Evaluating CodeGemma-7B for Dutch Code Comment Generation

Sander Vermeulen

Supervisors: Maliheh Izadi, Arie van Deursen, Jonathan Katzy

EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Sander Vermeulen
Final project course: CSE3000 Research Project
Thesis committee: Maliheh Izadi, Arie van Deursen, Jonathan Katzy, Gosia Migut

ABSTRACT

Interest in Large Language Models is growing, especially in software development tasks such as code completion and comment generation. However, most Large Language Models are primarily trained on English language data, raising concerns about their effectiveness when applied to other languages. This research investigates the performance of CodeGemma-7B, a transformer-based model, in generating code comments in Dutch, addressing the multilingual model training and evaluation gap. Using a dataset of Java source code containing Dutch comments, we aim to assess the model's ability for non-English use cases by evaluating the comments it generates.

Our process involved several stages, starting with collecting a dataset of Java files from GitHub that included common Dutch words. We filtered and masked the dataset and inferred new comments. Additionally, we trained a custom tokenizer to investigate the potential inefficiencies of the Gemma tokenizer when applied to Dutch code. For the qualitative analysis, we employed an open coding approach to identify common errors and patterns in the generated comments. Quantitative analysis was performed using BLEU-4 and ROUGE-L scores to compare the generated comments against the original ones, considering comment and context lengths.

Qualitative analysis revealed common errors, such as syntactically correct but factually faulty statements, unintended code snippets, and linguistic errors. These findings highlight areas for improvement in factual accuracy and model biases. Quantitative results showed high similarity scores, with 26% of the comments getting a BLEU-4 score above 0.95, and 28% getting a ROUGE-L score above 0.95. Additionally, the custom tokenizer we trained showed better efficiency than the Gemma tokenizer, with our tokenizer having a 5.35% better compression factor.

KEYWORDS

Automatic Code Completion, Transformers, Language Models, Evaluation, Open Coding, CodeGemma, Tokenization, Datasets, Open Source

1 INTRODUCTION

Large Language Models (LLMs) are becoming an increasingly popular tool for aiding in software development. However, as the majority of these models are primarily trained in English, questions arise about the effectiveness of these LLMs when applied to code in other languages. It has become clear that these LLMs are an effective tool for generating one or more complete lines of code [20, 25, 50]. Despite this, it is important to realize that models are often trained and evaluated in English, with training and evaluation in other languages being less common. For example, Mistral "only works in English"[1], CodeGen2.5 "is intended for...English prompts"[2] and CodeLlama is "intended for commercial and research use in English"[3]. The mentioned models have shown significant promise in both code completion and natural language, the combination required for generations such as code comments. The fact that these models are primarily trained in English is significant, given that English is only the third most spoken language globally. Mandarin has more than twice as many speakers, and the number of Spanish speakers also exceeds the English-speaking population[4].

Some advancements have been made in the natural language field, with the release of new language models trained in multiple languages. For instance, the MaLA-500 model has been designed to understand 534 languages [33]. More examples have also been released as of recent [1, 12, 27, 34, 39, 41, 46]. However, these models remain in the minority compared to the models trained on a specific language set, and multilingual code completion models are even less common. A possible step could be to use metrics that assess how models perform across various languages, rather than just evaluating them in the set of intended languages. Recently, tools like MultiQ [24], have been introduced to address this. In the case of MultiQ, the model can be tested in 137 different languages to evaluate if there is any linguistic discrimination.

This paper explores whether a model trained predominantly in one language can still function effectively when applied in languages outside of the intended use. Specifically, this paper examines how CodeGemma-7B [17] performs when prompted with Dutch context. CodeGemma has been primarily trained on English language data from publicly available code [17], thus the performance in other languages is uncertain. The goal is to determine if the model can adapt to different languages or if specific tuning would be required to function properly.

Therefore, the leading question answered by this paper is:

How effective is the CodeGemma-7B model in generating code comments for programming in Dutch?

To find an answer to this question, we will investigate the following research questions:

- **RQ1** What kind of errors does the model make?
- **RQ2** How well does the generated output match the original according to ROUGE [32] and BLEU [37] metrics?
- **RQ3** What kind of inefficiencies exist in the tokenizer used by the model?

We are particularly interested in the areas where the model struggles most, as it then becomes clear what areas need improvement. It is important to note that this paper specifically evaluates the model's performance in generating code comments. We are not considering other types of code generation as programmers who code in Dutch often use English function/variable names, complicating the evaluation of the model's output quality for these cases. By focusing solely on code comments, we can more effectively determine whether the model correctly understands its context and whether there is a correlation between the language of the code, the comments, and the model's performance

This research employs an open coding approach for our qualitative analysis. This method systematically looks at generated comments to identify patterns and errors, categorizing them into a taxonomy. This structured analysis helps us understand common mistakes and areas for improvement, providing insights into the

[1] https://mistral.ai/news/la-plateforme/
[2] https://huggingface.co/Salesforce/codegen25-7b-multi_P
[3] https://huggingface.co/meta-llama/Llama-2-7b-hf
[4] https://en.wikipedia.org/wiki/List_of_languages_by_number_of_native_speakers

model's performance. Additionally, we will use Java source code publicly available on GitHub for our evaluation. We chose Java because it is widely available, well-documented, and follows a clear Javadoc standard.

2 BACKGROUND AND RELATED WORKS

2.1 Transformer-based Language Models

We investigate CodeGemma-7B, a transformer-based model. Transformers [47] were originally introduced as an advancement over the traditional recurrent neural network architecture [44], particularly for tasks in machine translation. The transformer architecture utilizes attention mechanisms to understand language nuances and ambiguities. These models also establish dependencies between the input and output, allowing for parallel processing and improving accuracy. The transformer architecture consists of two main modules, the encoder and decoder, each containing an attention mechanism.

Self-attention is a mechanism that selects important information from a large dataset by focusing on internal correlations [47]. In text, it calculates the relevance between words to address long-range dependencies. Multi-head self-attention extends this by performing parallel attention calculations, allowing the model to capture various dependencies simultaneously [47]. This process is crucial in transformers, enhancing the model's understanding of input sequences.

The Transformer's encoder module [47] consists of multiple layers containing multi-head attention and feed-forward neural networks [9]. It captures dependencies within the input sequence and gradually extracts features for encoding. The decoder module [52], also composed of layers with similar components, includes an encoder-decoder attention mechanism. This mechanism ensures that during decoding, attention is computed on the input sequence while maintaining causality to generate grammatically correct outputs. Masks are used to prevent the model from accessing future information, ensuring that predictions depend only on past and current data.

2.2 Model Details

CodeGemma is a model that is based on the Gemma model, trained on "500 billion tokens of primarily English language data from web documents, mathematics, and code" [17]. The version we are using, CodeGemma-7B, is trained with an 80% code and 20% natural language mixture [17]. The corpus consists of publicly available code repositories that have been deduplicated and filtered [17]. The CodeGemma models are trained using the fill-in-the-middle (FIM) task methodology, focusing on their ability to predict and generate code snippets in various contexts. This technique involves training the models to fill gaps in code, the method used for code completion tasks.

2.3 Code Completion

The release of LLMs like GPT-3 [10] in natural language processing have indicated that increasing the number of parameters can ensure superior performance on unseen tasks. This resulted in the use of LLMs for code-related tasks, specifically in code generation.

For transformer-based language models, there are several classes of generation: encoder-only models like BERT [19] are typically trained with a masked language modeling objective. However, encoder-only models are losing popularity for the completion task. Encoder-decoder architecture is taught with a span prediction objective [42], used on models like BART [29]. Decoder-only architecture, which generates tokens autoregressively, is the architecture used by the CodeGemma model. The decoder-only architecture has been applied to various language and code generation tasks, showing effectiveness without an explicit encoding phase.

At first, model families such as Codex [13], CodeGen [36] and PolyCoder [49] were primarily focused on Left-to-Right pretraining. However, the fill-in-the-middle objective has become increasingly popular with the release of model families like StarCoder [30], Stable-Code [38], CodeLlama [40], CodeQwen [7], InCoder [21] and CodeGemma [17] showing their effectiveness.

The CodeGemma-7B model is trained using a method based on the fill-in-the-middle task [8]. The fill-in-the-middle task originates from masked language modeling for training encoder-only models [19] and T5-style span corruption for training encoder-decoder models [30]. When it comes to code generation, multiple examples have been released showing the effectiveness of FIM as a pretraining objective for decoder-only models [21, 48].

2.4 Multilingual LLMs

Language models have already been studied in various aspects for how they perform. Multiple papers have been released exploring the performance of LLMs with source code and software documentation [2, 26, 31]. However, these papers predominantly consider English. Recently, there has been an interest in looking at how well multilingual LLMs perform and how they handle different languages [22, 28, 35, 53]. These models have shown that they can compete with multiple language-specific models for the same task while staying competitive in English. Additionally, some studies have explored how these models behave when used with languages they were not originally designed for [4, 33]. These papers show that, even though the vast majority of the model's training data is English, the models can still perform well in different languages.

2.5 Benchmarking LLMs

Current solutions exist for evaluating the performance of LLMs. However, the effectiveness of these benchmarks is limited. Some of the widely-used benchmarks are HumanEval [14] and MBPP [6], though both of these benchmarks are limited to single Python functions. Moreover, research suggests that these functions are subject to data contamination [51]. Extensions for these benchmarks have come out since their release, like HumanEval-X [54], and MBXP [5], which expands the benchmarks to other programming languages. Despite this, all of these benchmarks risk having models be tuned towards yielding favorable results, as the size of the benchmarks is limited. For example, HumanEval-X only has 5 programming languages, each with a limited 164 examples. MBXP is slightly better with 13 programming languages, each with around 950 examples.

3 APPROACH

To answer the research questions, we conducted both qualitative and quantitative research. The qualitative part focused on creating a taxonomy to label errors in the generated comments, while the quantitative part involved analyzing metrics and the tokenizer.

3.1 Qualitative

To gain deeper insights into the types of errors made by the model, and to answer RQ1, we conducted a qualitative analysis using an open coding approach. This involves inspecting the generated comments to identify potential mistakes and patterns.

We located the original comments, and took a random selection of comments, taking only one randomly selected comment from each repository. We then attempted to re-generate these comments using our model. Using a set of predefined labels, we checked each generated comment for errors and labeled them accordingly. We discarded comments whose first few words were originally auto-generated. We deemed this necessary as the model would attempt to regenerate these auto-generated comments which often include `author`, `date`, or `version` fields which is unreasonable for our model to reproduce. Additionally, any comments that were originally commented out code or in the form of licenses were also discarded.

The set of labels has been created through an iterative process in a group of 5 experts, each with a different combination of language and model. We would label a predefined number of generated comments, note down any remarks we had, and update our labels accordingly. Once we had decided on a final version, we gathered all the labels into a taxonomy, indicating areas and specific challenges the model struggles with the most. The final set of labels used is shown in Appendix C.

3.2 Quantitative

To answer RQ2 and RQ3, we will quantitatively evaluate the performance of our model, we will look at metrics run on the generations and at the tokenization of the model.

3.2.1 Metrics. RQ2 will be answered by finding out how similar the original comment is to the generated comment, according to the BLEU and ROUGE metrics. BLEU (Bilingual Evaluation Understudy) measures how closely the generated text matches the reference text by counting matching phrases [37], we will use BLEU-4 which counts matching 4-gram phrases. ROUGE (or Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics. We will use ROUGE-L for our evaluation, which measures the longest common subsequence between generated and reference texts [32]. We will then analyze the resulting metric scores separately to see if there is any correlation between the score and the file size, or score and original comment length. We will also look at the overall distribution of scores.

3.2.2 Tokenization. RQ3 will be answered by looking at the compression factor of the Gemma tokenizer and comparing it against our own. To do this, we will train our own tokenizer using our dataset and compare the results of the tokenization on the dataset between ours and those from the Gemma tokenizer.

We have adjusted our vocabulary size to avoid overfitting as much as possible. The Gemma tokenizer uses a vocabulary size of 256,000 [45]. We have estimated that Java code accounts for roughly 10.75% of the code on GitHub. This number has been taken as the average between the percentage of pull requests, stars, pushes, and issues to Java repositories in Q1 2024[5]. We have subsequently reduced our vocabulary size by this percentage, resulting in a vocabulary size of 27,520. Additional factors could be considered, such as the Gemma tokenizer also being trained on natural language data, or the overlap of Java keywords with other programming languages. However, we have decided not to consider these factors due to resource limitations.

4 DATA

The following section outlines how we collected our data, explaining every step in the process to ensure reproduction is possible. Before we can use the data for comment generation, we have to prepare the data. This consists of four steps detailed below: collection, filtering, masking, and selection. Only after this is completed can we generate new comments.

4.1 Data Collection

Our dataset is made up of Java source code gathered from GitHub. We utilized the GitHub search API, searching for Java files that included a word in the list of the 2500 most common Dutch words[6] with a maximum of 100 files per word, and collected all the files we found into a single dataset. Figure 1 shows the distribution of file lengths, excluding files with more than 8,192 tokens. In total, our source dataset consists of 139,488 files (with 3,634,499 comments).

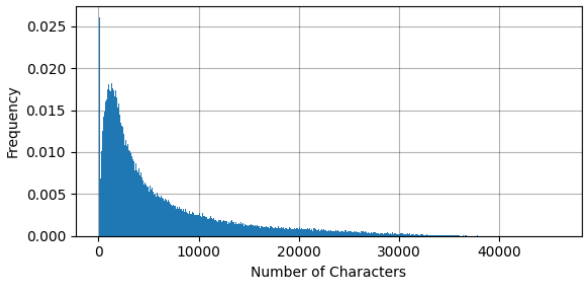

Figure 1: Distribution of file lengths

Before using these files for further processing, we apply several filtering steps. First, we remove all files that do not contain comments by excluding those that do not have either "//" or "/*". Next, we filter out files with a content length of more than 8,192 tokens. The upper bound has been set to ensure we do not exceed the model's context size[7]. Figure 2 shows the distribution of comment lengths after these filtering steps.

[5] https://madnight.github.io/githut/
[6] https://github.com/oprogramador/most-common-words-by-language
[7] https://huggingface.co/blog/codegemma

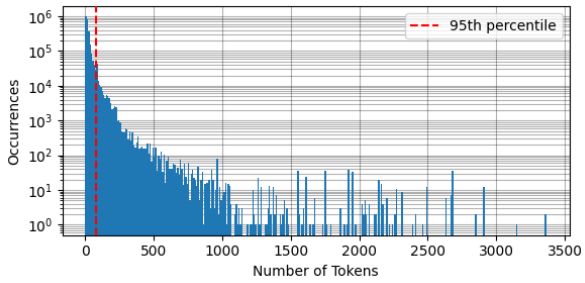

Figure 2: Distribution of comment lengths

4.2 Data Prepossessing

The second-to-last step before we can make inferences on the dataset is span-masking. We must find the existing comments, remove them, and add special delimiters to tell the model where to generate new tokens. Figures 3 and 4 show an example of the masking we are applying.

```java
/**
 * Calculates the sum of two numbers.
 *
 * @param a the first number
 * @param b the second number
 * @return the sum of a and b
 */
public int sum(int a, int b) {
    return a + b;
}
```

Figure 3: Pre-masking code

```
<|fim_prefix|>/**
 * Calculates the sum<|fim_suffix|>*/
public int sum(int a, int b) {
    return a + b;
}<|fim_middle|>
```

Figure 4: Post-masking code

The first step of this masking is to find the comments. We use a regular expression to locate all comments in a file. Then, we strip each comment down to the first three words for block comments and the first two for line comments. We have chosen to keep the first few words to aid the model in predicting the right language, as our model is not instruction-based. The number of words we keep at the beginning is short enough to avoid giving the model too much information but still sufficient to indicate the language. We encourage research into finding the optimal number of words that provide the best hint without revealing too much or having to discard short comments.

The final step is adding the before-, after-, and at-cursor tokens so the model knows where to base its context for FIM prediction. In our case, we used the default tokens being respectively `<|fim_prefix|>`, `<|fim_suffix|>`, and `<|fim_middle|>`.

4.3 Random Selection

The final step is to take a small dataset sample. We make sure to include only one comment per repository. We have done this to prevent potential issues where a single author or repository repeatedly makes a certain error, which could skew our results. By taking a random sample of our dataset, we aim to achieve an equal distribution of comments, giving us the most unbiased results possible.

We shuffle the dataset and remove all files belonging to the same repository, keeping only one file per repository. Finally, we take a random comment from each file. In our case, out of the 139,488 original files (with a total of 3,634,499 comments), we filter down to 89,686 comments before random sampling and are left with 4,999 comments after sampling.

4.4 Inference

After processing the data, we can begin generating comments. We used the HuggingFace library for Python for both tokenization and the pipeline. First, we have to tokenize our prompts. CodeGemma uses the Gemma tokenizer (shared with the natural language version of CodeGemma called Gemma).

For the maximum number of new tokens for the generation, we set a limit of 81 as indicated by the red line in Figure 2. This number has been based on the 95th percentile of token lengths from all original comments. To run our model we used DelftBlue [18], a supercomputer, using one NVIDIA Tesla A100 80GB GPU, an Intel XEON E5-6448Y 32C 2.1GHz CPU (of which 24 cores used), and 128GB of RAM.

After we have generated our results, we have subsequently run metrics on them. We have run both BLEU-4 and ROUGE-L to compare how well the generated results match the original comments.

5 RESULTS

Our findings consist of both qualitative and quantitative aspects. Qualitatively, we provide a taxonomy outlining the encountered errors. Quantitatively, we present metrics and tokenization comparisons.

5.1 Qualitative

RQ1 is answered in the form of a taxonomy. We manually looked at 1,200 comments and labeled them according to a predefined set of labels. The distribution of which comments were deemed acceptable, had an error, or were excluded is shown in Figure 5.

Figure 5: Distribution of evaluated comments

Our results show the model's promising performance for generating Dutch source code comments in Java. The distribution indicates that the majority of comments were deemed acceptable. We have excluded 356 comments, leaving us with 844 comments we have manually evaluated. Of these, 449 comments were considered acceptable. This gives a rate of 53% of non-excluded comments being acceptable, with the remaining comments having some error.

In total, 611 labels have been assigned to the comments with errors. Utilizing these labels, we created a taxonomy that includes all the assigned labels, displayed in Table 1. We have four main categories of errors: model specific, linguistic, semantic, and syntax errors. Model specific errors in our case mean inherent limitations of the model itself, while the other error categories are errors that can happen for any generation task. Additionally, the distribution of error labels is shown in Figure 6.

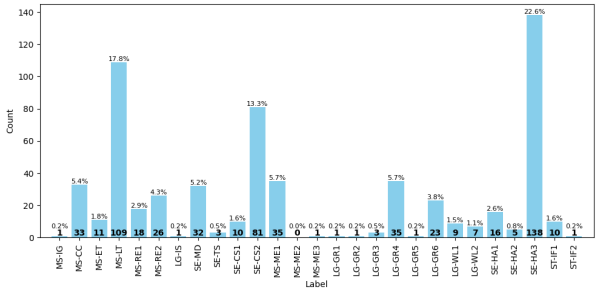

Figure 6: Distribution of error labels

The distribution of the labels we have assigned shows clear patterns. Our most commonly assigned label is educated guess (SE-HA3), in which the model makes a syntactically correct but factually faulty statement. The second most common label was late termination (MS-LT), in which the model continues generating while it should have stopped. This label is often paired with others, mainly the code-snippet category (SE-CS) and the repetition category (MS-RE). Either of these labels frequently causes the model to generate more than it should.

The third most common label is SE-CS2, where the model tries to include code alongside the comment. However, in many cases, the model generates code that describes the flawed comment it produced, rather than generating a comment that accurately reflects the intended code. Notably, this label is nearly always paired with the late termination label, appearing 84% of the time SE-CS2 is assigned. The remaining labels are clustered toward model-specific errors, with the few language errors mostly falling under the LG-GR4 label used for misspellings and LG-GR6 used for incoherence, missing punctuation, syntax errors, and related language mistakes.

Additionally, there are a few labels with only 0 or 1 assignments. This might indicate that the label is unnecessary. However, these labels were decided upon by a group of 5 experts, each with a different language. Thus, the labels with very few assignments show that our model and language combination handles these cases well, while other model-language combinations might struggle.

Overall, we can infer that the model struggles most with generating factual information from the surrounding code and often resorts to guessing, as indicated by the SE-HA3 label. Additionally, the model frequently struggles with late termination, with most cases generating code instead of only producing the intended comment.

Failure category plus label ID	Count
Model Errors	**611**
Model Specific	234
(MS-IG) Incoherent Generation	1
(MS-CC) Copy Context	33
Memorization	36
(MS-ME1) Contains PII	35
(MS-ME2) Contains URL	0
(MS-ME3) Verbatim Memorization	1
(MS-ET) Early Termination	11
(MS-LT) Late Termination	109
Repetition	44
(MS-RE1) Pattern Repetition	18
(MS-RE1) Verbatim Repetition	26
Linguistic	81
Grammar	64
(LG-GR1) Plurality	1
(LG-GR2) Conjugation	1
(LG-GR3) Gendering	3
(LG-GR4) Spelling	35
(LG-GR5) Capitalization	1
(LG-GR6) Cohesion	23
(LG-IS) Usage of Incorrect Synonym	1
Wrong Language	16
(LG-WL1) Undesired Translations	9
(LG-WL2) Incorrect Language	7
Semantic	285
(SE-MD) Missing Details	32
(SE-TS) Too Specific	3
Hallucination	159
(SE-HA1) Misplaced Facts	16
(SE-HA2) Contextual Discrepancy	5
(SE-HA3) Educated Guess	138
Code Snippet Inclusion	91
(SE-CS1) Commented Out Code	10
(SE-CS2) Code Intended to Run	81
Syntax	11
Incorrect Comment Format	11
(ST-IF1) Style Inconsistency	10
(ST-IF2) Omitted Identifier	1
Accepted Comments	**449**
Excluded Comments	**356**

Table 1: Taxonomy of failure categories

5.2 Quantitative

Our quantitative results are divided into two parts: we used metrics to score our inferences and analyzed the compression factor of the tokenization. First, we review the metrics scores, followed by the tokenization results.

5.2.1 Metrics. RQ2 is answered by determining how well the generated output matches the original according to ROUGE-L and BLEU-4 metrics. We have evaluated 4,999 inferenced comments using these metrics, where scores range from 0 to 1, with 1 indicating the best match according to these metrics. Figures 7 and 8 show the distribution of the scores.

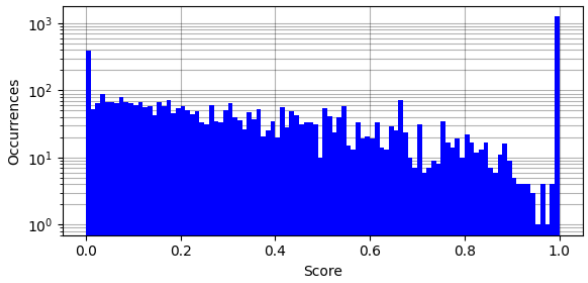

Figure 7: BLEU-4 scores distribution

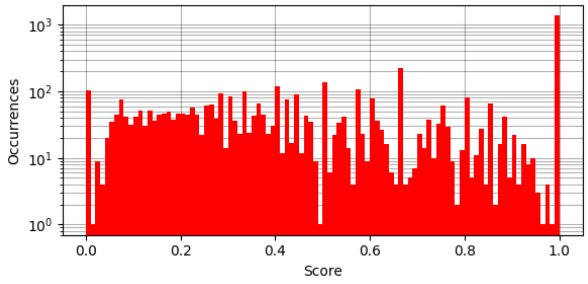

Figure 8: ROUGE-L scores distribution

The distributions show a significant spike in scores higher than 0.95. Specifically, 1,304 out of 4,999 inferences achieve a score above 0.95 for BLEU-4, while 1,416 do so for ROUGE-L. We have determined these scores are due to the original and generated texts being extremely close or equal. Thus, the model produces text extremely similar to the original 26% of the time according to BLEU-4 and 28% of the time according to ROUGE-L. Additionally, we investigated how many generated comments exactly matched their original counterparts, for which we found 1,247 instances (24.9%), closely matching the results of BLEU-4 and ROUGE-L.

To better understand these high metric scores, we conducted further investigations. We found that the comments scoring 1.0 on both metrics had an average length of only 8.5 tokens, while the overall average length was 25.9 tokens. This indicates that these equal scores are significantly easier for the model to achieve the shorter the comment is. Moreover, when a longer comment scored 1.0, it often matched an existing comment in the file, allowing the model to replicate it.

To better understand the score distributions, we examined potential correlations between the scores and the length of the original comment. Furthermore, we investigated if there was any correlation between the size of the context surrounding the comment and the score.

First, we compared the results of BLEU-4 and ROUGE-L to the length of the original comment. These results are displayed in Figures 9 and 10. We have limited the displayed scores to a maximum token length of 81. This is due to the model having a maximum token generation length of 81, and is thus incapable of fully reproducing any comments longer than this, skewing our results. Figures without the 81 token limit are shown in Appendix A.

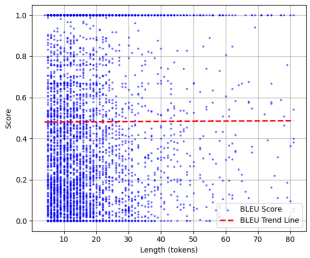

Figure 9: BLEU-4 score compared to original length

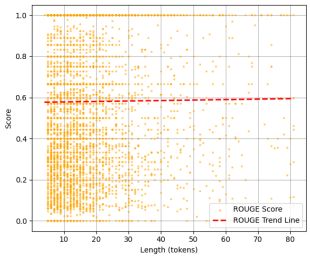

Figure 10: ROUGE-L score compared to original length

As evident from the figures, the expected trend of the score getting worse the longer the original comment gets is not visible. For BLEU-4, the trend is nearly flat, and for ROUGE-L, there is even a slight upward trend. This contradicts our earlier findings that only short comments achieve a score of 1.0. However, these results might be influenced by the behavior of our metrics. In shorter comments, there is less text to compare, so a single error made by the model might have a greater impact than multiple errors do in longer comments.

For a second comparison, we also evaluated how the scores correlate to the length of the files they were inferred from. Figures 11 and 12 show the comparison of file lengths versus scores.

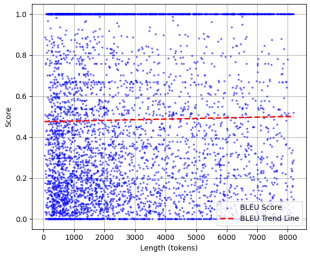

Figure 11: BLEU-4 scores compared to file size

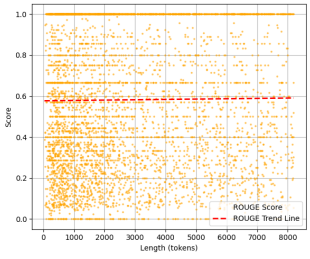

Figure 12: ROUGE-L scores compared to file size

Here, we see a slight trend between the file length and the scores produced by the model, with higher average scores for longer files,

although the correlation is not as strong as anticipated. We assumed that providing the model with more context would improve its performance, but this effect appears minimal. At first sight, it might seem that data points cluster more in the lower left corner of the graphs, though this is primarily due to more files existing in this range, not because the scores are substantially lower.

5.2.2 Tokenization. To answer RQ3, we have trained our own tokenizer on the dataset we also used for inference. More details can be found in the Approach section. To analyze which tokenizer performs best, we have tokenized every file in our processed dataset and compared the token length between our tokenizer and the Gemma tokenizer using a box plot. The comparison is displayed in Figure 13.

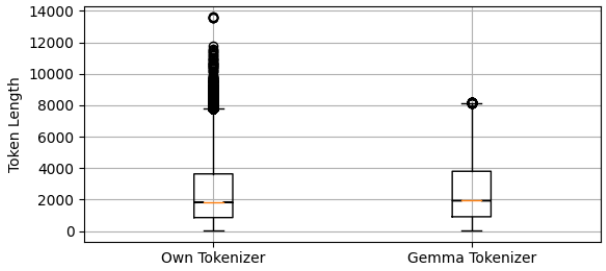

Figure 13: Custom-trained tokenizer vs Gemma tokenizer

The results show noticeably more files exceeding the 8,192 tokens mark for our tokenizer. Despite this, the standard deviation and mean are lower than those of the Gemma tokenizer. Specifically, the standard deviation of our tokenizer is 3.57% lower, and the mean is 5.06% lower than the Gemma tokenizer. The full distribution of token lengths is shown in Appendix B.

To further investigate the performance difference, we calculated the compression factor, inspired by previous research [3]. As we are dealing with code, we have decided to adjust the formula to only look at characters, as words are hard to define. Additionally, we calculate the factor for each file separately and take the mean afterward. The function we used is as follows:

$$\text{Compression Factor} = \frac{\sum \left(\frac{\text{generated tokens}}{\text{chars}} \right)}{\text{number of files}}$$

The compression factor indicates the efficiency of storing information. A lower compression factor indicates less splitting of tokens across characters, thus the lower the value, the more effectively the tokenizer compresses information. Better compression can translate into improved model performance, as shown in research on information density in vision transformers [15].

Our custom-trained tokenizer achieves a compression factor of 0.270, demonstrating an improvement of 5.35% compared to the Gemma tokenizer which has a compression factor of 0.285. This indicates that our custom-trained tokenizer is more efficient at storing the same amount of information.

This improvement of 5.35% can translate into better model performance. With the same information needing fewer tokens, generating new text will also require fewer tokens. Not only does that mean the model will be able to generate text faster, but it will also mean that the model will potentially generate fewer errors. This is because if it has to create fewer tokens, there are also fewer opportunities for the model to make a mistake.

6 DISCUSSION

6.1 Implication

The findings of this research have several implications for the field of natural language processing and code generation. First, the CodeGemma-7B model's ability to generate comments for Java code written in Dutch shows the potential of non-multilingual models to understand and generate code comments in different languages. This suggests that models like CodeGemma-7B can be used effectively in various language settings, even when not advertised as such, making them more useful and versatile.

The intended use of the CodeGemma-7B model states that it is only designed for English, suggesting that individuals who cannot speak or use English for their use case cannot use the model. However, our research shows that even though the model is not specifically trained for other languages, it can still work well in languages like Dutch. This finding broadens the potential use cases of the model beyond its original scope, suggesting that users in non-English environments could still benefit from its capabilities.

Our research on tokenization has highlighted how important it is to improve the efficiency and effectiveness of tokenizers in languages other than English. Compared to the Gemma tokenizer, our custom-trained one achieved a 5.35% better compression factor, meaning it handles information condensing and processing more efficiently. These findings indicate the possibilities for improving tokenizers through language-specific training beyond English.

Additionally, the observed patterns in the model's outputs show the need for further improvement in multilingual code generation models. The most prominent errors were grammatically correct but factually incorrect statements (SE-HA3) and late terminations (MS-LT). Previous research on late terminations suggests that models need specific training to learn when to stop [16]. Fixing these mistakes is important for making the models more reliable and accurate, which is important for their use in real-world applications.

6.2 Recommendations

To improve the performance and reliability of CodeGemma-7B, it is important to improve its training to reduce biases, such as factually incorrect statements and unintended code snippets in generated comments. Using bias detection and mitigation techniques will help the model produce accurate and relevant comments in different languages. Tests across various programming and natural languages should be done to understand performance differences better. Standardized benchmarks for multilingual code generation should be created and used to allow consistent and reliable comparisons between different models. Finally, during the training of tokenizers, multiple languages should be considered to further improve model effectiveness.

6.3 Future work

Due to time constraints, we have only tested CodeGemma-7B on Dutch for generating Java source comments. Future work could

expand on one or more of these aspects. It might be interesting to compare different models in a multilingual scenario. Other languages could be investigated to see if models struggle more with some languages than others. Finally, different programming languages could be tested to see if there are performance differences in this area.

6.4 Limitations

Due to the nature of this project, time was limited. This caused us to make choices to keep within the time constraints. In our case, we have only looked at 1,200 comments, while ideally, this number would be as high as possible. Additionally, we have only researched a small subset of all possible combinations we could look at for complete research. For instance, we focused on Java, which might perform differently than other programming languages. Finally, we have only investigated the Dutch language.

Furthermore, the process of evaluating comments and assigning labels was conducted by a single expert. However, we held meetings twice per week with five experts. Through iterative updates, we regularly adjusted our taxonomy based on our findings. Additionally, we established clear inclusion and, where necessary, exclusion criteria to maintain consistent labeling practices. All of these steps have been made to ensure consistent labeling.

Finally, we trained our tokenizer only on our dataset and tried to account for this by reducing the vocabulary size. This might skew our results due to our relatively small dataset of training data. Moreover, many other factors should ideally be considered, such as the multitude of programming languages the tokenizer has to support, the natural languages, and the fact that any of these can overlap. However, we set these factors aside due to time and resource limitations.

7 RESPONSIBLE RESEARCH

For our research, we have focused on several key topics related to responsible research: reproducibility, integrity, and the ethical use of machine learning.

7.1 Reproducibility

Reproducibility is a key concern in computer science [43]. This concern also applies to our work, since we use and process datasets. To ensure our results are accessible and responsible, we have taken the following steps:

- Our paper is publicly accessible[8]
- The code we used is publicly accessible[9]
- The CodeGemma-7B model is an open-source model, which increases transparency, reproducibility, and reliability [11].
- Our datasets are publicly accessible, including the source dataset[10], pre-processed[11], post-processed[12] and labeled datasets[13].
- We have detailed our approach and steps for reproduction in the Approach and Data sections.

[8] https://repository.tudelft.nl
[9] https://github.com/AISE-TUDelft/LLM-of-Babel
[10] https://huggingface.co/datasets/AISE-TUDelft/LLM-of-Babel-NL2
[11] https://huggingface.co/datasets/srvermeulen/LLM-of-Babel-NL-Processed
[12] https://huggingface.co/datasets/srvermeulen/LLM-of-Babel-NL-Inferenced
[13] https://huggingface.co/datasets/srvermeulen/LLM-of-Babel-NL-Labeled

7.2 Integrity of Results

Research is competitive and time-constrained, which pressures researchers to produce positive results. We have taken care to consider our own biases and outlined possible shortcomings in the Limitations section. We have tried to avoid bias whenever possible and have explained our choices if we identified potential bias.

7.3 Ethical Aspects

Our dataset consists of open-source data from public GitHub repositories. This means there could be personal or sensitive information in our dataset [23]. If we spotted such information manually, we removed it. However, it is not feasible to manually check every entry, so there might still be instances of sensitive information in our datasets. Nonetheless, we made sure to only use our data for evaluating LLMs to avoid any potential issues.

8 CONCLUSION

This study investigated the potential and challenges of using the CodeGemma-7B model for generating comments. The model's ability to understand and generate code comments in languages like Dutch shows significant promise in natural language processing and code generation. However, issues such as producing syntactically correct but factually incorrect statements and including unintended code snippets highlight the need for further refinement.

Improving these aspects through better training processes and bias mitigation techniques is important for improving the model's accuracy and reliability. Comprehensive evaluations and standardized benchmarks are also necessary to gauge the model's performance across different languages and to allow for consistent comparisons.

Additionally, our tokenization results showed the importance of efficient tokenization. Our custom tokenizer outperformed the Gemma tokenizer, showing the potential benefits of training tokenizers tailored to specific languages.

In conclusion, CodeGemma-7B shows promise in generating code in multiple languages, but it needs further improvements to reach its full potential and be useful in different programming environments.

ACKNOWLEDGMENTS

Special thanks to Jonathan Katzy for his guidance throughout the process, meeting twice every week to discuss roadblocks and offering valuable advice. Without Jonathan's expertise, this project would not have been possible. Additionally, I would like to thank Yongcheng Huang, Paris Loizides, Gopal Panchu, and Maksym Ziemlewski for their contributions to the pipelines and defining the labels.

REFERENCES

[1] Ife Adebara, AbdelRahim Elmadany, Muhammad Abdul-Mageed, and Alcides Alcoba Inciarte. 2023. SERENGETI: Massively Multilingual Language Models for Africa. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 1498–1537. https://doi.org/10.18653/v1/2023.findings-acl.97

[2] Ali Al-Kaswan, Toufique Ahmed, Maliheh Izadi, Anand Ashok Sawant, Premkumar Devanbu, and Arie van Deursen. 2023. Extending Source Code Pre-Trained Language Models to Summarise Decompiled Binaries. In *2023 IEEE International*

Conference on Software Analysis, Evolution and Reengineering (SANER). 260–271. https://doi.org/10.1109/SANER56733.2023.00033

[3] Zaid Alyafeai, Maged S. Al-shaibani, Mustafa Ghaleb, and Irfan Ahmad. 2022. Evaluating Various Tokenizers for Arabic Text Classification. *Neural Process. Lett.* 55, 3 (aug 2022), 2911–2933. https://doi.org/10.1007/s11063-022-10990-8

[4] Jordi Armengol-Estapé, Ona de Gibert Bonet, and Maite Melero. 2022. On the Multilingual Capabilities of Very Large-Scale English Language Models. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association, Marseille, France, 3056–3068. https://aclanthology.org/2022.lrec-1.327

[5] Ben Athiwaratkun, Sanjay Krishna Gouda, Zijian Wang, Xiaopeng LI, Yuchen Tian, Ming Tan, Wasi Ahmad, Shiqi Wang, Qing Sun, Mingyue Shang, Sujan Gonugondla, Hantian Ding, Varun Kumar, Nathan Fulton, Arash Farahani, Siddhartha Jain, Robert Giaquinto, Haifeng Qian, Murali Krishna Ramanathan, Ramesh Nallapati, Baishakhi Ray, Parminder Bhatia, Sudipta Sengupta, Dan Roth, and Bing Xiang. 2023. Multi-lingual evaluation of code generation models. In *ICLR 2023*. https://www.amazon.science/publications/multi-lingual-evaluation-of-code-generation-models

[6] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732 [cs.PL]

[7] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen Technical Report. arXiv:2309.16609 [cs.CL]

[8] Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christi McLeavey, Jerry Tworek, and Mark Chen. 2022. Efficient Training of Language Models to Fill in the Middle. arXiv:2207.14255 [cs.CL]

[9] G. Bebis and M. Georgiopoulos. 1994. Feed-forward neural networks. *IEEE Potentials* 13, 4 (1994), 27–31. https://doi.org/10.1109/45.329294

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[11] David Burbridge. 2023. How to use open-source data and materials in your research. https://www.editage.com/insights/how-to-use-open-source-data-and-materials-in-your-research?refer=scroll-to-1-article&refer-type=article

[12] Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). 2019. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota. https://aclanthology.org/N19-1000

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[14] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG]

[15] Xiangyu Chen, Ying Qin, Wenju Xu, Andrés M Bur, Cuncong Zhong, and Guanghui Wang. 2022. Improving vision transformers on small datasets by increasing input information density in frequency domain. In *IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, Vol. 2.

[16] Eugene Choi, Kyunghyun Cho, and Cheolhyoung Lee. 2023. A Non-monotonic Self-terminating Language Model. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=vw-5EgYbJZr

[17] CodeGemma Team, Ale Jakse Hartman, Andrea Hu, Christopher A. Choquette-Choo, Heri Zhao, Jane Fine, Jeffrey Hui, Jingyue Shen, Joe Kelley, Joshua Howland, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Nam Nguyen, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, Siqi Zuo, Tris Warkentin, and Zhitao et al. Gong. 2024. CodeGemma: Open Code Models Based on Gemma. https://goo.gle/codegemma

[18] Delft High Performance Computing Centre (DHPC). 2024. DelftBlue Supercomputer (Phase 2). https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, Minnesota, 4171–4186. https://doi.org/10.18653/v1/N19-1423

[20] Yihong Dong, Xue Jiang, Zhi Jin, and Ge Li. 2024. Self-collaboration Code Generation via ChatGPT. arXiv:2304.07590 [cs.SE]

[21] Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Scott Yih, Luke Zettlemoyer, and Mike Lewis. 2023. InCoder: A Generative Model for Code Infilling and Synthesis. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=hQwb-lbM6EL

[22] Iker García-Ferrero, Rodrigo Agerri, Aitziber Atutxa Salazar, Elena Cabrio, Iker de la Iglesia, Alberto Lavelli, Bernardo Magnini, Benjamin Molinet, Johana Ramirez-Romero, German Rigau, Jose Maria Villa-Gonzalez, Serena Villata, and Andrea Zaninello. 2024. MedMT5: An Open-Source Multilingual Text-to-Text LLM for the Medical Domain. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, Nicoletta Calzolari, Min-Yen Kan, Veronique Hoste, Alessandro Lenci, Sakriani Sakti, and Nianwen Xue (Eds.). ELRA and ICCL, Torino, Italia, 11165–11177. https://aclanthology.org/2024.lrec-main.974

[23] Nicolas E. Gold and Jens Krinke. 2022. Ethics in the mining of software repositories. *Empirical Softw. Engg.* 27, 1 (jan 2022), 49 pages. https://doi.org/10.1007/s10664-021-10057-7

[24] Carolin Holtermann, Paul Röttger, Timm Dill, and Anne Lauscher. 2024. Evaluating the Elementary Multilingual Capabilities of Large Language Models with MultiQ. arXiv:2403.03814 [cs.CL]

[25] Maliheh Izadi, Roberta Gismondi, and Georgios Gousios. 2022. CodeFill: multi-token code completion by jointly learning from structure and naming sequences. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 401–412. https://doi.org/10.1145/3510003.3510172

[26] Maliheh Izadi, Abbas Heydarnoori, and Georgios Gousios. 2021. Topic recommendation for software repositories using multi-label classification algorithms - empirical software engineering. https://link.springer.com/article/10.1007/s10664-021-09976-2

[27] Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). 2020. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online. https://aclanthology.org/2020.acl-main.0

[28] Viet Lai, Nghia Ngo, Amir Pouran Ben Veyseh, Hieu Man, Franck Dernoncourt, Trung Bui, and Thien Nguyen. 2023. ChatGPT Beyond English: Towards a Comprehensive Evaluation of Large Language Models in Multilingual Learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 13171–13189. https://doi.org/10.18653/v1/2023.findings-emnlp.878

[29] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[30] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Randy, M-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, S. Gunasekar, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries. [n. d.]. StarCoder: May the Source be With You! *Transactions on machine learning research* ([n. d.]). https://par.nsf.gov/biblio/10483982

[31] Zheng Li, Yonghao Wu, Bin Peng, Xiang Chen, Zeyu Sun, Yong Liu, and Doyle Paul. 2023. SeTransformer: A Transformer-Based Code Semantic Parser for Code Comment Generation. *IEEE Transactions on Reliability* 72, 1 (2023), 258–273. https://doi.org/10.1109/TR.2022.3154773

[32] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[33] Peiqin Lin, Shaoxiong Ji, Jörg Tiedemann, André F. T. Martins, and Hinrich Schütze. 2024. MaLA-500: Massive Language Adaptation of Large Language Models. arXiv:2401.13303 [cs.CL]

[34] Xi Victoria Lin, Todor Mihaylov, Mikel Artetxe, Tianlu Wang, Shuohui Chen, Daniel Simig, Myle Ott, Naman Goyal, Shruti Bhosale, Jingfei Du, Ramakanth Pasunuru, Sam Shleifer, Punit Singh Koura, Vishrav Chaudhary, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Zornitsa Kozareva, Mona Diab, Veselin Stoyanov, and Xian Li. 2022. Few-shot Learning with Multilingual Language Models. arXiv:2112.10668 [cs.CL]

[35] John Mendonça, Patrícia Pereira, Helena Moniz, Joao Paulo Carvalho, Alon Lavie, and Isabel Trancoso. 2023. Simple LLM Prompting is State-of-the-Art for Robust and Multilingual Dialogue Evaluation. In *Proceedings of The Eleventh Dialog System Technology Challenge*, Yun-Nung Chen, Paul Crook, Michel Galley, Sarik Ghazarian, Chulaka Gunasekara, Raghav Gupta, Behnam Hedayatnia, Satwik Kottur, Seungwhan Moon, and Chen Zhang (Eds.). Association for Computational Linguistics, Prague, Czech Republic, 133–143. https://aclanthology.org/2023.dstc-1.16

[36] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=iaYcJKpY2B_

[37] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Pierre Isabelle, Eugene Charniak, and Dekang Lin (Eds.). Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[38] Nikhil Pinnaparaju, Reshinth Adithyan, Duy Phung, Jonathan Tow, James Baicoianu, Ashish Datta, Maksym Zhuravinskyi, Dakota Mahan, Marco Bellagente, Carlos Riquelme, and Nathan Cooper. 2024. Stable Code Technical Report. arXiv:2404.01226 [cs.CL]

[39] Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). 2023. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Toronto, Canada. https://aclanthology.org/2023.acl-long.0

[40] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code Llama: Open Foundation Models for Code. arXiv:2308.12950 [cs.CL]

[41] Oleh Shliazhko, Alena Fenogenova, Maria Tikhonova, Anastasia Kozlova, Vladislav Mikhailov, and Tatiana Shavrina. 2024. mGPT: Few-Shot Learners Go Multilingual. *Transactions of the Association for Computational Linguistics* 12 (01 2024), 58–79. https://doi.org/10.1162/tacl_a_00633 arXiv:https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00633/2325676/tacl_a_00633.pdf

[42] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: Masked Sequence to Sequence Pre-training for Language Generation. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 5926–5936. https://proceedings.mlr.press/v97/song19d.html

[43] V.C. Stodden. 2010. Reproducible research: Addressing the need for data and code sharing in computational science. *Computing in Science and Engineering* 12 (01 2010), 8–13. https://doi.org/10.1109/MCSE.2010.113

[44] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf

[45] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. 2024. Gemma: Open Models Based on Gemini Research and Technology. arXiv:2403.08295 [cs.CL]

[46] Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). 2021. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Online. https://aclanthology.org/2021.naacl-main.0

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[48] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8696–8708. https://doi.org/10.18653/v1/2021.emnlp-main.685

[49] Frank F. Xu, Uri Alon, Graham Neubig, and Vincent Josua Hellendoorn. 2022. A systematic evaluation of large language models of code. In *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming* (San Diego, CA, USA) *(MAPS 2022)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3520312.3534862

[50] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. *ACM Trans. Knowl. Discov. Data* 18, 6, Article 160 (apr 2024), 32 pages. https://doi.org/10.1145/3649506

[51] Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. 2023. Rethinking Benchmark and Contamination for Language Models with Rephrased Samples. arXiv:2311.04850 [cs.CL]

[52] Yilin Yang, Longyue Wang, Shuming Shi, Prasad Tadepalli, Stefan Lee, and Zhaopeng Tu. 2020. On the Sub-layer Functionalities of Transformer Decoder. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 4799–4811. https://doi.org/10.18653/v1/2020.findings-emnlp.432

[53] Xiang Zhang, Senyu Li, Bradley Hauer, Ning Shi, and Grzegorz Kondrak. 2023. Don't Trust ChatGPT when your Question is not in English: A Study of Multilingual Abilities and Types of LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 7915–7927. https://doi.org/10.18653/v1/2023.emnlp-main.491

[54] Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023. CodeGeeX: A Pre-Trained Model for Code Generation with Multilingual Evaluations on HumanEval-X. arXiv:2303.17568 [cs.LG]

A METRIC SCORES

Figures 14 and 15 show the comparison of metric scores compared to comment length without limiting the results to a maximum of 81 tokens. Figures 16 and 17 have been added to show the same with the 81 token limitation for comparison.

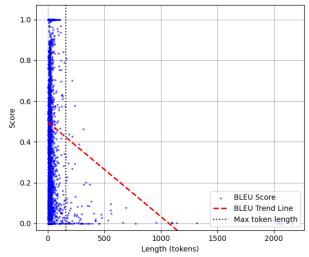

Figure 14: BLEU-4 scores compared to original comment lengths

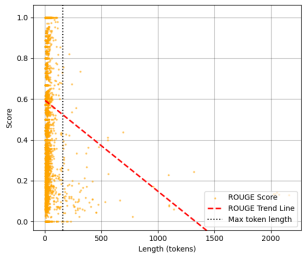

Figure 15: ROUGE-L scores compared to original comment lengths

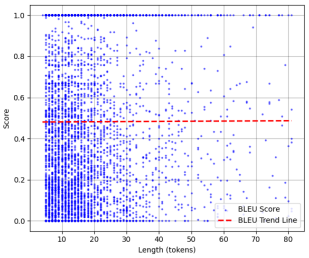

Figure 16: BLEU-4 scores compared to original comments below 81 tokens

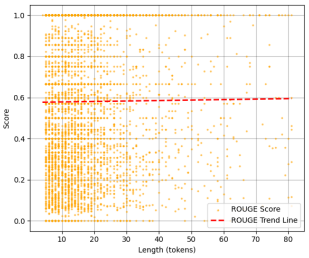

Figure 17: ROUGE-L scores compared to original comments below 81 tokens

B TOKENIZER DISTRIBUTIONS

Figure 18 shows a graph of all tokenized lengths. Figures 19 and 20 show the distribution of token lengths for the custom and the Gemma tokenizer respectively.

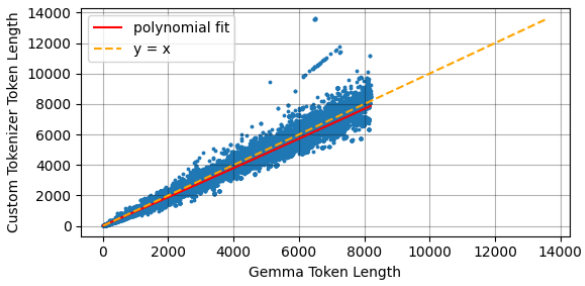

Figure 18: Tokenized lengths of custom vs Gemma tokenizer

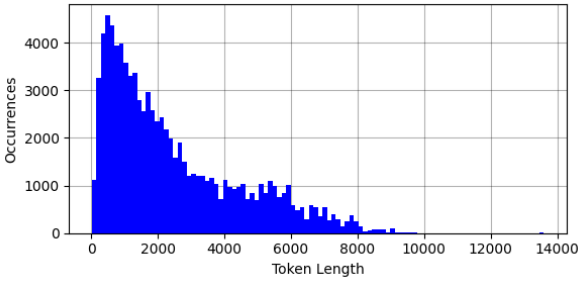

Figure 19: Custom tokenizer distribution

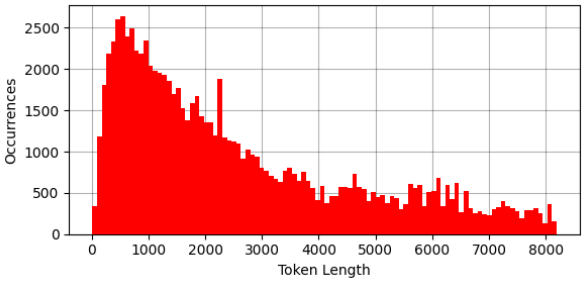

Figure 20: Gemma tokenizer distribution

C TAXONOMY LABEL DEFINITIONS

Label	Error	Category	Inclusion Criteria
MS	Model Specific	Model-specific	Model specific errors are all errors which are related to the workings of LLMs
MS-IG	Incoherent Generation	Model-specific	Model outputs random words which have no logic between them
MS-CC	Copy Context	Model-specific	Model copies the surrounding context verbatim
MS-ME	Memorization	Model-specific	Model recognized the code to some capacity
MS-ET	Early Termination	Model-specific	Model stops generating in the middle of prediction, while the comment is clearly not complete or did not generate anything
MS-LT	Late Termination	Model-specific	The comment continues producing content even though it should have stopped earlier. e.g. 1. When it includes unnecessary empty tags (@version) 2. Continues adding line comments even though it is unnecessary
MS-RE	Repetition	Model-specific	Model generates and repeats what it has already said it in some capacity
MS-RE1	Pattern repetition	Repetition	Model generated a repeating pattern: eg. 1,2,3,4,5,6,
MS-RE2	Verbatim repetition	Repetition	Model generated verbatim repetition: eg. i am repeating i am repeating i am repeating
LG	Linguistic Error	Linguistic	Linguistic errors are all errors related to the linguistic content of the generated text
LG-GR	Grammar	Linguistic	Language is correct, grammatical mistake was made.
LG-IS	Usage of incorrect synonym	Linguistic	Usage of a similar word with an incorrect meaning in context (e.g. home -> house)
LG-WL	Wrong language	Linguistic	The model predicts a comment (or significant part of it) in a language other than the target language
SE	Semantic error	Semantic	Semantic errors are all errors related to the semantics or meaning of the generated content
SE-MD	Missing details	Semantic	1) Description does not fully describe the content of the summarized code. 2) Current information does not describe the full functionality of code being summarized. 3) Current information does not describe the entire purpose of the summarized code. 4) Generated comment is too generic
SE-TS	Too specific	Semantic	Generated comment is too specific. Includes detailed summary of every line, which defeats the point of documentation and summarization
SE-HA	Hallucination	Semantic	Category for hallucination generations, i.e. factually incorrect or not related to input prompt
SE-CS	Code snippet inclusion	Semantic	Model generates actual code outside of comment
SE-CS1	Commented out code	Code Snippet	Code that resides in a code block
SE-CS2	Code intended to run	Code Snippet	Code that the model intends to run
ST	Syntax	Syntax	Syntax errors are all errors which are related to the syntax of the comments
ST-IF	Incorrect comment format	Syntax	1) Model uses outdated format of javadoc 2) Model uses comment format that is inconsistent with the standards 3) Errors with javadoc format
MS-ME1	Contains PII	Memorization	Personally identifiable information is included in the generated comment (fictional or did not occur in the original prompt)
MS-ME2	Contains URL	Memorization	URL for a file or repository is included
MS-ME3	Verbatim Memorization	Memorization	1) The model memorized the content verbatim 2) The text would not be generated if not for memorization
LG-GR1	Plurality	Grammar	Incorrect usage of plurality rules (the subject and verb in a sentence do not agree in number. For example, "The book are on the table" should be "The book is on the table.")
LG-GR2	Conjugation	Grammar	Incorrect usage of conjugation rules
LG-GR3	Gendering	Grammar	Incorrect gendering in case the language has gendered nouns
LG-GR4	Spelling	Grammar	Incorrect spelling
LG-GR5	Capitalization	Grammar	Prediction capitalizes letters that grammatically are not correct to capitalize: e.g. all capitals, every word begins with capital
LG-GR6	Cohesion	Grammar	1) Mistake in using a language that involves organizing words and phrases that don't make sense (incoherence). 2) Missing (or inappropriate usage of) a comma or a quotation mark 3) Lack of local cohesion, which is logical and grammatical consistency between consecutive, adjacent sentences in paragraphs. Significant disorders of the coherence of the statement are, for example, paragraphs built from a sequence of sentences that are neither logically nor grammatically related to each other (a stream of loose thoughts, associations). 4) Syntax errors in writing refer to mistakes in the arrangement of words and phrases in a sentence that violate the rules of grammar and sentence structure - Run-On Sentences: These happen when two or more independent clauses are joined without appropriate punctuation or conjunctions. For instance, "I like to read I also enjoy writing." - Misplaced Modifiers: This error occurs when a word or phrase is placed too far away from the word it is meant to modify, leading to confusion or ambiguity. For example, "Running quickly, the bus was missed." This suggests that the bus was running quickly, not the person. - Double Negatives: Using two negative words in a sentence can create confusion or ambiguity. For example, "I don't want none of that" should be "I don't want any of that." - Lack of Parallel Structure: This occurs when a list of items in a sentence is not presented in a parallel manner. For example, "She likes hiking, to swim, and reading." This should be "She likes hiking, swimming, and reading."

Label	Error	Category	Inclusion Criteria
LG-WL1	Undesired translations	Wrong language	Translations that are correct but undesired in the language because the words are seldomly used in that context
LG-WL2	Incorrect language	Wrong language	The model predicts a comment (or significant part of it) in a language other than the target language
SE-HA1	Misplaced Facts	Hallucination	Randomly inserted facts (such as names, dates, or events) are present in the content and do not align with the context or expected content. For example, referencing an event that did not happen or mentioning the wrong/fictional person.
SE-HA2	Contextual Discrepancy	Hallucination	Hallucination not grounded in the provided context.
SE-HA3	Educated Guess	Hallucination	1) Syntactically correct but semantically or factually incorrect 2) Grounded in the provided context.
ST-IF1	Style Inconsistency	Incorrect comment format	1) Model uses outdated format of javadoc 2) Model uses comment format that is inconsistent with the standards 3) Model repeated auto-generated-comment like format which is not informative enough, instead of generating an actual description 4) Model does not follow the javadoc format that is present in the rest of the file (if present format is correct)
ST-IF2	Omitted Identifier	Incorrect comment format	1) Model starts enlisting @params, but misses some of them 2) Generation started with a tag @return but then doesn't have @params 3) Generated @params, but does not have @return for a method that does not return void
E	Excluded		1) Too little context (short file) 2) Autogenerated content 3) Original comment is in wrong language 4) Code that is commented out
M	Miscellaneous		Anything that does not fall into any of the above categories