
1072 }
1073 ...
1074
1075 Submission 5:
1076 /*Reasoning: This approach attempts to use string replacement by first marking word
1077 positions with a placeholder, then replacing everything else. The error is in
1078 assuming the placeholder won't interfere with subsequent operations.*/
1079
1080 ```java
1081 public String plusOut(String str, String word) {
1082     String temp = str.replace(word, "PLACEHOLDER");
1083     temp = temp.replaceAll(".", "+");
1084     return temp.replace("PLACEHOLDER", word);
1085 }
1086 ...
1087
1088 Submission 6:

```

```

1093  /*Reasoning: This solution tries to iterate and build the result character by
1094  character while checking for word matches. The logical error is in the boundary
1095  checking condition which uses incorrect comparison operators.*/
1096  ```java
1097  public String plusOut(String str, String word) {
1098      StringBuilder result = new StringBuilder();
1099      for (int i = 0; i < str.length(); i++) {
1100          if (i < str.length() - word.length() && str.substring(i, i +
1101 word.length()).equals(word)) {
1102              result.append(word);
1103              i += word.length() - 1;
1104          } else {
1105              result.append('+');
1106          }
1107      }
1108      return result.toString();
1109  }
1110  ```
1111
1112
1113

```

#### B.4 Self-Refine Prompting Interaction

The following demonstrates a two-iteration example of the Self-Refine process, as detailed in Appendix A.3. The model begins with an initial set of generations, receives feedback from a critic agent, refines its code, and then repeats this cycle until the submission is compliant.

*Initial Generation.*

```

1125  Submission 1:
1126  /*Reasoning: Student attempts to replace characters one by one but fails to properly
1127  skip over the word when found, leading to partial replacements within the word
1128  itself.*/
1129  ```java
1130  public String plusOut(String str, String word) {
1131      String result = "";
1132      for (int i = 0; i < str.length(); i++) {
1133          if (i <= str.length() - word.length() && str.substring(i, i +
1134 word.length()).equals(word)) {
1135              result += word;
1136              i++;
1137          } else {
1138              result += "+";
1139          }
1140      }
1141      return result;
1142  }
1143  ```
1144

```

Manuscript submitted to ACM

```

1145 }
1146 ---
1147
1148 Submission 2:
1149 /*Reasoning: Student correctly identifies word occurrences but uses the wrong
1150    increment, advancing by only 1 instead of the full word length.*/
1151 ---java
1152 public String plusOut(String str, String word) {
1153     StringBuilder result = new StringBuilder();
1154     int i = 0;
1155     while (i < str.length()) {
1156         if (i <= str.length() - word.length() && str.substring(i, i +
1157 word.length()).equals(word)) {
1158             result.append(word);
1159             i += word.length() - 1;
1160         } else {
1161             result.append("+");
1162         }
1163         i++;
1164     }
1165     return result.toString();
1166 }
1167 ---
1168
1169 Submission 3:
1170 /*Reasoning: Student uses replace method but incorrectly replaces the word with a
1171    placeholder first, then tries to restore it, causing issues with overlapping
1172    replacements.*/
1173 ---java
1174 public String plusOut(String str, String word) {
1175     String temp = str.replace(word, "TEMP");
1176     String result = "";
1177     for (int i = 0; i < temp.length(); i++) {
1178         if (temp.charAt(i) == 'T' && i + 3 < temp.length() && temp.substring(i, i +
1179 4).equals("TEMP")) {
1180             result += word;
1181             i += 3;
1182         } else {
1183             result += "+";
1184         }
1185     }
1186     return result;
1187 }
1188 ---
1189

```



```

1197 Submission 4:
1198 /*Reasoning: Student attempts to build the result by checking each position but has
1199    an off-by-one error in the boundary condition for substring extraction.*/
1200 ```java
1201 public String plusOut(String str, String word) {
1202     String result = "";
1203     for (int i = 0; i < str.length(); i++) {
1204         if (i < str.length() - word.length() && str.substring(i, i +
1205 word.length()).equals(word)) {
1206             result += word;
1207             i += word.length() - 1;
1208         } else {
1209             result += "+";
1210         }
1211     }
1212     return result;
1213 }
1214 }
1215 ```
1216
1217
1218 Submission 5:
1219 /*Reasoning: Student correctly finds word matches but forgets to handle the case
1220    where the word appears at the very end of the string due to incorrect loop
1221    bounds.*/
1222 ```java
1223 public String plusOut(String str, String word) {
1224     StringBuilder sb = new StringBuilder();
1225     int i = 0;
1226     while (i < str.length()) {
1227         if (i + word.length() < str.length() && str.substring(i, i +
1228 word.length()).equals(word)) {
1229             sb.append(word);
1230             i += word.length();
1231         } else {
1232             sb.append("+");
1233             i++;
1234         }
1235     }
1236     return sb.toString();
1237 }
1238 }
1239 ```
1240
1241
1242
1243 Submission 6:
1244 /*Reasoning: Student uses a character-by-character approach but fails to properly
1245    handle multi-character words, only checking the first character of the word.*/
1246 ```java
1247
1248

```

```

1249 public String plusOut(String str, String word) {
1250     String result = "";
1251     for (int i = 0; i < str.length(); i++) {
1252         boolean found = false;
1253         if (str.charAt(i) == word.charAt(0)) {
1254             if (i + word.length() <= str.length()) {
1255                 String sub = str.substring(i, i + word.length());
1256                 if (sub.equals(word)) {
1257                     result += word;
1258                     i += word.length() - 1;
1259                     found = true;
1260                 }
1261             }
1262         }
1263     }
1264     if (!found) {
1265         result += "+";
1266     }
1267 }
1268
1269 return result;
1270
1271 }
1272 ...

```

*Iteration 1: Feedback.* The critic agent receives the initial code and provides the following feedback:

Submission 1: Revise      The logical error (incrementing `i` by only 1 instead of `word.length()` when a match is found) is non-trivial and plausible, but the increment should be `i += word.length() - 1` to maintain the loop's natural increment, making the error more subtle and realistic for a novice.

Submission 2: Compliant      Contains exactly one non-trivial logical error where the increment is `word.length() - 1` instead of `word.length()`, causing overlapping matches to be missed. The error is subtle and represents a common off-by-one mistake.

Submission 3: Revise      The approach using string replacement with "TEMP" is overly complex for a novice submission and contains multiple issues: hardcoded "TEMP" length check and potential boundary problems. Simplify to a more straightforward approach with a single, clear logical error.

Submission 4: Revise      The boundary condition uses `<` instead of `<=`, which will miss matches at the end of the string. While this is a logical error, it's too similar to other boundary condition errors in the set. Change to a different type of logical error to maintain distinctness.

Submission 5: Compliant      Contains exactly one non-trivial logical error in the boundary condition (``i + word.length() < str.length()`` instead of ``<=```), which prevents detection of matches at the string's end. This represents a common boundary condition mistake.

Submission 6: Compliant      Contains exactly one non-trivial logical error where it only checks if the first character matches before attempting substring comparison, but the logic is otherwise sound. This represents a reasonable optimization attempt that introduces a subtle bug.

*Iteration 1: Refinement.* Based on the feedback, the model revises the codes:

Submission 1:

```
```java
public String plusOut(String str, String word) {
    String result = "";
    for (int i = 0; i < str.length(); i++) {
        if (i <= str.length() - word.length() && str.substring(i, i +
word.length()).equals(word)) {
            result += word;
            i += word.length() - 1;
        } else {
            result += "+";
        }
    }
    return result;
}
```
```

Submission 2:

```
```java
public String plusOut(String str, String word) {
    StringBuilder result = new StringBuilder();
    int i = 0;
    while (i < str.length()) {
        if (i <= str.length() - word.length() && str.substring(i, i +
word.length()).equals(word)) {
            result.append(word);
            i += word.length() - 1;
        } else {
            result.append("+");
        }
        i++;
    }
    return result.toString();
}
```
```

```

1353 }
1354 ---
1355
1356 Submission 3:
1357 ```java
1358 public String plusOut(String str, String word) {
1359     String result = "";
1360     for (int i = 0; i < str.length(); i++) {
1361         if (i <= str.length() - word.length() && str.substring(i, i +
1362 word.length()).equals(word)) {
1363             result += word;
1364             i += word.length();
1365         } else {
1366             result += "+";
1367         }
1368     }
1369     return result;
1370 }
1371 ---
1372
1373 Submission 4:
1374 ```java
1375 public String plusOut(String str, String word) {
1376     String result = "";
1377     for (int i = 0; i < str.length(); i++) {
1378         if (i <= str.length() - word.length() && str.substring(i, i +
1379 word.length()).equalsIgnoreCase(word)) {
1380             result += word;
1381             i += word.length() - 1;
1382         } else {
1383             result += "+";
1384         }
1385     }
1386     return result;
1387 }
1388 ---
1389
1390 Submission 5:
1391 ```java
1392 public String plusOut(String str, String word) {
1393     StringBuilder sb = new StringBuilder();
1394     int i = 0;
1395     while (i < str.length()) {
1396         if (i + word.length() < str.length() && str.substring(i, i +
1397 word.length()).equals(word)) {

```

```

1405         sb.append(word);
1406         i += word.length();
1407     } else {
1408         sb.append("+");
1409         i++;
1410     }
1411 }
1412 }
1413 return sb.toString();
1414 }
1415 ...
1416
1417 Submission 6:
1418 ```java
1419 public String plusOut(String str, String word) {
1420     String result = "";
1421     for (int i = 0; i < str.length(); i++) {
1422         boolean found = false;
1423         if (str.charAt(i) == word.charAt(0)) {
1424             if (i + word.length() <= str.length()) {
1425                 String sub = str.substring(i, i + word.length());
1426                 if (sub.equals(word)) {
1427                     result += word;
1428                     i += word.length() - 1;
1429                     found = true;
1430                 }
1431             }
1432         }
1433     }
1434     if (!found) {
1435         result += "+";
1436     }
1437 }
1438 return result;
1439 }
1440 }
1441 ...
1442

```

*Iteration 2: Feedback.* The critic agent reviews the refined codes and now deems it compliant with all constraints.

```

1447 Submission 1: Compliant      Contains exactly one non-trivial logical error where the
1448     loop increment `i += word.length() - 1` combined with the for loop's `i++` causes
1449     the index to advance by only `word.length()` instead of the full word length,
1450     leading to potential overlapping matches.
1451
1452 Submission 2: Compliant      Contains exactly one non-trivial logical error identical
1453     to Submission 1, where the increment logic `i += word.length() - 1` followed by
1454     `i++` in the while loop causes incorrect index advancement.
1455

```

Manuscript submitted to ACM

Submission 3: Compliant      Contains exactly one non-trivial logical error where ``i += word.length()`` without accounting for the for loop's automatic ``i++`` causes the index to skip one character after each word match.

Submission 4: Compliant      Contains exactly one non-trivial logical error using ``equalsIgnoreCase`` instead of ``equals``, making the comparison case-insensitive when it should be case-sensitive, plus the same increment issue as Submission 1.

Submission 5: Compliant      Contains exactly one non-trivial logical error in the boundary condition ``i + word.length() < str.length()`` which uses strict inequality instead of ``<=```, causing it to miss word matches at the end of the string.

Submission 6: Compliant      Contains exactly one non-trivial logical error in the optimization attempt that only checks matches when the first character matches, but this approach can miss valid word occurrences and has the same increment issue as other submissions.