

## **Mini Project 2: Pipeline of consolidating bug report explanations**

### **Outcome of consolidating bug report explanations:**

As a complexity measure we chose TTR (token-type-ratio). It measures the diversity of the vocabulary used in a piece of text – here our explanations. We chose the metric as it is easy to comprehend and implement. As a trade-off the TTR can vary drastically with different text lengths. Nevertheless, the other suggested metric, i.e., the Halstead Volume, does not seem to fit the application of summarizing natural language explanations. The Halstead Volume specifically was incepted as a metric for code.

We chose a Random Forest Classifier as our decision tree approach because it should be more robust and less prone to overfitting than a single decision tree. For the training the outcome precision and recall metrics were already quite poor:

Cross-Validation Precision: 0.22857142857142856

Cross-Validation Recall: 0.0380952380952381

Cross-Validation Metrics per Bug Report:

HIT03\_6: Precision=0.44, Recall=0.09

HIT04\_7: Precision=0.08, Recall=0.02

HIT05\_35: Precision=0.25, Recall=0.03

HIT06\_51: Precision=0.11, Recall=0.02

HIT07\_33: Precision=0.60, Recall=0.05

HIT08\_54: Precision=0.14, Recall=0.02

This is likely because of the relatively small amount of data and because the distribution of the ground truth (training set - 0: 1840, 1: 420) is heavily skewed.

It was therefore of little surprise that the model also performed poorly on the holdout set:

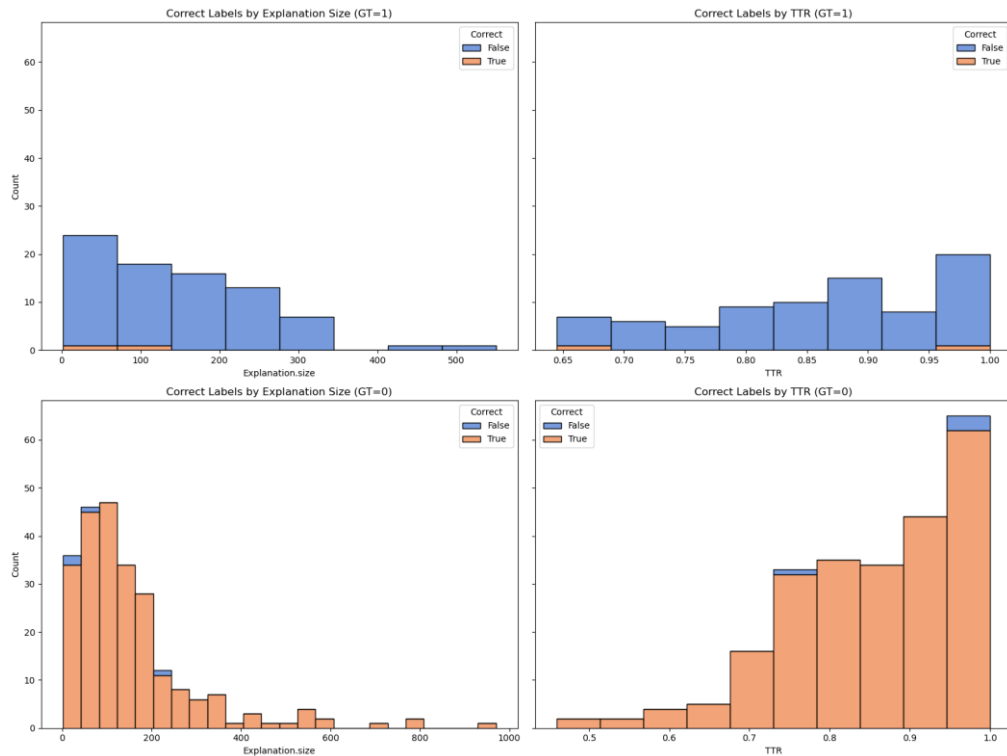
Holdout Precision: 0.3333333333333333

Holdout Recall: 0.025

Holdout Set Metrics per Bug Report:

HIT01\_8: Precision=0.25, Recall=0.03

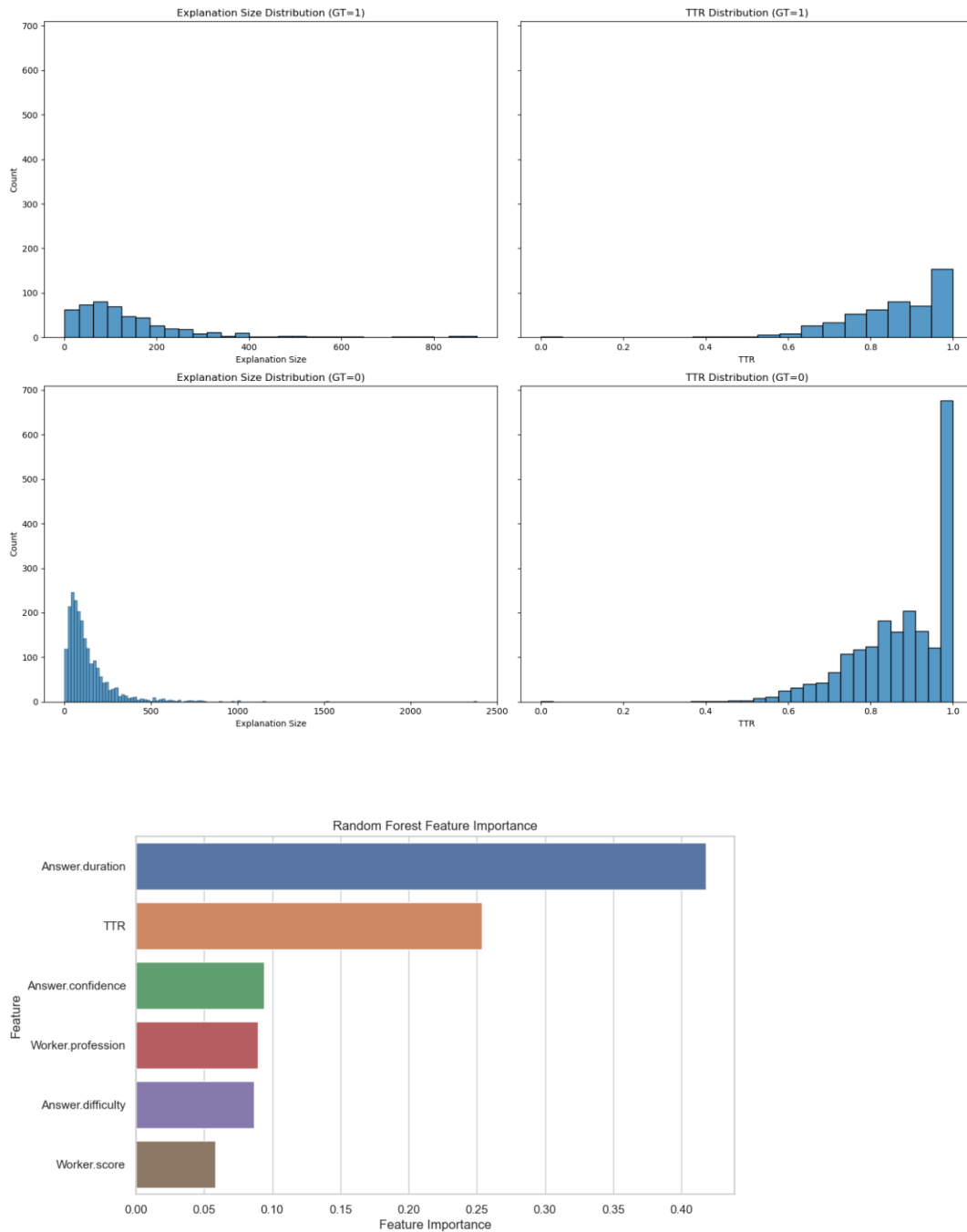
HIT02\_24: Precision=0.50, Recall=0.03



We created four plots analysing the detection performance from our Random Forest classifier on the holdout set, the top row shows the cases where the ground truth is one, the other cases are in the bottom row. It is apparent that our classifier is heavily biased towards predicting 0. In fact, we only predict the label to be 1 in six out of 320 cases. A more balanced dataset or techniques like data augmentation could help mitigate this bias.

Looking at the correctly as 0 identified bug reports we can see that most explanations were rather short. The TTR, on the other hand, seemed to be on the bigger side. This may suggest that these are traits of wrong bug reports. However, the correct (GT=1) examples follow similar trends. This, thus, seems to be coherent with general characteristics of the data and not a strong predictor. It is, however, noteworthy that our model relied on the TTR as the second most important feature, which is in line with the poor performance.

Stronger trends that differentiate between both labels may once again be observable with more data from positive examples.



To compare the results, we counted the words of the final answer, calculated the TTR on the final answer, assessed Readability and Generation Speed by hand and identified whether there was a suggestion how to fix the bug in the generated summary.

In the first prompt we adopted the method Chain of Density proposed by Adams et al. To our use case:

Adams, Griffin & Fabbri, Alexander & Ladhak, Faisal & Lehman, Eric & Elhadad, Noémie. (2023). From Sparse to Dense: GPT-4 Summarization with Chain of Density Prompting. 10.48550/arXiv.2309.04269.

You will generate increasingly concise, entity-dense summaries of the above answer explanations.

Repeat the following 2 steps 5 times.

Step 1. Identify 1-3 informative entities (";" delimited) from the article that are missing from the previously generated summary.

Step 2. Write a new, denser summary of identical length which covers every entity and detail from the previous summary plus the missing entities.

A missing entity is:

- Relevant to the main answer explanations,
- Specific yet concise (5 words or fewer),
- Novel (not in the previous summary),
- Faithful (present in the answer explanations),
- Anywhere (can be located anywhere in the answer explanations).

Guidelines:

- The first summary should be long (4-5 sentences, ~80 words) yet highly non-specific, containing little information beyond the entities marked as missing. Use overly verbose language and fillers (e.g., "these answer explanations discuss") to reach ~80 words.
- Make every word count: rewrite the previous summary to improve flow and make space for additional entities.
- Make space with fusion, compression, and removal of uninformative phrases like "the article discusses".
- The summaries should become highly dense and concise yet self-contained, i.e., easily understood without the answer explanations.
- Missing entities can appear anywhere in the new summary.
- Never drop entities from the previous summary. If space cannot be made, add fewer new entities.

Remember, use the exact same number of words for each summary.

Chain of Density: <https://chatgpt.com/share/677ffb6e-2ce0-8013-8cd6-72e277b5b6f1>

The "g" variable, scaled within a 255 range and cast to int, impacts the Color class, triggering IllegalArgumentException if bounds fail. this.lowerBound and this.upperBound inadequacies prevent sanitization, allowing negatives. Misusing "value" over "v" (line 117) worsens errors.

Reliance on `Math.max()` and `Math.min()`, double types, and input consistency is emphasized. The `Color(float, float, float)` constructor and `RuntimeException` encapsulate broader dependency issues in parameter handling.

The second prompt is just free form, how we would design the prompt. It is much simpler.

```
I have a list of explanations of where there is a bug in a given code statement. Please merge the explanations in a way that minimizes redundant information, while keeping the information that would be necessary for someone else to fix the bug.'
```

```
• Types of information that, if present in the explanation, should be preserved - how the program works, how the failure is happening, what is problem in the code, etc.
```

```
Be short, concise, do not use verbose language.
```

Simple: <https://chatgpt.com/share/677ffba6-c3b0-8013-b189-b514d59f14b9>

## Final Result:

### Explanation of the Bug:

- How the Program Works: The method involves computing the variable `g` based on `value`, constrained by `lowerBound` and `upperBound`. The computed `g` is passed to the `Color` constructor.
- The method involves computing the variable `g` based on `value`, constrained by `lowerBound` and `upperBound`. The computed `g` is passed to the `Color` constructor.
- How the Failure Happens: The calculation of `g` may result in a negative or out-of-range value when `value` is outside the `lowerBound`-`upperBound` range or when no proper sanitization ensures that `value` falls between 0 and 255.
- The calculation of `g` may result in a negative or out-of-range value when `value` is outside the `lowerBound`-`upperBound` range or when no proper sanitization ensures that `value` falls between 0 and 255.
- Problem in the Code: The variable `v` is defined to sanitize `value` within `lowerBound` and `upperBound`, but it is never used. Instead, `value` is directly used in the computation of `g`, leading to potential invalid inputs for the `Color` constructor. The `Color` constructor expects `g` to be between 0-255 (or 0.0-1.0 for float parameters), but the lack of proper sanitization allows invalid values, causing an `IllegalArgumentException`.
- The variable `v` is defined to sanitize `value` within `lowerBound` and `upperBound`, but it is never used. Instead, `value` is directly used in the computation of `g`, leading to potential invalid inputs for the `Color` constructor.
- The `Color` constructor expects `g` to be between 0-255 (or 0.0-1.0 for float parameters), but the lack of proper sanitization allows invalid values, causing an `IllegalArgumentException`.
- Recommendations for Fixing: Replace `value` with `v` in the calculation of `g` to ensure it falls within the acceptable range defined by `lowerBound` and `upperBound`. Add explicit checks

or constraints to guarantee value or v always satisfies the expected input range for the Color constructor.

- Replace value with v in the calculation of g to ensure it falls within the acceptable range defined by lowerBound and upperBound.
- Add explicit checks or constraints to guarantee value or v always satisfies the expected input range for the Color constructor.

This explanation captures all relevant details, enabling effective debugging and resolution.

For the third prompt (Summarize) we just asked the LLM to summarize and nothing more.

Summarize

Summarize: <https://chatgpt.com/share/677ffbd1-cbfc-8013-9d65-89e0a228c937>

### Final Result:

The text appears to be a discussion or troubleshooting session related to a programming issue, specifically involving color calculations and the use of a Color class in Java. Here's a summary:

- Issue Highlighted: The problem involves passing invalid values (outside the acceptable range) to a Color constructor, causing an IllegalArgumentException.
- Key Variables:
  - value: A variable that may exceed its intended bounds due to lack of proper sanitization.
  - g: Derived from value, potentially outside the range of 0-255 expected for the Color object.
  - v: A correctly constrained variable that should be used in calculations but is overlooked.
- value: A variable that may exceed its intended bounds due to lack of proper sanitization.
- g: Derived from value, potentially outside the range of 0-255 expected for the Color object.
- v: A correctly constrained variable that should be used in calculations but is overlooked.
- Observations: Some responses emphasize that value should be constrained or replaced with v for consistent results. Others suggest that improper casting and insufficient checks for negative inputs could lead to errors. Misuse of variables and lack of validation for input values (lowerBound, upperBound) are central to the error.
- Some responses emphasize that value should be constrained or replaced with v for consistent results.
- Others suggest that improper casting and insufficient checks for negative inputs could lead to errors.
- Misuse of variables and lack of validation for input values (lowerBound, upperBound) are central to the error.
- Code Suggestions: Ensure value is within bounds before using it. Replace value with v where appropriate. Review the lowerBound and upperBound definitions to ensure logical correctness.
- Ensure value is within bounds before using it.

- Replace value with v where appropriate.
- Review the lowerBound and upperBound definitions to ensure logical correctness.

The overarching recommendation is to sanitize inputs and correctly use constrained variables to prevent invalid arguments in the Color constructor.

For the fourth prompt (Professional Summarizer) we adopted a prompt recommended on [Reddit](#) prompting the LLM to act as a professional summarizer and what that entails.

As a professional summarizer, create a concise and comprehensive summary of the provided text, be it an article, post, conversation, or passage, while adhering to these guidelines:

1. Craft a summary that is detailed, thorough, in-depth, and complex, while maintaining clarity and conciseness.

2. Incorporate main ideas and essential information, eliminating extraneous language and focusing on critical aspects.

3. Rely strictly on the provided text, without including external information.

4. Format the summary in paragraph form for easy understanding.

Professional Summarizer: <https://chatgpt.com/share/677ffbf9-0260-8013-a0d5-15fd3ce281dd>

### Final Result:

The provided text discusses a technical issue involving the use of the "g" variable in a coding context, where it is scaled within a range of 255 and cast to an integer, potentially causing problems in the Color class. Specifically, an IllegalArgumentException is triggered when values exceed defined bounds. The failure to properly sanitize the this.lowerBound and this.upperBound variables allows negative values, compounding the issue. Additionally, incorrect usage of "value" instead of "v" on line 117 introduces further complications. The text underscores the importance of using methods like Math.max() and Math.min() with consistent double-type inputs to ensure reliability. Broader challenges are identified with parameter handling in the Color(float, float, float) constructor, with dependencies further obscured by encapsulating issues in a RuntimeException.

The fifth prompt was just the Task copied over.

generate a single explanation by merging the participants' explanations in a way that minimizes redundant information, while keeping the information that would be necessary for someone else to fix the bug.

Types of information that, if present in the explanation, should be preserved  
- how the program works, how the failure is happening, what is problem in the code, etc.

TaskCopied: <https://chatgpt.com/share/677ffc24-ea94-8013-bb09-90a9d9b89d12>

## Final Result:

### Explanation of the Bug

The issue arises from how the variable `g` is calculated and used in the code, specifically in the context of the `Color` constructor, which throws an `IllegalArgumentException` when the parameters are outside the valid range. Here's a breakdown of the problem:

- **How the Program Works:**The program computes `g` using a formula involving `value`, `lowerBound`, and `upperBound`. The computed `g` is cast to an integer and passed to the `Color` constructor, which expects values in the range of 0 to 255.`value` is a parameter passed to the method and is checked against `lowerBound` and `upperBound` to ensure it lies within the expected range.
- The program computes `g` using a formula involving `value`, `lowerBound`, and `upperBound`. The computed `g` is cast to an integer and passed to the `Color` constructor, which expects values in the range of 0 to 255.
- `value` is a parameter passed to the method and is checked against `lowerBound` and `upperBound` to ensure it lies within the expected range.
- **How the Failure Happens:**If `value` is negative or not properly sanitized against `lowerBound` and `upperBound`, the formula for `g` may produce a negative result.This negative `g` is outside the acceptable range for the `Color` constructor, leading to an `IllegalArgumentException`.
- If `value` is negative or not properly sanitized against `lowerBound` and `upperBound`, the formula for `g` may produce a negative result.
- This negative `g` is outside the acceptable range for the `Color` constructor, leading to an `IllegalArgumentException`.
- **Problem in the Code:**While the variable `v` is defined as the sanitized version of `value` (clamped between `lowerBound` and `upperBound`), it is never used in the calculation of `g`. Instead, `value` is directly used, bypassing the intended validation.The issue lies specifically in the line where `g` is computed:`javag = (int) ((value - lowerBound) / (upperBound - lowerBound) * 255);` This should use `v` instead of `value`:`javag = (int) ((v - lowerBound) / (upperBound - lowerBound) * 255);`
- While the variable `v` is defined as the sanitized version of `value` (clamped between `lowerBound` and `upperBound`), it is never used in the calculation of `g`. Instead, `value` is directly used, bypassing the intended validation.
- The issue lies specifically in the line where `g` is computed:`javag = (int) ((value - lowerBound) / (upperBound - lowerBound) * 255);` This should use `v` instead of `value`:`javag = (int) ((v - lowerBound) / (upperBound - lowerBound) * 255);`
- Additional [Notes:](#)The `Color` constructor being used expects three float values in the range of 0.0 to 1.0 or three integer values in the range of 0 to 255. Misalignment in the



parameter range causes the exception. If lowerBound and upperBound are not properly defined, they could also contribute to invalid calculations for g.

- The Color constructor being used expects three float values in the range of 0.0 to 1.0 or three integer values in the range of 0 to 255. Misalignment in the parameter range causes the exception.
- If lowerBound and upperBound are not properly defined, they could also contribute to invalid calculations for g.

## Proposed Fix

Replace the use of value in the calculation of g with the validated v. Ensure that v is clamped within the range [lowerBound, upperBound].

Result:

	Chain of Density [Adams23]	Our prompt	Summarize	Professional Summarizer	Task copied
Word Count	63	348	299	121	511
TTR	0.838	0.277	0.377	0.717	0.227
Readability	-	o	o	++	+
Speed*	-	o	o	++	o
Suggestion	No	Yes	Yes	No	Yes

## Comparing synthetic and original explanations

When comparing synthetic and original explanations, the BLEU scores were rather low.

Issue	BLEU Score
HIT01_8	0.15

HIT02_24	0.00
HIT03_6	0.00
HIT04_7	0.11
HIT05_35	0.00
HIT06_51	0.00
HIT07_33	0.00
HIT08_54	0.00

This is to be expected in this case due to the following reasons:

The BLEU Score is designed for precise matching. It is typically used when comparing sequences that are lexically and syntactically similar by analysing the n-gram overlap between the reference texts and the summary. Explanations by different users are likely to use diverse vocabulary and phrasing. A summary typically condenses and paraphrases information, further reducing lexical overlap and resulting in a low BLEU score.

Furthermore, the provided user explanations show significant variation in structure and word choice. For example, some explanations are concise ("Minutes are set to -15; which is less than 0 and it throws illegal arg exception"), while others are detailed and include reasoning or suggestions for fixing the code.

BLEU also penalizes shorter summaries because they result in fewer n-gram matches. If the summary is much shorter than the average user explanation, this penalty will lower the score.

Finally, BLEU doesn't account for semantic similarity.

Issue	ROUGE Score
HIT01_8	0.13
HIT02_24	0.11
HIT03_6	0.09
HIT04_7	0.07

HIT05_35	0.08
HIT06_51	0.06
HIT07_33	0.11
HIT08_54	0.09

The ROUGE scores were similarly low, likely mostly because of the same reasons. The low values suggest that the summary doesn't closely match the user explanations. However, this doesn't necessarily mean the summary is poor or that the evaluation is invalid. Instead, the low scores likely result from specific characteristics of the data and task, such as the high diversity among user explanations, the condensation in summaries that lowers n-gram overlap, the lack of account of semantic overlap, no weighted importance consideration, and length disparities. For the full ROUGE score results please refer to the Jupyter Notebook.

## Reflection:

*What are the concerns about:*

### *1. guaranteeing the quality of the data*

User Input can be very diverse in quality. It should thus be cleaned, stemmed and tokenized to improve the model performance. There might specifically be issues with crowd sourced data – like the data at hand – due to participants providing the correct answer but only gibberish as an explanation to save time.

### *2. keeping the classifier up-to-date in the case of changes in the demographic of programmers or types of bugs*

Outdated models may fail to perform well with new scenarios. Regularly updating the training data by incorporating recent real-world examples of bugs and programmer feedback may help with this. Continuous learning or periodic re-training strategies to adapt the classifier to changing trends could also be employed. Since training the classifier in our case does not seem to take too long, retraining seems to be a promising avenue. The hard part is to detect the point in time when retraining is necessary, i.e., enough concept drift has occurred.

### *3. testing the output of the classifier and the LLM*

There's the risk of errors propagating from the user explanations or the classifier to the LLM, since the LLM doesn't know the exact correct solution. We can only rely on manual human review for critical explanations to validate correctness. Furthermore, monitoring precision, recall, and other relevant classifier performance metrics and finetuning the classifier are

necessary. It might be helpful to curate a golden data set to assess how summaries should look like and to make sure that the output of the classifier is still as intended.

#### *4. estimating the quality of the consolidated explanations*

Automated evaluation metrics such as BLEU and ROUGE could be used for an initial estimate of quality. However, given the poor performance in our scenarios, a different more fitting metric may be preferred. Manual evaluation by domain experts to assess the coherence, relevance, and completeness of the explanations could further help. Most time intensive but probably also most insightful would be observing developers using the explanations in practical settings.

#### *5. debugging the integration between the classifier and the LLM*

Our “integration” was manual copying and pasting, so there was not anything to debug. For automated systems integration issues, such as misaligned outputs or data mismatches between the classifier and LLM, can lead to failures or degraded system performance. Clear data exchange pipelines should be put in place and edge cases should be tested during integration to avoid potential issues.