# Reflection

## Task 01

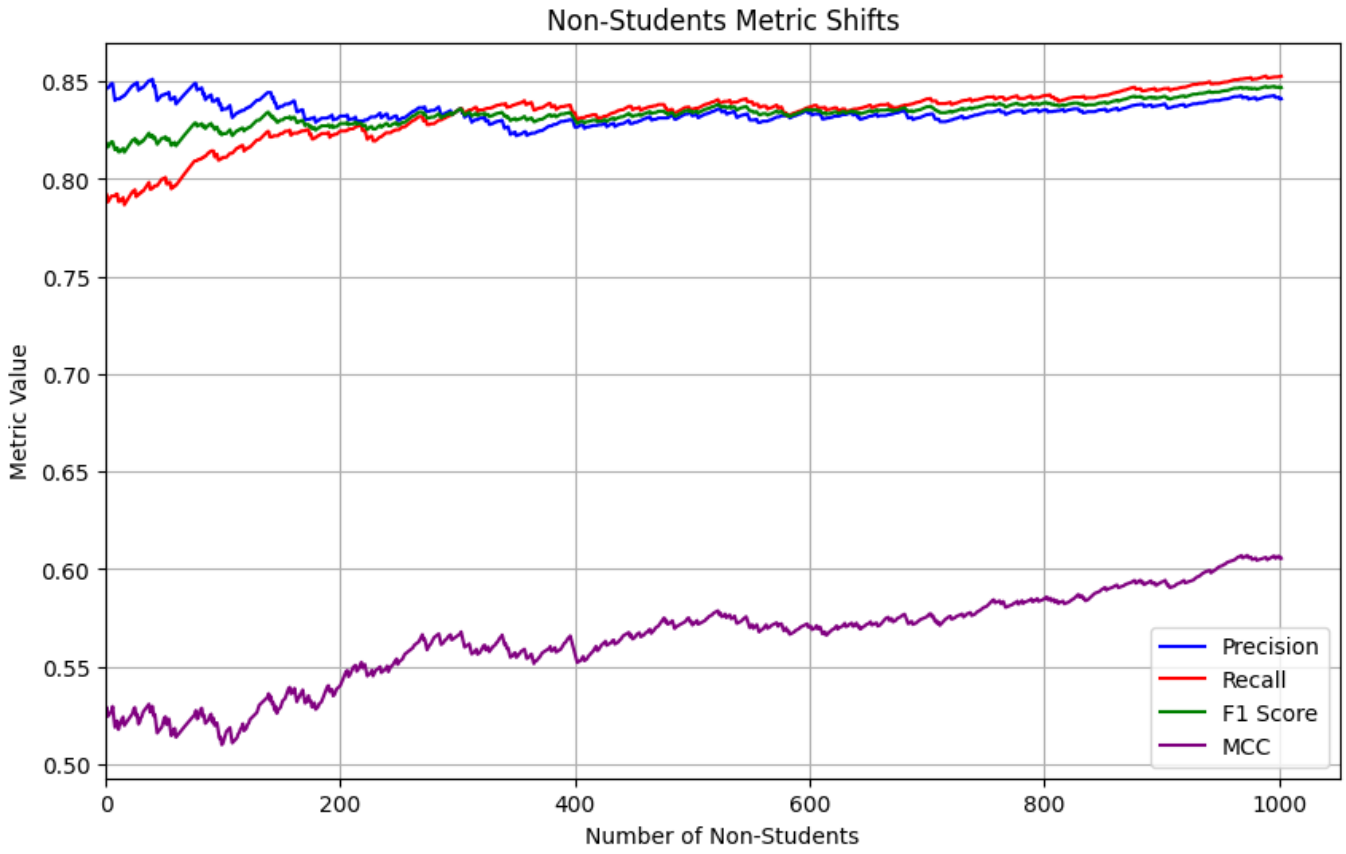
### Impact of Adding Non-Students to the Holdout Set

**Question 1.1:** How many "Non-Students" need to be added to the holdout set, on average, to observe a 5% or 10% impact on precision and recall?

Our analysis found no significant impact at the 10% level, even when all 1002 Non-Students were added to the holdout set. This suggests that our XGBoost classifier remains highly robust to dataset imbalance.
For the 5% level, we observed a slight impact on recall when adding all 1002 Non-Students.

To test the model's sensitivity, we deliberately increased overfitting potential by setting the `max_depth` to 200 and the number of estimators to 500. Despite these adjustments, performance on the holdout set remained strong.

The only metrics affected by adding all Non-Students were the **Recall** and **MCC** for Non-Students. Both increased consistently as more Non-Students were included in the holdout set. This **6% recall increase** can be attributed to distribution shifts. If Non-Students are easier to classify correctly than Students, recall naturally improves. The fact that the model was trained primarily on Students but still generalizes well to Non-Students suggests that the decision boundaries it learned for Students also apply effectively to Non-Students.




Additionally, this result indicates that the two groups do not differ significantly in terms of key features. To verify this, we conducted a **t-test for independence** on "Answer.duration," the model's most important

feature. The test yielded a **P-value of 0.926** with **2578 degrees of freedom**, suggesting strong similarity between the two groups for this feature. We expect similar results for other features as well.

We also measured the impact of adding only Professional Developers to the holdout set. This showed no significant difference compared to the Non-Students.

---

## Training a Model Exclusively on Non-Students

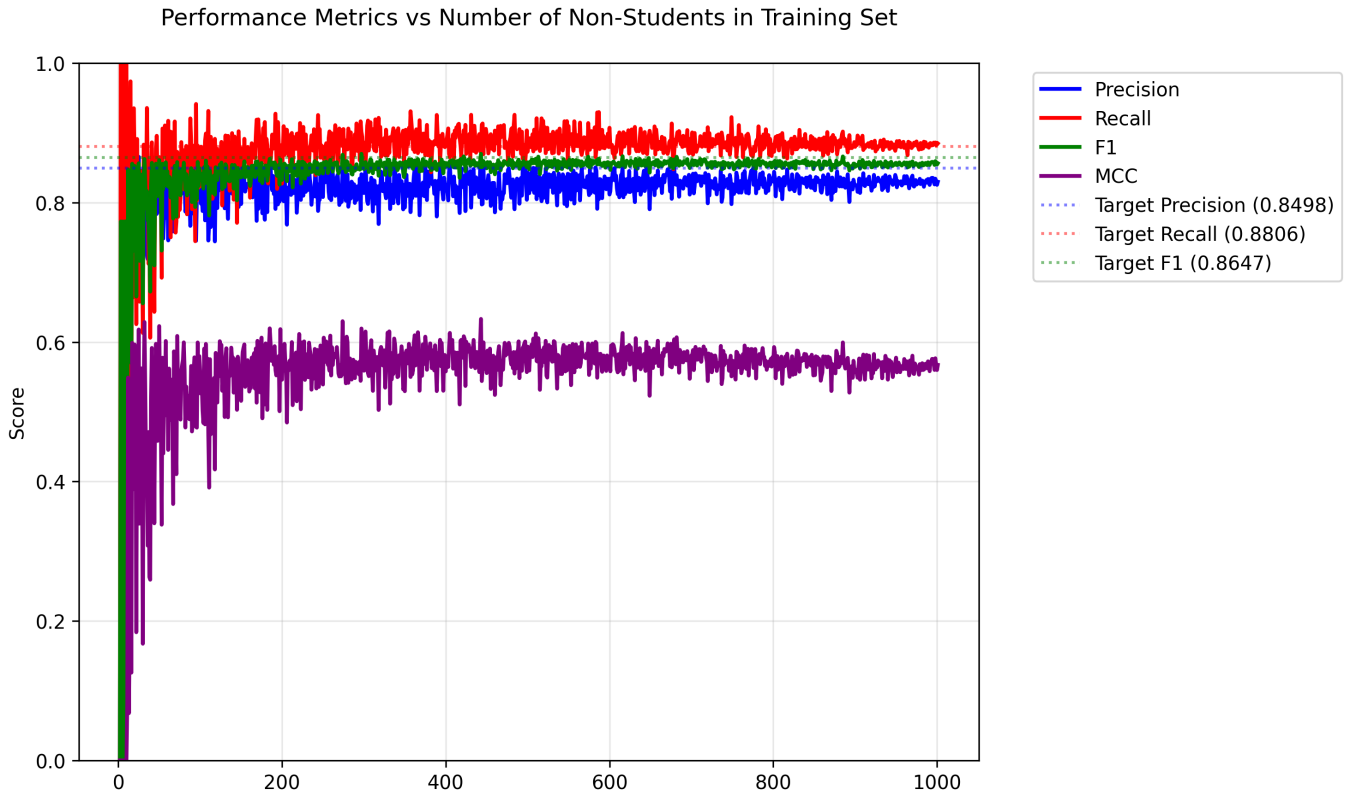
**Question 1.2:** What is the minimum number of Non-Students required to train a model that achieves similar performance to the mixed-data model from Mini Project 2?

After discovering that the Mini Project 2 model was trained on incorrect ground truth labels, we corrected this issue and updated the model accordingly. The revised model, available in **task1_project2_fixed.ipynb**, achieved the following **5-Fold Cross Validation** results:

- **Average Precision:** 0.8498 (+/- 0.0467)
- **Average Recall:** 0.8806 (+/- 0.0394)
- **Average F1:** 0.8647 (+/- 0.0300)

Since "similar" is not explicitly defined, we define a model as similar if its performance difference from the above results remains within **0.05**. This threshold was first met when **24 Non-Students** were included in the training set:

- **Precision:** 0.8643
- **Recall:** 0.8617
- **F1:** 0.8630




However, given the randomness in training data selection, this result may not be stable. As more Non-Students are added, performance fluctuations decrease, and the model's stability improves.

For training sets with **400+ Non-Students**, performance stabilizes within the following ranges:

- **Average Precision:** 0.80–0.84
- **Average Recall:** 0.87–0.89
- **Average F1:** 0.84–0.87

# Task 02

**Question 2.1 :** Define thresholds on readability and semantic similarity to reason about if and how many explanation(s) need to be added to the final consolidated explanations.

## Ground Truth Generation

For generating our ground truth, we utilized the deepseek r1 model with a specific prompt designed for bug fix consolidation:

```
You are a developer whose job is to consolidate bug fixes based on the answers
provided by colleagues of yours. Your professionalism is being measured by your
ability to describe the bug as crisp, but still correct as possible such that
another developer who actually has access to the codebase can fix the code in no
time, without having to read through too much of your explanation. Make sure that
your explanation contains all necessary and sufficient information to understand
and fix the bug. Do not provide information on how to fix the bug, as this is the
job of another developer. Only explain what the bug is.
```

Together with the prompt, we provided the model a text file with the explanations.

The model produced excellent results. Since LLMs don't have an assigned thinking process but generate text based on probabilities, providing space for "thinking" output allows the model to focus on the final answer without including the reasoning process. Without this approach, the model tends to incorporate reasoning processes in the final answer, leading to suboptimal results.

## Sampling Strategy

Our sampling approach utilized random sampling with a fixed seed to obtain different counts of explanations. For smaller sample sizes (1 and 3), we conducted sampling three times due to the high variance in explanation quality, ensuring a better representation. Larger sample sizes only required one sampling iteration since they naturally included more explanations from the total pool.

## Consolidation Process

For the consolidation process, we used ChatGPT-4o using the same prompt as the ground truth generation. We provided the sampled explanations alongside the prompt and used a new chat context for each request to ensure consistent results.

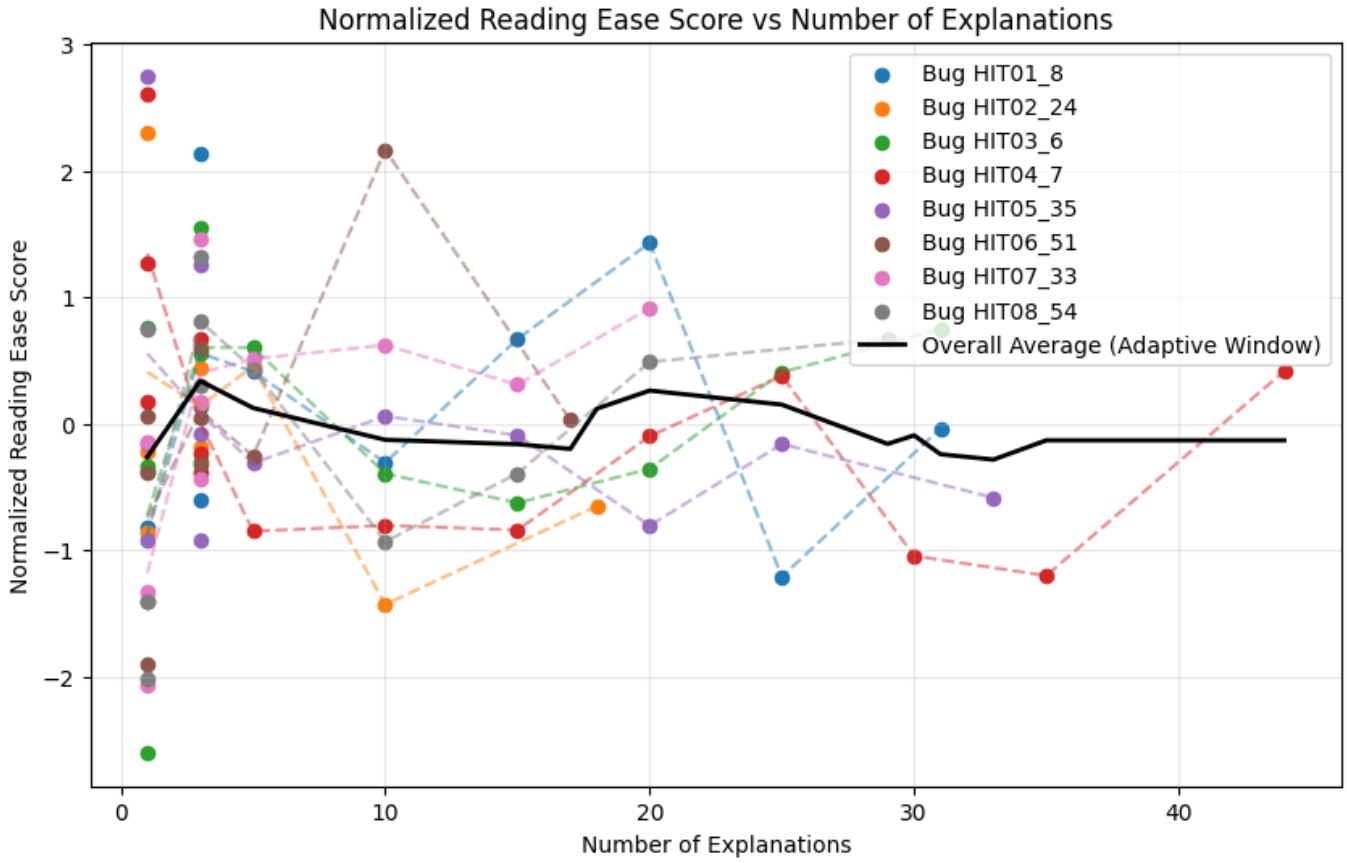
## Readability Metric

We chose the Flesch Reading Ease score for several reasons. First, it provides an intuitive interpretation on a 0-100 scale, where higher scores indicate easier readability. Second, the metric effectively balances both

sentence and word complexity by considering sentence length and syllable count per word. Finally, research has demonstrated strong correlations between Flesch Reading Ease scores and actual reading comprehension levels across various age groups.

## Readability Results

In our analysis of the consolidated explanations' readability, we normalized the scores by the mean readability score for each bug to exclude variance due to different bugs. This allows us to correctly analyze how the readability of the consolidated explanations changes with the number of explanations.



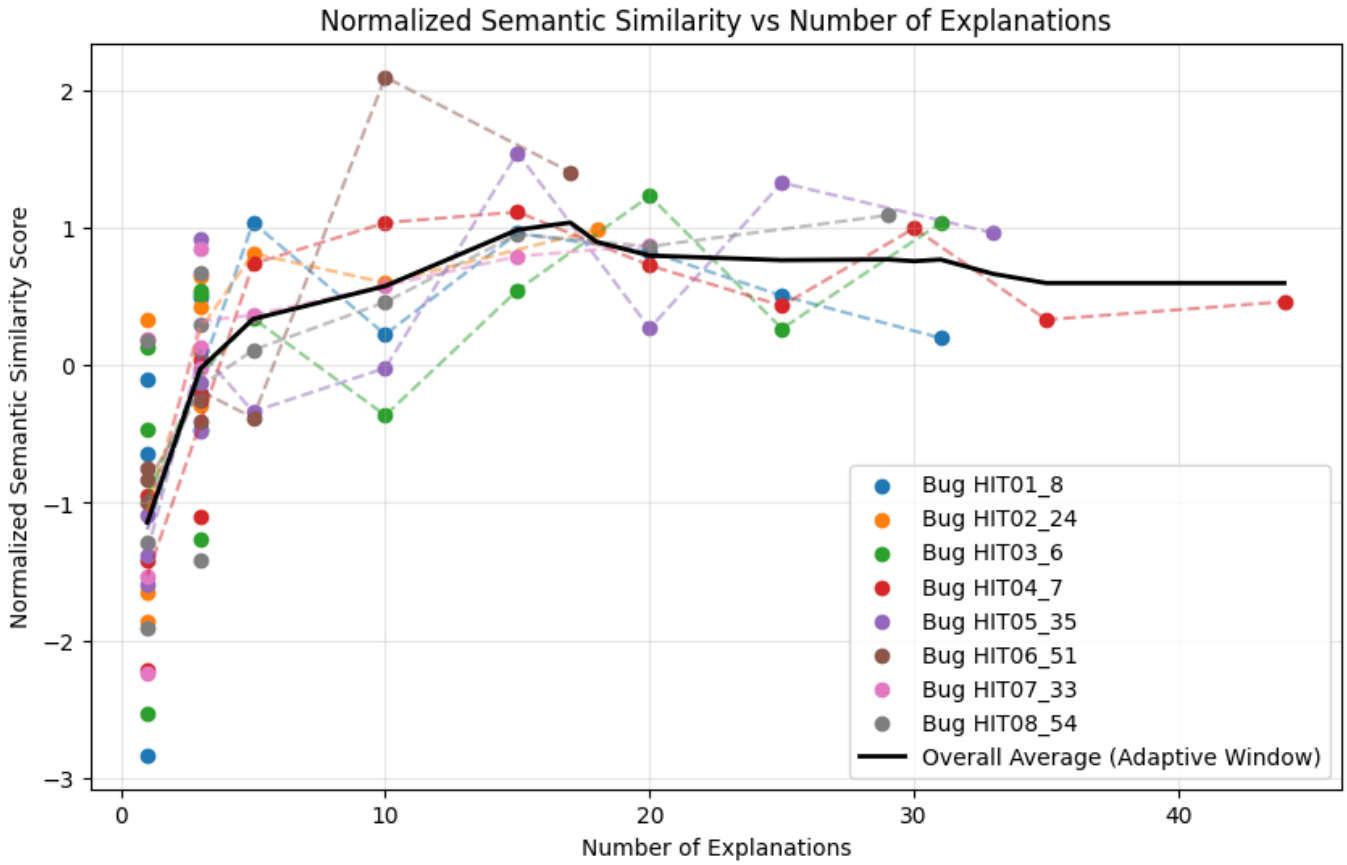

The results indicate that the overall readability remains relatively stable regardless of the number of explanations. However, we observed that the variance in readability scores decreases as the number of explanations increases. This suggests that for fewer explanations, the result heavily depends on the quality and information density of the given explanations.Also, it is important to note that the readability does not have any indication of the correctness of the explanations. Short and simple explanations might be readable, but do not have to include any relevant information. Larger sets of explanations tend to produce more consistent readability scores and are less dependent on the quality of the individual explanations.

## Semantic Similarity Metric

For semantic similarity measurement, we employed cosine similarity, which measures the similarity between two vectors. We converted the explanations into embeddings using the `sentence-transformers/all-mpnet-base-v2` model. Research has demonstrated that this embedding approach effectively captures semantic meaning by positioning semantically similar texts close to each other in the embedding space. The cosine similarity measurement then allows us to quantify the semantic relationships between these embeddings.

### Semantic Similarity Results

The semantic similarity analysis yielded particularly interesting results. After normalizing by the mean semantic similarity score for each bug, we observed several key patterns.




Similar to readability, the variance in semantic similarity decreases with more explanations, indicating that fewer explanations make the semantic similarity highly dependent on individual explanation quality.

The similarity score shows a distinct pattern: starting very low with one explanation, it increases rapidly with additional explanations, peaking at 15 explanations before slightly declining. This pattern suggests that there is a minimum number of explanations that is needed to achieve a good consolidation. Also, the similarity isn't solely dependent on explanation quantity but also quality, as some consolidated explanations with fewer inputs (1 or 3) demonstrated higher similarity to the ground truth than those with more explanations.

### Conclusion

Based on our analysis, while the number of explanations doesn't significantly impact readability scores, but to decrease the variance, a minimum of 5-10 explanations are needed. The semantic similarity however is notably influenced by the number of explanations. The data indicates optimal results are achieved with approx. 15 explanations.

## Task 03

**Question 3.1 :** how would you measure diversity? E.g., entropy of each feature

There are different metrics to determine the diversity of a dataset. As we want this metric to be based on the demographics of the study participants and the answer attributes, we do not consider the *Answer.explanation*

feature itself, readability, or semantic similarity score at this point (see 3.2 and 3.3 how these are considered when determining the subset).

In our dataset, this results in 148 remaining features. Of these 148 features, only 13 features are individual features (ordinal, numerical, encoded categorical features). The other features were originally two features ("*Worker.whereLearnedToCode*", "*Worker.programmingLanguage*") that were then one-hot-encoded.

Due to the nature of our data, we choose two different metrics for the one-hot-encoded and remaining features and weight them. As the one-hot encoded columns represent two features from our original dataset, their weight is equal to 2/15~0.13.

- weight_one_hot_encoded_features = 0.13
- weight_numerical_features = 0.87

For the one-hot-encoded features we use the Jaccard similarity to assess diversity among the one-hot encoded features. This metric directly computes the proportion of shared active features between pairs of data instances. In other words: it measures how similar the values of the one-hot-encoded rows are. The result is an interpretable measure of overlap - the higher the score, the higher the overlap. We treat this overlap as an approximation towards diversity by computing the Jaccard diversity score as 1 - Jaccard similarity.

For the remaining features, we choose entropy as a simple way to measure how uniformly the values are distributed. The higher the entropy, the more diverse we consider the subset to be.

Diversity_total = weight_numerical_features * entropy_score + weight_one_hot_encoded_features * Jaccard_diversity_score

For the dataset that we obtained in mini project 02, this results in the following scores:

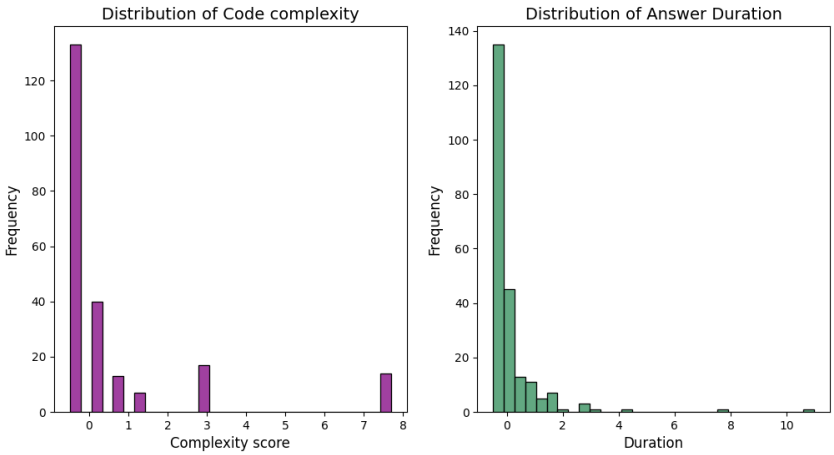
```
Normalized Feature Entropy for non-hot-encoded features:
FailingMethod: 0.679
Answer.duration: 0.230
Answer.confidence: 0.864
Answer.difficulty: 0.973
Answer.order: 0.927
Code.LOC: 0.546
Code.complexity: 0.470
Worker.score: 0.921
Worker.profession: 0.888
Worker.yearsOfExperience: 0.713
Worker.age: 0.783
Worker.gender: 0.758
Worker.country: 0.610

Entropy Score averaged: 0.72

Jaccard Similarity for one-hot-encoded features: 0.38
Jaccard Diversity: 0.62

Final Weighted Diversity Score: 0.71
```

The metrics show that the entropy-based diversity is rather high for most features. However, it is low (<0.50) for the code complexity and the duration needed to answer a question. This observation is true to our data as the plots below show. Both distributions are highly skewed. The diversity of the one-hot-encoded features is also rather high. This was to be expected due to our approach to one-hot-encoding where many columns were created. To get a better estimate in future projects, we would suggest to use an llm to consolidate answers further first, one-hot-encode next, and lastly derive the diversity based on fewer consolidated columns.




---

**Question 3.2 :** what is the max readability and semantic similarity independent of the diversity?

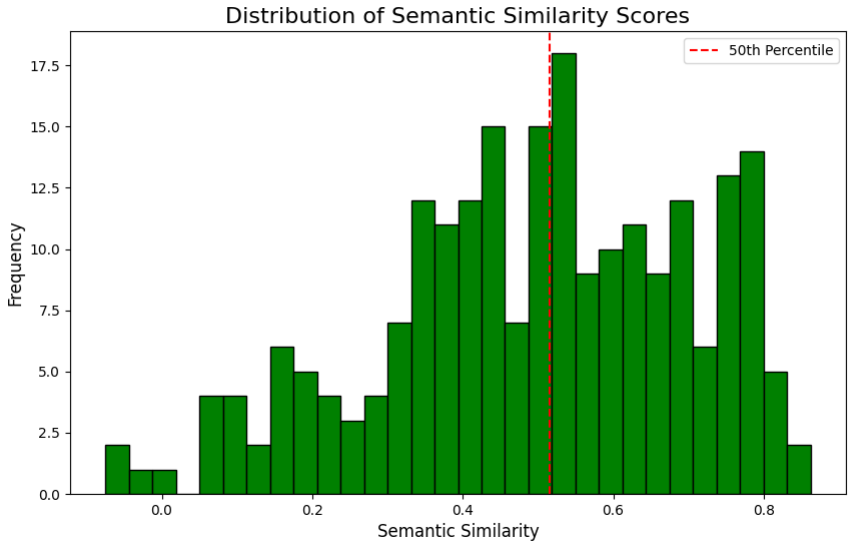
As introduced in Task 2, we use the Flesch-Reading-Score for readability, and the cosine similarity for semantic similarity rating. However, in this task, we calculate both scores for each data entry. This is equivalent to one readability and one semantic similarity score per row. We calculate these scores based on the *Answer.explanation* column of our preprocessed dataframe from mini project 02. We only take correct answers given (*TP==1*) into account.

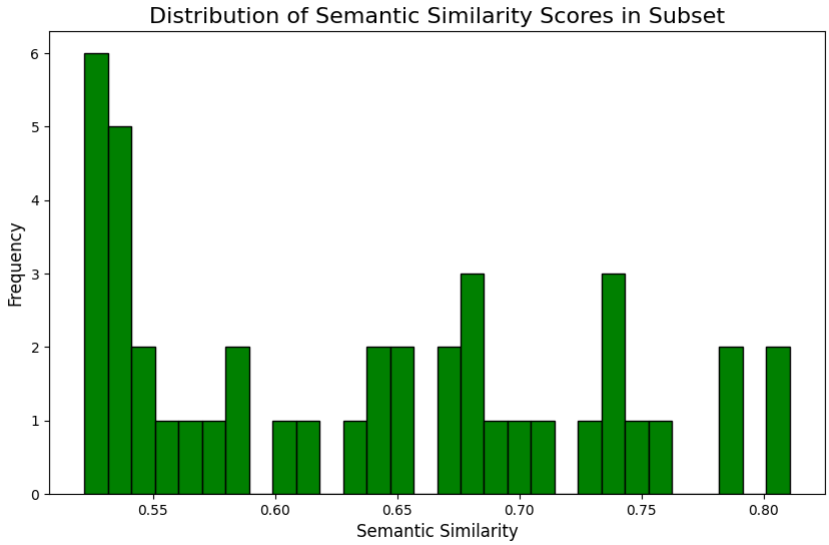
To determine our max values, we set a threshold for the 0.5 quantile for each score:

- threshold_readability_0.5 = 70.55
- threshold_semantic_similarity_0.5 ≈ 0.52

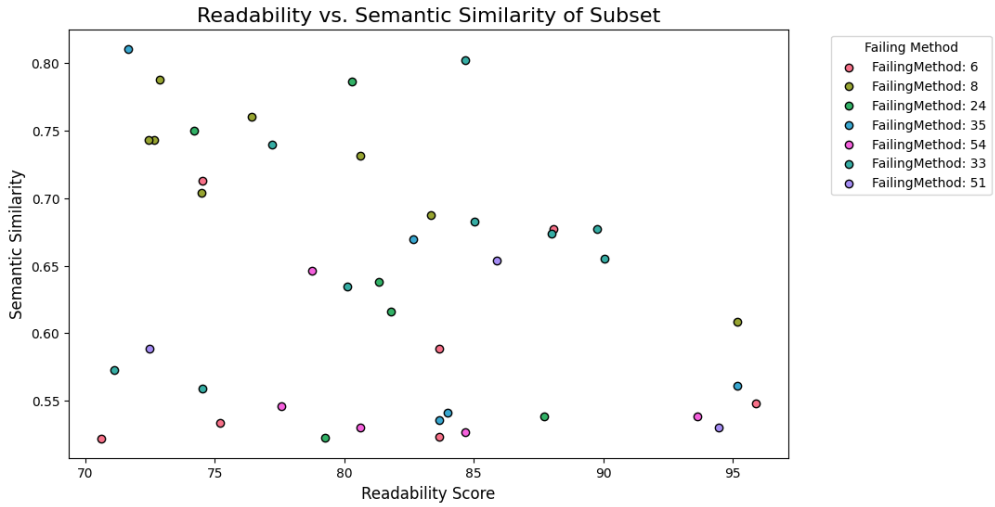
As a result, we obtain a subset with 43 entries where both scores are >= the 0.5 quantiles.


Distribution of Readability Scores
50th Percentile
Frequency
Readability Score
0
5
10
15
20
25
30
0
20
40
60
80
100
120

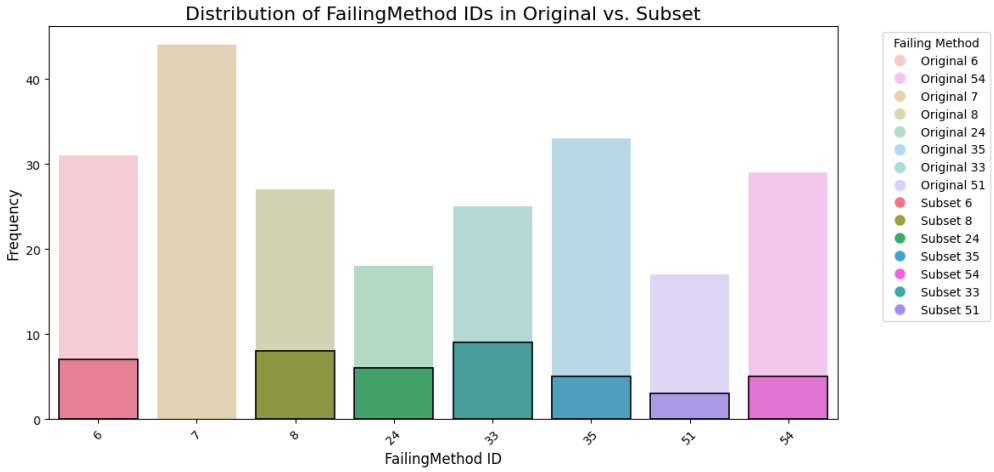

Distribution of Semantic Similarity Scores
50th Percentile
Frequency
Semantic Similarity
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0.0
0.2
0.4
0.6
0.8


The graphic below shows the distributions of data entries based on the readability and semantic similarity scores. It pictures that, for example, those entries with high readability scores were not included in the subset. Further investigation has shown that this is indeed a desired behaviour. Due to the nature of the readability score calculation, higher scores are also obtained, if only few (e.g. one word) are given in the *Answer.explanation* column. By combining the semantic of the answer given, this bottleneck can be avoided.



The colours of the data points signify the different bug IDs (*FailingMethod*). We can see that the highest scores are distributed across the different IDs and do not seem to be dependent on one ID only. To get a better understanding of the subset according to the IDs, we can refer to the following plot. The plot shows the count of correct entries by FailingMethod / Bug ID of the subset and original dataset. We can see that ID 7 is not part of the subset while it has the highest count in the original dataset. The reason for this is plausibly that we have used different prompts of ground truth generation. As a result, for bug ID 7, the ground truth is a multi-step list that explains how to fix the bug. In comparison, ID 33 has the highest share of entries in the subset. Here, the ground truth explanation is a crisp and short description of the bug. Both ground truth entries can be found below. This effect highlights the importance of prompting and ground truth generation. As we want to proceed with our work based on the results from mini project 02, we will nonetheless continue with this subset that excludes ID 07.




---

```
** Ground Truth ID Failing Method 7 **

1. Incorrect Index Usage in Data Retrieval:
Variables s (start time) and e (end time) are initialized using
this.minMiddleIndex (e.g., lines 299, 301) instead of this.maxMiddleIndex in
getDataItem() calls. This leads to incorrect calculation of maxMiddleIndex and
assertion failures.

2. Method Signature Mismatch:
TimePeriodValues.add() is called with two parameters (e.g., SimpleTimePeriod and
double), but the method expects a single TimePeriod argument, causing
compilation/runtime errors.

3. Uninitialized/Unupdated Count Variable:
A test asserts count equals 1, but count is initialized to 0 and never modified,
resulting in assertion failures. Another note suggests the expected value should
be 3 instead of 1.

4. Division by Zero Risk:
Calculation (e - s) / 2 (or similar) may involve division by zero if e == s,
though exact context is unspecified.

5. Type Mismatch in Method Calls:
updateBounds() is called with an int parameter where a different type (e.g.,
TimePeriod) might be expected, causing type errors.
```

```
6. Variable Confusion in Middle Calculation:
Variable e in maxMiddle calculation references this.minMiddleIndex (line 301)
instead of this.maxMiddleIndex, skewing results.
```

```
** Ground Truth ID Failing Method 33 **

A NullPointerException occurs on line 910 when iterating through an array. The
code dereferences array[i] (via getClass()) without checking if the element is
null. Specifically, if array[i] is null (e.g., the second element in the test
input), calling getClass() throws the
exception. The loop lacks a null check before accessing the method.
```

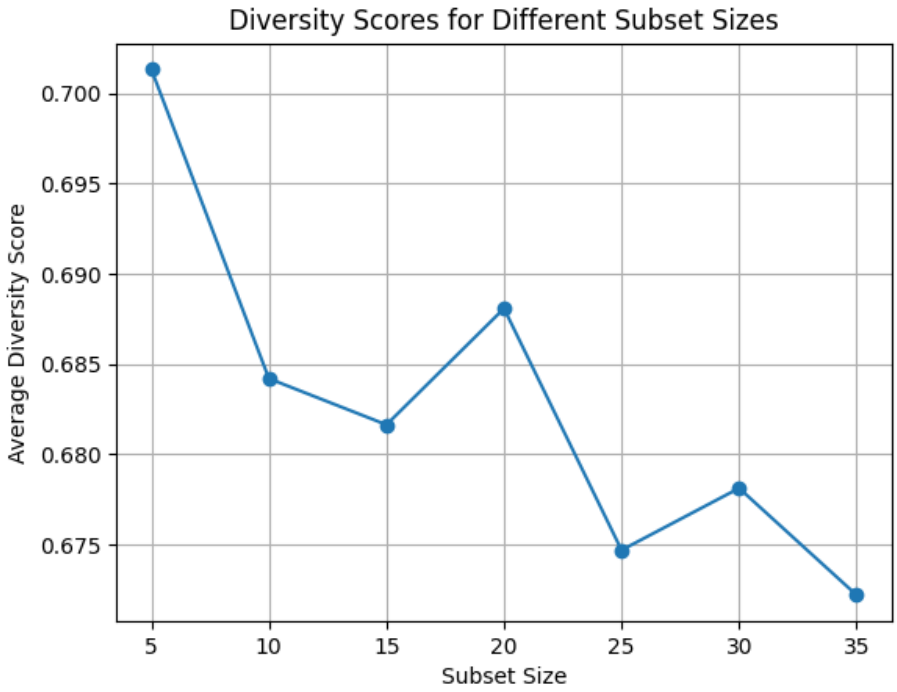
---

**Question 3.3 :** what is the max diversity for (previously achieved) max semantic similarity ? ( compromising readability)

In this task, we want to find the number of entries in our subset (i.e. a subset of the subset) that maximizes diversity. We assume that this maximization can only be reached by optimizing the number of entries (rows), not the number of features (columns), i.e. we consider our number of features fixed.

In the previous task 3.2, we have created a subset of our original dataset that satisfies both thresholds for readability and semantic similarity. This subset consists of 43 entries where all readability and semantic similarity scores are above their 0.5 quantile. Based on this subset, we optimize the diversity by applying the diversity score calculations form task 3.1. Therefore, we sample random subsets of our 43-sample-subset in an iterative approach. We sample 100 times for different number of samples ([5, 10, 15, 20, 25, 30, 35, length of the subset]) in the new subset. The subset with the highest diversity score is then taken and saved. We have observed that the smaller the subset, the higher the diversity. As we calculate the diversity by comparing the different rows, this is an expected behaviour. An alternative for future projects would be to calculate a diversity metric per row and then take the average of the sampled set.

The final subset details can be found here. The final subset can be found accessed here.


Diversity Scores for Different Subset Sizes
Average Diversity Score
0.700
0.695
0.690
0.685
0.680
0.675
5
10
15
20
25
30
35
Subset Size