Task 2: Necessary and Sufficient Explanations

Prepare data

We only select data where the ground truth confirms that the explanation is correct. We don't use our classifiers predictions to select the data to avoid wrong explanations affecting our results.

In [1]:
```
import pandas as pd

data = pd.read_csv("answerList_data.csv")
data = data[data["GroundTruth"] == 1][["FailingMethod", "Answer.explanati
data.head()
```

Out[1]:

	FailingMethod	Answer.explanation
20	HIT01_8	Minutes are set to -15; which is less then 0 a...
21	HIT01_8	The code never gets that far. The problem is a...
22	HIT01_8	In the code there is a check that 0 <= minutes...
23	HIT01_8	There is a logical check for if minuteOffset i...
24	HIT01_8	YES. The issue is on line 279 (as I explained ...

In [4]:
```
grouped_explanations = (
    data.groupby("FailingMethod")["Answer.explanation"].apply(list).to_di
)
grouped_explanations.keys()
```

Out[4]:
```
dict_keys(['HIT01_8', 'HIT02_24', 'HIT03_6', 'HIT04_7', 'HIT05_35', 'HIT
06_51', 'HIT07_33', 'HIT08_54'])
```

Generate Ground Truth

To obtain ground truth explanations that contain all information we use our insights from the last mini project and generate an summary of all explanations using an LLM. We use the Meta LLama 3.1 405B model and prompt it to include all information in the summary and produce an easily understandable text. Due to the limited input length of the model we limit the number of explanations per bug report to 40.

We generated the ground truths once and saved them in a json file so that we can work with the same data again and have reproducable results. The following code that is commented out is the code we used to generate the ground truths.

In [6]:
```
import os
from typing import Dict, List
from huggingface_hub import login
```

```python
from openai import OpenAI
from transformers import AutoTokenizer, PreTrainedTokenizerFast
import dotenv

dotenv.load_dotenv()


MODEL_NAME = "neuralmagic/Meta-Llama-3.1-405B-Instruct-quantized.w4a16"
ANSWER_TOKEN_LENGTH = 2048
MODEL_TEMPERATURE = 0.2


class Assistant:
    model_name: str
    temperature: float
    _client: OpenAI
    _system_message: Dict[str, str]
    _messages: List[Dict[str, str]]
    _tokenizer: PreTrainedTokenizerFast

    def __init__(self, model_name: str, model_temperature: float):
        login(token="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")
        self._client = OpenAI(base_url="http://localhost:8000/v1")
        self._system_message = {
            "role": "system",
            "content": (
                "You are an helpful AI assistant. You will be given texts
            ),
        }
        self._messages = []
        self.temperature = model_temperature
        self.model_name = model_name
        self._tokenizer = AutoTokenizer.from_pretrained(model_name)
        assert (
            self._tokenizer != False
        ), f"Something went wrong when fetching the default tokenizer for

    def _all_messages(self):
        return [self._system_message] + self._messages

    def _tokenized_messages(self):
        return self._tokenizer.apply_chat_template(self._all_messages(),

    def generate_answer(self, prompt: str) -> str:
        self._messages += [
            {"role": "user", "content": prompt},
        ]
        num_tokens = len(self._tokenized_messages())
        while num_tokens > ANSWER_TOKEN_LENGTH:
            self._messages.pop(0)
            num_tokens = len(self._tokenized_messages())

        completion = self._client.completions.create(
            model=self.model_name,
            max_tokens=ANSWER_TOKEN_LENGTH,
            prompt=self._tokenized_messages(),
            temperature=self.temperature,
        )
        answer = completion.choices.pop().text
        self._messages += [
            {
                "role": "assistant",
                "content": answer,
            }
        ]

        return answer
```

In [16]:

```
# assistant = Assistant(MODEL_NAME, MODEL_TEMPERATURE)
# bug_reports = grouped_explanations.keys()
# ground_truths = {}

# for report in bug_reports:
#     prompt = f"The following explanations describe a bug. Please summar
#     ground_truths[report] = assistant.generate_answer(prompt)
#     print(f"Bug report for {report}: {ground_truths[report]}")
```

In [29]:

```
# import json
# with open("ground_truths.json", "w") as f:
#     json.dump(ground_truths, f)
```

Here we load the previously generated ground truth explanations from the json file.

In [12]:

```
import json
with open("ground_truths.json", "r") as f:
    ground_truths = json.load(f)
for report in ground_truths:
    print(f"Bug report for {report}: {ground_truths[report]}")
```

```
Bug report for HIT01_8: A bug is reported where an `IllegalArgumentExcepti
on` is thrown when the `minutesOffset` variable is set to -15, which is a
valid value according to the comments in the code. The issue is caused by
a conditional statement on line 279 that checks if `minutesOffset` is less
than 0, and if so, throws an exception. However, the comments indicate tha
t `minutesOffset` can be negative, up to -59, if the hour is positive. The
code should be updated to check if `minutesOffset` is less than -59 or gre
ater than 59, rather than just less than 0. This will allow the method to
properly progress and invoke further methods. The bug is not related to th
e variable declaration or the method call, but rather the incorrect condit
ional statement.
Bug report for HIT02_24: A bug is reported where an `IllegalArgumentExcept
ion` is thrown due to a color parameter being outside the expected range.
The issue is caused by the variable "g" being passed a negative value, whi
ch is not acceptable to the `Color` constructor. The variable "g" is calcu
lated using the formula `(int) (lowerBound + (value - lowerBound) / (upper
Bound - lowerBound) * 255)`, which can result in a negative value if the i
nput "value" is negative. The code does not properly sanitize the input "v
alue" to ensure it is within the valid range. The variable "v" is defined
but not used, and it is suggested that "v" should be used instead of "valu
e" in the calculation of "g". The valid range for the color parameter is b
etween 0.0 and 1.0, and the value of "g" should be an integer between 0 an
d 255.
Bug report for HIT03_6: The bug is a StringIndexOutOfBoundsException erro
r. The variable "pos" is being incremented beyond the length of the input
string, causing an out-of-range error when trying to access a character in
the string. The issue is likely in the loop where "pos" is being increment
ed, possibly due to "pos" being incremented at a rate faster than characte
rs from the input are consumed. The error occurs when "pos" exceeds the le
ngth of the input string, causing an exception to be thrown. The code is t
rying to access a character in the string that does not exist, resulting i
n a StringIndexOutOfBoundsException error.
Bug report for HIT04_7: The bug is related to the `maxMiddleIndex` variabl
e, which is not being updated correctly. The issue is likely in the `updat
eBounds` method, where the `maxMiddleIndex` is being set to the `index` pa
rameter, but this is not the correct value. The `maxMiddleIndex` should be
calculated based on the `s` and `e` values, which are obtained from the `g
etDataItem` method. However, the `getDataItem` method is being called with
the wrong parameter, `this.minMiddleIndex` instead of `this.maxMiddleIndex
`. Additionally, there are some type mismatch issues in the code, and some
variables are not being used correctly. The bug is causing an `AssertionFa
iledError` to be thrown, indicating that the `maxMiddleIndex` value is not
being calculated correctly.
Bug report for HIT05_35: The bug is a ClassCastException that occurs when
trying to cast an Object array to a String array. This happens because the
`type` variable is set to `Object.class` when both the `array` and `elemen
t` parameters are null. The `copyArrayGrow1` method returns an Object arra
y, which cannot be cast to a String array. The issue is caused by the line
`type = array != null ? array.getClass() : element != null ? element.getCl
ass() : Object.class;`, which sets the `type` variable to `Object.class` w
hen both `array` and `element` are null. This value is then passed to the
`copyArrayGrow1` method, which returns an Object array, leading to the Cla
ssCastException.
Bug report for HIT06_51: The bug is a ComparisonFailure that occurs when c
omparing two strings. The issue is not with the conditional statements or
the logic in the provided code, but rather with the parsing of a negative
zero fractional portion earlier in the code. The `parsePrint` method is us
ed on the value to be compared, and the parsed value is not necessarily th
e same as the original value, leading to the ComparisonFailure. The error
is not caused by the provided code, but rather by the test function itself
```

```
or the `parsePrint` method, which is not included in the source code. The
issue may be related to the treatment of large numbers or decimals, and th
e fact that the `add` function takes a string as an argument. The error me
ssage suggests that the comparison is between two strings, such as "a" and
"b".
Bug report for HIT07_33: A NullPointerException occurs when calling the ge
tClass() method on a null object in an array. The array is being properly
declared and used, but the issue arises when trying to dereference a null
member of the array without checking if it's null first. The error is caus
ed by the fact that the second element of the input array is null, and the
getClass() method cannot be called on a null object. The issue is not with
the variable "array" itself, but rather with the handling of null values w
ithin the array. The code should check if an object is null before trying
to call getClass() on it to prevent the NullPointerException.
Bug report for HIT08_54: The bug is that the input string "fr__POSIX" is n
ot in the correct format, which is expected to be "cc_CCCC" where "cc" is
the language code in lowercase and "CCCC" is the country code in uppercas
e, separated by an underscore. The issue arises when the code checks the t
hird character of the string, which is an underscore, and expects it to be
an uppercase letter. This causes the condition at line 115 to be true and
throws an IllegalArgumentException with the message "Invalid locale forma
t: fr__POSIX". The bug can be fixed by ensuring that the input string is i
n the correct format.
```

Readability score

For the readability score we use the Flesch-Reading-Ease. Ths is a score between 0 and 100. The higher the score is the easier readable the text is. Easily readable texts should have a score above 60. Flesch-Reading-Ease for English language is calculated with the following formula:

FRE = 206,835 - (1,015 * ASL) - (85,6 * ASW)

- ASL = Average Sentence Length
- ASW = Average Number of Syllables per Word

In [28]:

```python
from readability import Readability
import nltk
nltk.download('punkt_tab')

readability_gt = {}
for key, value in ground_truths.items():
    r = Readability(value*8)
    score = r.flesch().score
    readability_gt[key] = score
    print("Readability score for", key, "is", score)
```

```
Readability score for HIT01_8 is 44.01170243902442
Readability score for HIT02_24 is 43.392630662020935
Readability score for HIT03_6 is 46.69708319948606
Readability score for HIT04_7 is 33.23200362149856
Readability score for HIT05_35 is 52.29404791047389
Readability score for HIT06_51 is 47.7685698447894
Readability score for HIT07_33 is 51.7558152319657
Readability score for HIT08_54 is 49.59896782902139
```

```
[nltk_data] Downloading package punkt_tab to
[nltk_data]     /home/smilla.fox/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Semantic Similarity

To calculate the semantic similarity between two explanations we use the all-MiniLM-L6-v2 model from Hugging Face. This is an embedding model that encodes sentences and short paragraphs. The maximum number of words it can encode is 256. Our maximum ground truth text is a lot shorter than that. Therefore we expect the following generated explanations also to include less than 256 words. The embedding model maps the texts into a vector space which allows us to compare them using cosine similarity.

In [24]:

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
max_num_words = max([len(tokenizer.tokenize(text)) for text in ground_tru
print(f"The maximum number of words in a ground truth explanation  is {ma
```

```
The maximum number of words in a ground truth explanation  is 139.
```

In [26]:

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')


def sem_similarity(ground_truth, new):
    embeddings1  = model.encode(ground_truth)
    embeddings2 = model.encode(new)
    return cosine_similarity(embeddings1.reshape(1, -1), embeddings2.resh

print("Testing the semantic similarity function:")
print(sem_similarity("This is a test.", "This is also a test.")[0][0])
```

```
Testing the semantic similarity function:
0.8977157
```

Define Threshold

Readablity

As a readability threshold we will use the lowest readability score from the ground truth.

From calculating the readability scores for the ground truths we know that the values will not be very high. This is most likely because the texts topics are complex and it is difficult to explain them in a simple language. This is why we decided to use the ground truths as a baseline for readability and expect the following generated explanations to be at least as readable as the ground truths.

Semantic Similarity

For the semantic similarity threshold, we will rephrase the ground truth explanations using the same LLM as before so that they still include all information. Then we compute the similarity scores and choose the lowest score as the threshold. We chose this threshold because a good explanation should include all information that the ground truth contains.

In [29]:

```
readability_threshold = min(readability_gt.values())
print(f"The readability threshold is: {readability_threshold}")
```

```
The readability threshold is: 33.23200362149856
```

In [34]:

```
# ground_truths_rephrased = {}
# for key, value in ground_truths.items():
#     prompt = f"Please rephrase the following bug report so that it stil
#     ground_truths_rephrased[key] = assistant.generate_answer(prompt)
#     print(f"Rephrased report for {key}: {ground_truths_rephrased[key]}"
Rephrased report for HIT01_8: When setting the `minutesOffset` variable to -15, a valid value according to the code comments, an `IllegalArgumentException` is thrown due to a conditional statement on line 279. The statement incorrectly checks if `minutesOffset` is less than 0, despite comments indicating that negative values up to -59 are allowed if the hour is positive. To fix this, the condition should be updated to check if `minutesOffset` is less than -59 or greater than 59, ensuring the method can progress and invoke subsequent methods correctly.
Rephrased report for HIT02_24: An `IllegalArgumentException` is thrown when a color parameter falls outside the expected range due to the variable "g" being assigned a negative value, which is not accepted by the `Color` constructor. The calculation of "g" using the formula `(int) (lowerBound + (value - lowerBound) / (upperBound - lowerBound) * 255)` can result in a negative value if the input "value" is negative. However, the code fails to ensure that "value" is within the valid range of 0.0 to 1.0. Instead, it's suggested that the defined but unused variable "v" should be used in the calculation of "g". The correct value of "g" should be an integer between 0 and 255, corresponding to the valid color parameter range.
Rephrased report for HIT03_6: A StringIndexOutOfBoundsException error occurs when the variable "pos" exceeds the length of the input string, causing the code to attempt to access a non-existent character. This happens because "pos" is being incremented at a rate faster than the characters from the input string are being consumed, resulting in "pos" going out of range. The issue lies in the loop where "pos" is incremented, allowing it to surpass the string's length and leading to an exception being thrown when trying to access a character that does not exist.
Rephrased report for HIT04_7: The bug involves the incorrect update of the `maxMiddleIndex` variable, leading to an `AssertionFailedError`. The issue is likely in the `updateBounds` method, where `maxMiddleIndex` is set to the `index` parameter, rather than being calculated based on the `s` and `e` values from the `getDataItem` method. However, the `getDataItem` method is called with the incorrect parameter `this.minMiddleIndex` instead of `this.maxMiddleIndex`. Furthermore, the code has type mismatch issues and unused variables, contributing to the error. To fix this, the `maxMiddleIndex` calculation should be corrected to use the `s` and `e` values obtained from the `getDataItem` method with the correct parameter.
Rephrased report for HIT05_35: A ClassCastException occurs when attempting to cast an Object array to a String array. The issue arises from the `type` variable being set to `Object.class` when both `array` and `element` parameters are null. This happens due to the line `type = array != null ? array.getClass() : element != null ? element.getClass() : Object.class;`, which defaults to `Object.class` when both `array` and `element` are null. As a result, the `copyArrayGrow1` method returns an Object array, which cannot be cast to a String array, leading to the ClassCastException. The bug can be resolved by ensuring the correct type is set for the `type` variable, allowing the `copyArrayGrow1` method to return an array of the correct type.
Rephrased report for HIT06_51: A ComparisonFailure occurs when comparing two strings, but the issue lies not in the provided code's logic or conditional statements. Instead, the problem arises from the earlier parsing of a negative zero fractional portion using the `parsePrint` method. This method, not included in the source code, is used on the value being compared, and the parsed value may differ from the original value, leading to the ComparisonFailure. The error is likely related to the handling of large numbers or decimals, and the fact that the `add` function takes a string argument. The test function itself or the `parsePrint` method is the probable cause of the issue, which manifests as a comparison between two strings, such as "a" and "b", as indicated by the error message.
Rephrased report for HIT07_33: A NullPointerException occurs when attempting to call the getClass() method on a null object within an array. The arr
```

```
ay itself is properly declared and used, but the issue arises from derefer
encing a null element without first checking for nullity. Specifically, th
e second element of the input array is null, and the getClass() method can
not be called on a null object, resulting in the error. The problem lies n
ot with the "array" variable, but rather with the handling of null values
within the array. To prevent the NullPointerException, the code should inc
lude a null check before attempting to call getClass() on an object, ensur
ing that the method is only called on non-null objects.
Rephrased report for HIT08_54: The bug occurs because the input string "fr
__POSIX" does not conform to the expected format of "cc_CCCC", where "cc"
is the language code in lowercase and "CCCC" is the country code in upperc
ase, separated by an underscore. The issue arises when the code checks the
third character of the string, expecting an uppercase letter, but instead
finds an underscore. This triggers the condition at line 115, resulting in
an IllegalArgumentException with the message "Invalid locale format: fr__P
OSIX". To resolve the bug, the input string must be formatted correctly, a
dhering to the "cc_CCCC" structure, to prevent the invalid locale format e
rror.
```

In [8]:

```python
# import json
# with open("ground_truths_rephrased.json", "w") as f:
#     json.dump(ground_truths_rephrased, f)
```

In [31]:

```python
import json
with open("ground_truths_rephrased.json", "r") as f:
    ground_truths_rephrased = json.load(f)

for report in ground_truths_rephrased:
    print(f"Rephrased bug report for {report}: {ground_truths_rephrased[r
Rephrased bug report for HIT01_8: When setting the `minutesOffset` variabl
e to -15, a valid value according to the code comments, an `IllegalArgumen
tException` is thrown due to a conditional statement on line 279. The stat
ement incorrectly checks if `minutesOffset` is less than 0, despite commen
ts indicating that negative values up to -59 are allowed if the hour is po
sitive. To fix this, the condition should be updated to check if `minutesO
ffset` is less than -59 or greater than 59, ensuring the method can progre
ss and invoke subsequent methods correctly.
Rephrased bug report for HIT02_24: An `IllegalArgumentException` is thrown
when a color parameter falls outside the expected range due to the variabl
e "g" being assigned a negative value, which is not accepted by the `Color
` constructor. The calculation of "g" using the formula `(int) (lowerBound
+ (value - lowerBound) / (upperBound - lowerBound) * 255)` can result in a
negative value if the input "value" is negative. However, the code fails t
o ensure that "value" is within the valid range of 0.0 to 1.0. Instead, i
t's suggested that the defined but unused variable "v" should be used in t
he calculation of "g". The correct value of "g" should be an integer betwe
en 0 and 255, corresponding to the valid color parameter range.
Rephrased bug report for HIT03_6: A StringIndexOutOfBoundsException error
occurs when the variable "pos" exceeds the length of the input string, cau
sing the code to attempt to access a non-existent character. This happens
because "pos" is being incremented at a rate faster than the characters fr
om the input string are being consumed, resulting in "pos" going out of ra
nge. The issue lies in the loop where "pos" is incremented, allowing it to
surpass the string's length and leading to an exception being thrown when
trying to access a character that does not exist.
Rephrased bug report for HIT04_7: The bug involves the incorrect update of
the `maxMiddleIndex` variable, leading to an `AssertionFailedError`. The i
ssue is likely in the `updateBounds` method, where `maxMiddleIndex` is set
to the `index` parameter, rather than being calculated based on the `s` an
d `e` values from the `getDataItem` method. However, the `getDataItem` met
hod is called with the incorrect parameter `this.minMiddleIndex` instead o
f `this.maxMiddleIndex`. Furthermore, the code has type mismatch issues an
d unused variables, contributing to the error. To fix this, the `maxMiddle
Index` calculation should be corrected to use the `s` and `e` values obtai
ned from the `getDataItem` method with the correct parameter.
Rephrased bug report for HIT05_35: A ClassCastException occurs when attemp
ting to cast an Object array to a String array. The issue arises from the
`type` variable being set to `Object.class` when both `array` and `element
` parameters are null. This happens due to the line `type = array != null
? array.getClass() : element != null ? element.getClass() : Object.class;
`, which defaults to `Object.class` when both `array` and `element` are nu
ll. As a result, the `copyArrayGrow1` method returns an Object array, whic
h cannot be cast to a String array, leading to the ClassCastException. The
bug can be resolved by ensuring the correct type is set for the `type` var
iable, allowing the `copyArrayGrow1` method to return an array of the corr
ect type.
Rephrased bug report for HIT06_51: A ComparisonFailure occurs when compari
ng two strings, but the issue lies not in the provided code's logic or con
ditional statements. Instead, the problem arises from the earlier parsing
of a negative zero fractional portion using the `parsePrint` method. This
method, not included in the source code, is used on the value being compar
ed, and the parsed value may differ from the original value, leading to th
e ComparisonFailure. The error is likely related to the handling of large
numbers or decimals, and the fact that the `add` function takes a string a
rgument. The test function itself or the `parsePrint` method is the probab
le cause of the issue, which manifests as a comparison between two string
s, such as "a" and "b", as indicated by the error message.
Rephrased bug report for HIT07_33: A NullPointerException occurs when atte
mpting to call the getClass() method on a null object within an array. The
```

```
array itself is properly declared and used, but the issue arises from dere
ferencing a null element without first checking for nullity. Specifically,
the second element of the input array is null, and the getClass() method c
annot be called on a null object, resulting in the error. The problem lies
not with the "array" variable, but rather with the handling of null values
within the array. To prevent the NullPointerException, the code should inc
lude a null check before attempting to call getClass() on an object, ensur
ing that the method is only called on non-null objects.
Rephrased bug report for HIT08_54: The bug occurs because the input string
"fr__POSIX" does not conform to the expected format of "cc_CCCC", where "c
c" is the language code in lowercase and "CCCC" is the country code in upp
ercase, separated by an underscore. The issue arises when the code checks
the third character of the string, expecting an uppercase letter, but inst
ead finds an underscore. This triggers the condition at line 115, resultin
g in an IllegalArgumentException with the message "Invalid locale format:
fr__POSIX". To resolve the bug, the input string must be formatted correct
ly, adhering to the "cc_CCCC" structure, to prevent the invalid locale for
mat error.
```

In [33]:

```python
similarities = [sem_similarity(ground_truths[key], ground_truths_rephrase
min_similarity = min(similarities)
print(f"The semantic similarity threshold is: {min_similarity[0][0]}")
```

```
The semantic similarity threshold is: 0.8900765180587769
```

Find minimal number of explanations necessary

Now we want to find out how many explanations we need to reach our defined thresholds for readability and semantic similarity to the ground truth. To investigate this we chose one bug report as an example. For this bug report we have 40 explanations. We try start with generating a summary with only one explanation and then keep increasing the number of explanations up to 40.

One problem is that it is not only important how many explanations we choose but also which ones. If we choose 5 explanations that all contain the same information but not all necessary information the results might be worse than if we include 2 explanations that contain different information. This is why generated the summarized explanation 10 times for each number of explanations. For each run we randomly chose the explanations. Then we calculated the mean readablility and mean similarity for these 10 runs. This allows us to explore the effects of different amounts of explanations without dealing with which explanations should be selected.

In [15]:

```python
test_report = list(ground_truths.keys())[0]
len(grouped_explanations[test_report])
```

Out[15]:

```
40
```

Again we saved the generated explanations to a json file because it takes a very long time to generate all 400 explanations.

In [9]:

```python
# import random
# num_explanations = len(grouped_explanations[test_report])
# explanations_selection = {}
# for i in range(1, num_explanations):
#     explanations_selection[i] = []
#     for _ in range(10):
#         selected = random.sample(grouped_explanations[test_report], i)
#         prompt = f"The following explanations describe a bug. Please su
#         explanations_selection[i].append(assistant.generate_answer(prom

```

In [10]:

```
# import json
# with open("explanations_selection.json", "w") as f:
#     json.dump(explanations_selection, f)
```

In [34]:

```
import json
with open("explanations_selection.json", "r") as f:
    explanations_selection = json.load(f)
```

In [35]:

```
import numpy as np
similarities_selection = {}
mean_similarity_selection = {}
readability_selection = {}
mean_readability_selection = {}
std_dev_similarity = {}
std_dev_readability = {}

for i in explanations_selection.keys():
    similarities_selection[i] = []
    readability_selection[i] = []
    for idx in range(10):
        similarities_selection[i].append(sem_similarity(ground_truths[tes
        readability_selection[i].append(Readability(explanations_selectio
    mean_similarity_selection[i] = sum(similarities_selection[i])/len(sim
    mean_readability_selection[i] = sum(readability_selection[i])/len(rea
    std_dev_similarity[i] = np.std(similarities_selection[i])
    std_dev_readability[i] = np.std(readability_selection[i])
    print(f"Bug report for {test_report} with {i} explanations.\n Mean Si

Bug report for HIT01_8 with 1 explanations.
 Mean Similarity: [0.6002471],Mean Readability: -197.6082300239021
Bug report for HIT01_8 with 2 explanations.
 Mean Similarity: [0.7277787],Mean Readability: -103.55853077559357
Bug report for HIT01_8 with 3 explanations.
 Mean Similarity: [0.77308923],Mean Readability: 43.18297405499383
Bug report for HIT01_8 with 4 explanations.
 Mean Similarity: [0.7967463],Mean Readability: 48.28078284219371
Bug report for HIT01_8 with 5 explanations.
 Mean Similarity: [0.7980503],Mean Readability: 44.334004977936445
Bug report for HIT01_8 with 6 explanations.
 Mean Similarity: [0.77855605],Mean Readability: 42.19359677759483
Bug report for HIT01_8 with 7 explanations.
 Mean Similarity: [0.79175127],Mean Readability: 50.50788325080467
Bug report for HIT01_8 with 8 explanations.
 Mean Similarity: [0.79561764],Mean Readability: 51.66494063991429
Bug report for HIT01_8 with 9 explanations.
 Mean Similarity: [0.80562705],Mean Readability: 48.36786399523165
Bug report for HIT01_8 with 10 explanations.
 Mean Similarity: [0.7857667],Mean Readability: 45.610730978800916
Bug report for HIT01_8 with 11 explanations.
 Mean Similarity: [0.84231603],Mean Readability: 45.14719513800473
Bug report for HIT01_8 with 12 explanations.
 Mean Similarity: [0.77642715],Mean Readability: 51.476000194806716
Bug report for HIT01_8 with 13 explanations.
 Mean Similarity: [0.7680849],Mean Readability: 51.93373176148144
Bug report for HIT01_8 with 14 explanations.
 Mean Similarity: [0.7937468],Mean Readability: 50.731907816073274
Bug report for HIT01_8 with 15 explanations.
 Mean Similarity: [0.7894934],Mean Readability: 47.95557262020733
Bug report for HIT01_8 with 16 explanations.
 Mean Similarity: [0.78490865],Mean Readability: 50.2714003663671
Bug report for HIT01_8 with 17 explanations.
 Mean Similarity: [0.79094523],Mean Readability: 50.15775453221234
Bug report for HIT01_8 with 18 explanations.
 Mean Similarity: [0.743606],Mean Readability: 52.49400339743697
Bug report for HIT01_8 with 19 explanations.
 Mean Similarity: [0.7658137],Mean Readability: 52.28118995966548
Bug report for HIT01_8 with 20 explanations.
 Mean Similarity: [0.7872475],Mean Readability: 47.87998559108129
Bug report for HIT01_8 with 21 explanations.
 Mean Similarity: [0.79384553],Mean Readability: 52.32183412985968
Bug report for HIT01_8 with 22 explanations.
 Mean Similarity: [0.7555443],Mean Readability: 51.57151886666982
Bug report for HIT01_8 with 23 explanations.
 Mean Similarity: [0.75641453],Mean Readability: 54.39981410513572
Bug report for HIT01_8 with 24 explanations.
 Mean Similarity: [0.7833136],Mean Readability: 57.76749123299834
Bug report for HIT01_8 with 25 explanations.
 Mean Similarity: [0.77561116],Mean Readability: 59.112316812778616
Bug report for HIT01_8 with 26 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 27 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 28 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 29 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 30 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
```

```
Bug report for HIT01_8 with 31 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 32 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 33 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 34 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 35 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 36 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 37 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 38 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
Bug report for HIT01_8 with 39 explanations.
 Mean Similarity: [0.77160984],Mean Readability: 58.40038935245752
```

Results

We can now calculate for which number of explanations the average semantic similarity and the average readablity are the highest. We also the mean semeantic similarity scores and readability scores for each number of explanations. The red lines in the graphs represent the selected thresholds. The light blue area shows the standard deviation.

In [36]:

```
max_similarity = max(mean_similarity_selection.values())
max_similarity_key = max(mean_similarity_selection, key=mean_similarity_s
max_readability = max(mean_readability_selection.values())
max_readability_key = max(mean_readability_selection, key=mean_readabilit
print(f"The maximum similarity is {max_similarity} with {max_similarity_k
print(f"The maximum readability is {max_readability} with {max_readabilit
```

```
The maximum similarity is [0.84231603] with 11 explanations.
The maximum readability is 59.112316812778616 with 25 explanations.
```

In [41]:

```
similarity_above_thresh = [s for s in mean_similarity_selection.values()
print(f"The number of bug reports with similarity above the threshold is
```

```
The number of bug reports with similarity above the threshold is 0.
```

In [42]:

```
readability_above_thresh = [s for s in mean_readability_selection.values(
print(f"The number of bug reports with readability above the threshold is
```

```
The number of bug reports with readability above the threshold is 37.
```

In [55]:

```
keys_readability_above_thresh = [k for k, s in mean_readability_selection
min_passing_readablility = min(keys_readability_above_thresh, key=lambda

print(f"The minimum number of explanations that pass the readability thre
```

```
The minimum number of explanations that pass the readability threshold is
3.
```

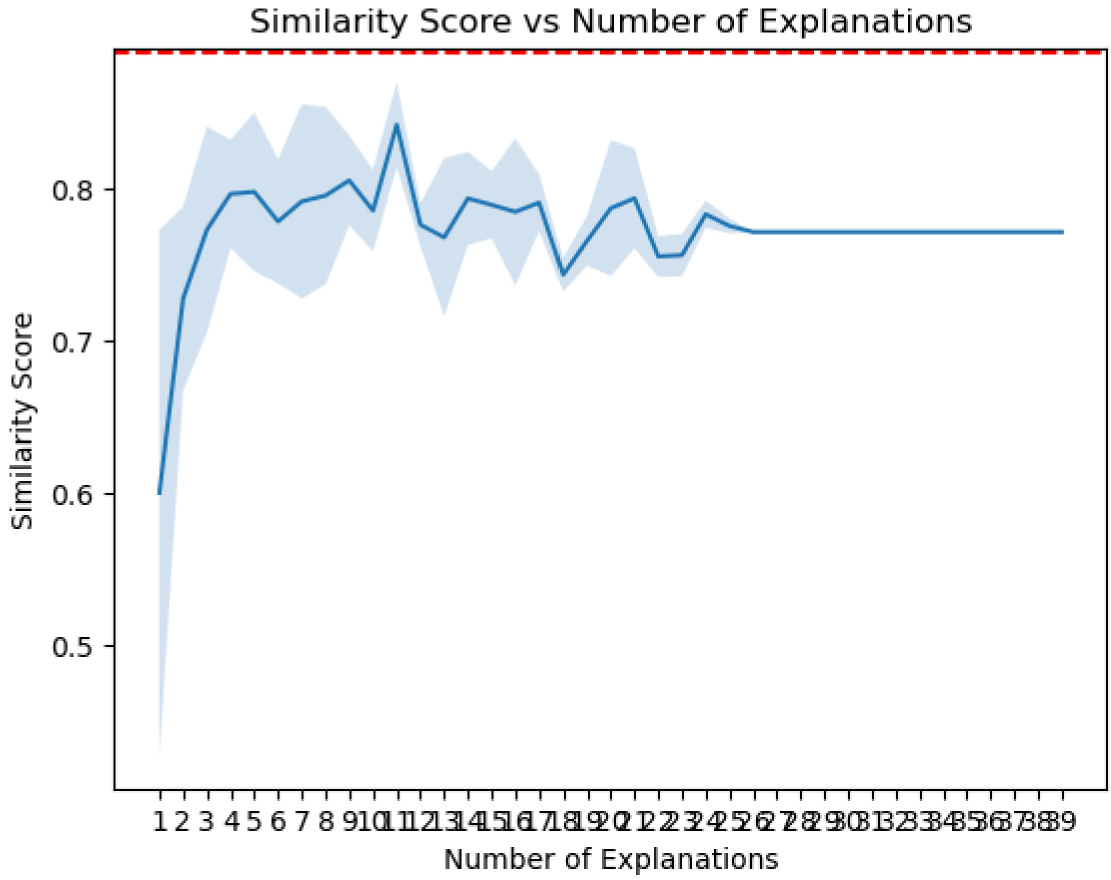
In [57]:

```
import matplotlib.pyplot as plt

plt.plot(mean_readability_selection.keys(), [v[0] for v in mean_similarit
# plot std dev
```

```
plt.fill_between(mean_readability_selection.keys(), [v[0] - std_dev_simil
plt.xlabel('Number of Explanations')
plt.ylabel('Similarity Score')
plt.title('Similarity Score vs Number of Explanations')

# Add line for threshold
plt.axhline(y=min_similarity, color='r', linestyle='--')
```

Out[57]:

```
<matplotlib.lines.Line2D at 0x7f6ccc3bd310>
```

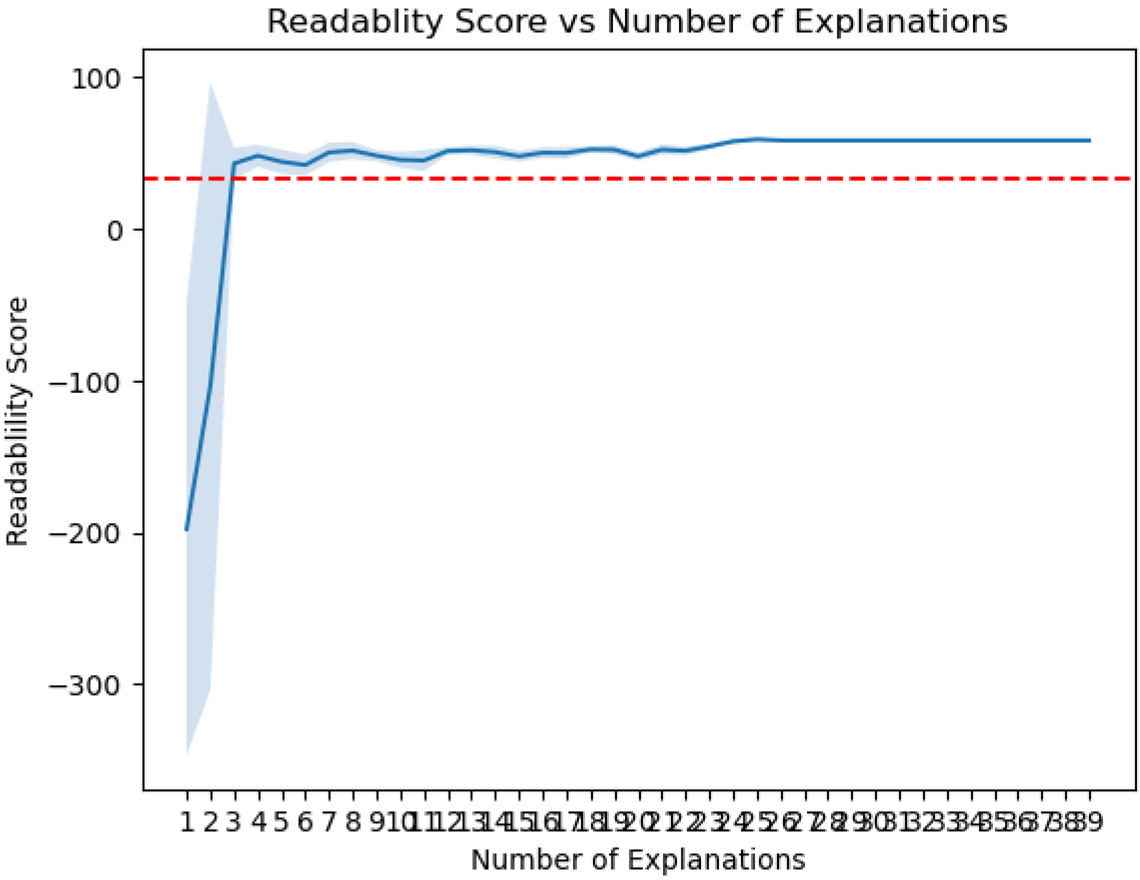


In [56]:

```
plt.plot(mean_similarity_selection.keys(),  mean_readability_selection.va
# plot std dev
plt.fill_between(mean_similarity_selection.keys(), [v - std_dev_readabili
plt.xlabel('Number of Explanations')
plt.ylabel('Readablility Score')
plt.title('Readablity Score vs Number of Explanations')

# Add line for threshold
plt.axhline(y=readability_threshold, color='r', linestyle='--')
```

Out[56]:

```
<matplotlib.lines.Line2D at 0x7f6cec1b4430>
```



Discussion of Results

Readability

The readability improves a lot between 1 and 3 explanations and then only gets a little better when adding more examples. The minimum number of explanations needed to pass the threshold is 3. The peak is reached at 25 examples.

When choosing very few explanations the chosen explanations might be very difficult to read which makes it hard for the LLM to create a well readable summary. With more examples there is a higher variety of ways how the same information can be phrased. This can help to create a readable summary. At some point enough readable explanations are included to generate a readable text from them and more explanations don't improve the readability a lot.

Semantic Similarity

The semantic similarity also improves a lot between 1 and 4 examples when more examples are added. With more examples the score does not get always get better when adding more examples but fluctuates a lot. A peak is reached at 11 examples then the scores decrease again when adding more examples.

When using very few explanations the chosen explanations might not include all aspects that are mentioned in the ground truth explanation. This is why the mean scores are low. The high fluctuation in the scores between 4 and 25 could be because the similarity to the ground truth depends a lot on which explanations are chosen as a base to generate the summary.

Above about 25 examples the semantic similarity scores and also the readability scores stay almost the same. The set of explanations chosen seems to be diverse enough for the model to have all necessary information. Additional explanations do not add anything new. Therefore the result stays nearly the same.

All scores are lower than the chosen threshold. This is probably because the words in the rephrased ground truth are still very similar to the ground truth. Even if the content of an explanation is the same as the ground truth the similarity score might not reflect this well enough. It is likely that the embedding model we use to calculate the semantic similarity has been trained on texts that are very different to the bug reports and therefore it cannot represent them well.