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    been replaced by pluses (+), except for appearances of the word which are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++) {
~~~

~~~
        if (str.substring(i, Math.min(i + word.length(), str.length())).equals(word))
    {
            i += word.length() - 1;
        } else {
            result = result.substring(0, i) + "+" + result.substring(i + 1);
        }
    }
    return result;
}
```

Submission 3:
```java
public String plusOut(String str, String word) {
    String result = "";
    int i = 0;
    while (i < str.length()) {
        if (str.startsWith(word, i)) {
            result += word;
            i += word.length();
        } else {
            result += "+";
            i++;
        }
    }
    return result;
}
```

Submission 4:
```java
public String plusOut(String str, String word) {
    boolean[] preserve = new boolean[str.length()];

    for (int i = 0; i < str.length() - word.length(); i++) {
        if (str.substring(i, i + word.length()).equals(word)) {
            for (int j = i; j < i + word.length(); j++) {
                preserve[j] = true;
            }
        }
    }

    String result = "";
    for (int i = 0; i < str.length(); i++) {
        if (preserve[i]) {
~~~

~~~~
            result += str.charAt(i);
        } else {
            result += "+";
        }
    }
    return result;
}
```

Submission 5:
```java
public String plusOut(String str, String word) {
    String result = "";
    for (int i = 0; i < str.length(); i++) {
        boolean found = false;
        if (i + word.length() <= str.length()) {
            String sub = str.substring(i, i + word.length());
            if (sub.equals(word)) {
                result += word;
                i += word.length() - 1;
                found = true;
            }
        }
        if (!found) {
            result += "+";
        }
    }
    return result;
}
```

Submission 6:
```java
public String plusOut(String str, String word) {
    String temp = str.replace(word, "PLACEHOLDER");
    String result = "";
    for (int i = 0; i < temp.length(); i++) {
        if (temp.charAt(i) == 'P' && temp.substring(i).startsWith("PLACEHOLDER")) {
            result += word;
            i += "PLACEHOLDER".length() - 1;
        } else {
            result += "+";
        }
    }
    return result;
~~~~

```
}
```

Submission 7:
```java
public String plusOut(String str, String word) {
    String result = "";
    int wordIndex = 0;

    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == word.charAt(wordIndex)) {
            wordIndex++;
            if (wordIndex == word.length()) {
                result = result.substring(0, result.length() - word.length() + 1) +
    word;
                wordIndex = 0;
            } else {
                result += "+";
            }
        } else {
            wordIndex = 0;
            result += "+";
        }
    }
    return result;
}
```

Submission 8:
```java
public String plusOut(String str, String word) {
    String result = "";
    for (int i = 0; i < str.length(); i++) {
        if (i + word.length() < str.length() && str.substring(i, i +
    word.length()).equals(word)) {
            result += word;
            i += word.length() - 1;
        } else {
            result += "+";
        }
    }
    return result;
}
```
````
````

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();
~~~

~~~
    for (int i = 0; i < chars.length; i++) {
        if (i <= str.length() - word.length() && str.substring(i, i +
    word.length()).equals(word)) {
            i += word.length() - 1;
        } else {
            chars[i] = '+';
        }
    }
    return new String(chars);
}
```
Submission 4:
/*Reasoning: This solution tries to build the result by checking each position for
    word matches. The logical flaw is that it doesn't skip the remaining characters
    of a matched word, causing partial overwriting.*/
```java
public String plusOut(String str, String word) {
    StringBuilder result = new StringBuilder();
    int i = 0;
    while (i < str.length()) {
        if (str.startsWith(word, i)) {
            result.append(word);
            i += word.length();
        } else {
            result.append('+');
            i++;
        }
    }
    return result.toString();
}
```
Submission 5:
/*Reasoning: This approach attempts to use string replacement by first marking word
    positions with a placeholder, then replacing everything else. The error is in
    assuming the placeholder won't interfere with subsequent operations.*/
```java
public String plusOut(String str, String word) {
    String temp = str.replace(word, "PLACEHOLDER");
    temp = temp.replaceAll(".", "+");
    return temp.replace("PLACEHOLDER", word);
}
```
Submission 6:
~~~

~~~
/*Reasoning: This solution tries to iterate and build the result character by
    character while checking for word matches. The logical error is in the boundary
    checking condition which uses incorrect comparison operators.*/
```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);
            i += word.length() - 1;
        } else {
            result.append('+');
        }
    }
    return result.toString();
}
```
~~~

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.

~~~
Submission 1:
/*Reasoning: Student attempts to replace characters one by one but fails to properly
    skip over the word when found, leading to partial replacements within the word
    itself.*/
```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++;
        } else {
            result += "+";
        }
    }
    return result;
~~~

~~~
}
```

Submission 2:
/*Reasoning: Student correctly identifies word occurrences but uses the wrong
    increment, advancing by only 1 instead of the full word length.*/
```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();
}
```

Submission 3:
/*Reasoning: Student uses replace method but incorrectly replaces the word with a
    placeholder first, then tries to restore it, causing issues with overlapping
    replacements.*/
```java
public String plusOut(String str, String word) {
    String temp = str.replace(word, "TEMP");
    String result = "";
    for (int i = 0; i < temp.length(); i++) {
        if (temp.charAt(i) == 'T' && i + 3 < temp.length() && temp.substring(i, i +
    4).equals("TEMP")) {
            result += word;
            i += 3;
        } else {
            result += "+";
        }
    }
    return result;
}
```
~~~

~~~
Submission 4:
/*Reasoning: Student attempts to build the result by checking each position but has
    an off-by-one error in the boundary condition for substring extraction.*/
```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 5:
/*Reasoning: Student correctly finds word matches but forgets to handle the case
    where the word appears at the very end of the string due to incorrect loop
    bounds.*/
```java
public String plusOut(String str, String word) {
    StringBuilder sb = new StringBuilder();
    int i = 0;
    while (i < str.length()) {
        if (i + word.length() < str.length() && str.substring(i, i +
    word.length()).equals(word)) {
            sb.append(word);
            i += word.length();
        } else {
            sb.append("+");
            i++;
        }
    }
    return sb.toString();
}
```


Submission 6:
/*Reasoning: Student uses a character-by-character approach but fails to properly
    handle multi-character words, only checking the first character of the word.*/
```java
~~~

~~~
public String plusOut(String str, String word) {
    String result = "";
    for (int i = 0; i < str.length(); i++) {
        boolean found = false;
        if (str.charAt(i) == word.charAt(0)) {
            if (i + word.length() <= str.length()) {
                String sub = str.substring(i, i + word.length());
                if (sub.equals(word)) {
                    result += word;
                    i += word.length() - 1;
                    found = true;
                }
            }
        }
        if (!found) {
            result += "+";
        }
    }
    return result;
}
```
~~~

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();
~~~

~~~
}
```

Submission 3:
```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();
        } else {
            result += "+";
        }
    }
    return result;
}
```

Submission 4:
```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()).equalsIgnoreCase(word)) {
            result += word;
            i += word.length() - 1;
        } else {
            result += "+";
        }
    }
    return result;
}
```

Submission 5:
```java
public String plusOut(String str, String word) {
    StringBuilder sb = new StringBuilder();
    int i = 0;
    while (i < str.length()) {
        if (i + word.length() < str.length() && str.substring(i, i +
    word.length()).equals(word)) {
~~~

~~~~
            sb.append(word);
            i += word.length();
        } else {
            sb.append("+");
            i++;
        }
    }
    return sb.toString();
}
```

Submission 6:
```java
public String plusOut(String str, String word) {
    String result = "";
    for (int i = 0; i < str.length(); i++) {
        boolean found = false;
        if (str.charAt(i) == word.charAt(0)) {
            if (i + word.length() <= str.length()) {
                String sub = str.substring(i, i + word.length());
                if (sub.equals(word)) {
                    result += word;
                    i += word.length() - 1;
                    found = true;
                }
            }
        }
        if (!found) {
            result += "+";
        }
    }
    return result;
}
```
~~~~

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

~~~~
Submission 1: Compliant     Contains exactly one non-trivial logical error where the
    loop increment `i += word.length() - 1` combined with the for loop's `i++` causes
    the index to advance by only `word.length()` instead of the full word length,
    leading to potential overlapping matches.

Submission 2: Compliant     Contains exactly one non-trivial logical error identical
    to Submission 1, where the increment logic `i += word.length() - 1` followed by
    `i++` in the while loop causes incorrect index advancement.
~~~~

```
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.
```