1 Current Version

Detection Model Invocation. Alg. 1 presents the procedure of invoking well-trained detection model to generate an extracted method as detected refactoring opportunities (I omitted the process of representation generation). The input of this algorithm is the code property graph of target method: $G(m)$. Line 2 generates extracted method. Line 3 finally returns the extracted method as detection results. It is easy to understand.

Algorithm 1 Refactoring Opportunity Detection Algorithm

Input: $G(m)$ - Code Property Graph of Target Method: m
Output: EM - Extracted Method
1: $EG \longleftarrow G(m)$ - initialization
2: Input EG to detection model, obtaining a set of code statements (they may non-consecutive) as the extract method: EM
3: **return** EM;

For example, according to your provided case, we will get only one detection result in Fig. 1:

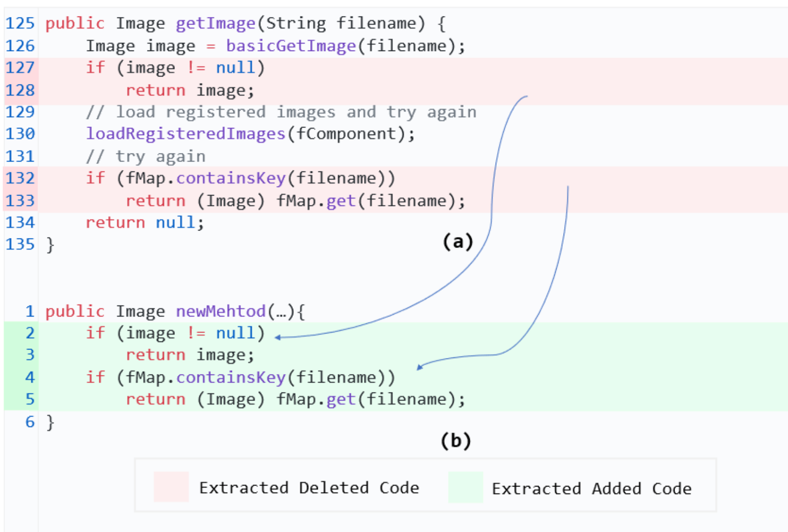

Figure 1: The result of Alg. 1 to generate an extracted method as detected refactoring opportunities.

We use precision, recall and F1 to evaluate our experimental results. These indexes can be obtained through four categories of prediction results. Here, TP represents the number of code lines where the tool correctly identifies a refactoring that actually exists in the method, TN represents the number of code lines where the tool incorrectly identifies a refactoring that doesn't actually exist in the method, FP represents the number of false positive cases where no actual refactoring is present but the tool detects one, and FN represents the number of false negative cases where an actual refactoring is present but the tool fails to detect it. So we will obtain the following formula results:

$$\text{Precision} = \frac{\#\text{ of correct recommended refactorings}}{\#\text{ of recommended refactorings}} = \frac{\text{TP}}{\text{TP+FP}} = \frac{2}{4} = 50\% \tag{1}$$

$$\text{Recall} = \frac{\# \, of \text{ correct recommended refactorings}}{\#\text{ of correct refactorings}} = \frac{\text{TP}}{\text{TP+FN}} = \frac{2}{2+1} = 66.7\% \tag{2}$$

2 Feature Version

While we are still in the experimental testing phase for this enhancement, we are committed to promptly updating our package once we have achieved significant results.

Detection Model Invocation. Alg. 2 presents the procedure of invoking well-trained detection model to generate a group of extracted methods as detected refactoring opportunities (I omitted the process of representation generation). The input of this algorithm is the code property graph of target method: $G(m)$ and number of extracted methods: k. This algorithm recursively invokes the detection model to generate the extracted method until the number of extracted methods: k is satisfied. Line 2 to 10 iteratively generate each extracted methods. Line 3 inputs the code property graph: EG into the detection model and retrieves the extracted method: em. Line 4 to 7 append the extracted method: EM_{new} to the extracted method list: $EMList$, and line 8 updates the code property graph for EG. Line 9 sets the current extracted method em as the previous extracted method for the next iteration. Lines 11 to 13 append the final round extracted method: EM_{pre} to the extracted method list: $EMList$. Line 14 finally returns the extracted method list as detection results.

Algorithm 2 Refactoring Opportunity Detection Algorithm

Input: $G(m)$ - Code Property Graph of Target Method: m
Input: k - Number of Extracted Methods
Output: $EMList$ - List of Extracted Methods

1: EG, $EMList$ $EM_{pre} \longleftarrow G(m)$, $\emptyset$, $\emptyset$ - initialization
2: **for** $i = 1$ **to** k **do**
3: Input EG to detection model, obtaining a set of code statements as the extract method: em
4: **if** $EM_{pre} \neq \emptyset$ **then**
5: $EM_{new} \longleftarrow EM_{pre} \setminus em$ - Get the set difference operation between EM_{pre} and em
6: Append extracted method: EM_{new} into extracted method list: $EMList$
7: **end if**
8: $EG \longleftarrow G(em)$ - Generate code property graph: $G(em)$ for the extracted method: em and update it to EG
9: $EM_{pre} \longleftarrow em$ - Set the current extracted method: em as the previous extracted method for the next iteration
10: **end for**
11: **if** $EM_{pre} \neq \emptyset$ **then**
12: Append the final round extracted method: EM_{pre} into extracted method list: $EMList$
13: **end if**
14: **return** $EMList$;

For example, according to your provided case, we will get two detection results in Fig. 2:

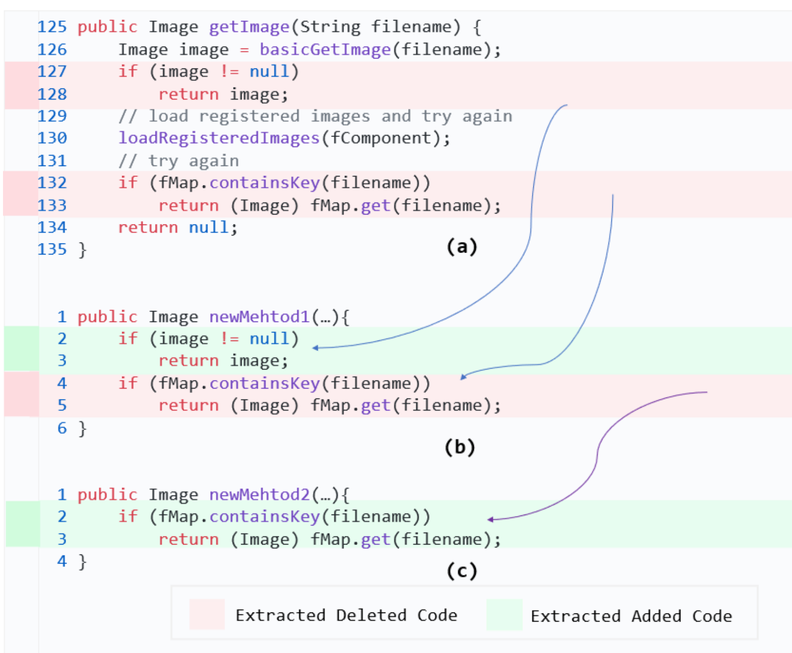

Figure 2: The result of Alg. 2 to generate an extracted method as detected refactoring opportunities.

There are two recommendations: `newMethod1()`, `newMethod2()`. If you have expert-annotated real refactoring results (as provided by Oracle in this case), we would then use the F1-measure to rank the recommendations. Recommendations with higher F1-measures would be ranked higher, indicating that they align well with the expert-annotated refactoring results.

So we will obtain the following formula results of `newMethod1()`:

$$\text{Precision} = \frac{0}{2} = 0\% \tag{3}$$

$$\text{Recall} = \frac{0}{0+3} = 0\% \tag{4}$$

$$\text{F1-Measure} = 0\% \tag{5}$$

So we will obtain the following formula results of `newMethod2()`:

$$\text{Precision} = \frac{2}{2} = 100\% \tag{6}$$

$$\text{Recall} = \frac{2}{2+1} = 66.7\% \tag{7}$$

$$\text{F1-Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2 \times 1 \times \frac{2}{3}}{1 + \frac{2}{3}} = 80\% \tag{8}$$

Following the calculation of the F1-measure, we can determine the ranking of the recommendations. As the F1-measure of `newMethod2()` is 80%, which is higher than 0%, it indicates that the recommendation for `newMethod2()` is ranked higher.