LLM of Babel: Evaluation of LLMs on code for non-English use-cases

Maksym Ziemlewski
EEMCS, Delft University of Technology, The Netherlands

Supervisor(s): Prof. Dr. Arie van Deursen, Assistant Prof. Dr. Maliheh Izadi, ir. Jonathan Katzy

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: Maksym Ziemlewski
Final project course: CSE3000 Research Project
Thesis committee: Prof. Dr. Arie van Deursen, Assistant Prof. Dr. Maliheh Izadi, Assistant Prof. Dr. Gosia Migut

LLM of Babel: Evaluation of LLMs on code for non-English use-cases

Maksym Ziemlewski
Delft University of Technology
Delft, Netherlands

ABSTRACT

This research evaluates the performance of Meta's Code Llama 7B model in generating comments for Java code written in Polish. Using a mixed-methods approach, we conduct both quantitative and qualitative methods to discover the model's accuracy and limitations. We preprocess a dataset of Polish Java code from GitHub, apply a Fill-in-the-Middle objective for code comment completion, and evaluate the results using BLEU and ROUGE-L metrics. Additionally, we manually evaluate approximately 1150 generated comments and document the encountered errors. Based on the findings, we iteratively develop a taxonomy of errors using an open coding approach.

Through an expert evaluation, we discover the limitation of the BLEU metric in assessing comment quality for non-English languages, showing substantial differences with human evaluation. Our research identifies the most frequent errors in code comment completion in Polish, which are the generation of code snippets, copying context, late termination, hallucinations and repetitions. Only 25.2% of the generated comments were classified to be correct. This study is a part of the broader research about multiple models across various non-English languages. We aim to contribute to raise the awareness of large language models for code accessibility in non-English environments, therefore improving their inclusivity.

KEYWORDS

Language Models, Evaluation, non-English, Code Llama, Code Comment Completion, Error Taxonomy, Open Coding, Error Classification

1 INTRODUCTION

In recent years, the integration of Large Language Models (LLMs) into software development has significantly increased, driven by multiple factors, such as boosting the productivity of developers [32], generating tests [23], and improving the quality of the system and software documentation [5, 10]. LLMs, such as OpenAI's ChatGPT[1] models, have shown promise in aiding programmers of varying proficiency levels and are increasingly considered for educational purposes [4].

However, a persistent challenge still stands out: the disparity in performance across multiple natural languages [13]. While significant attention has been directed towards evaluating LLMs for English, research in other languages has been notably limited [12]. The exploration of LLM evaluation and performance metrics outside of English-speaking contexts remains under-explored.

Most of the research has focused on English-based benchmarks [27], which results in biases and sub-optimal performance for non-English contexts [13]. Addressing this gap is crucial. This research investigates the performance of Meta's Code Llama[2], particularly in comment completion when coding in non-English languages, with a focus on Polish. By investigating this model, this study aims to highlight the limitations and errors encountered in generating comments for code in Polish.

This research is part of a collaborative effort to assess LLM capabilities in diverse natural languages for comment generation in Java[3]. The mutual goal is to identify and categorise problems that arise when generating comments in different languages. This paper covers the assessment for Polish, thereby contributing to develop a taxonomy of limitations for multiple natural languages.

The motivation for this research is to enhance LLM accessibility for developers worldwide and propagation of multilingual programming practices. We want to raise the awareness about the accessibility of the models for non-English speakers and non-English use cases. By systematically evaluating LLMs' performance in non-English settings, the study aims to contribute to the enhancement of software development tailored to multiple linguistic needs.

The paper has the following structure: We start with our research questions, followed by related works section. In the methodology section, we detail our research design, including data preparation, data collection, and dataset preprocessing techniques used. Quantitative and qualitative analyses are then described, along with the open coding process.

Next, we present dataset statistics with more insights about the Polish dataset as well as information about the language and its differences with comparison to English. The results section follows. The discussion reflects on the implications and limitations of our findings. Responsible research and ethics regarding datasets are addressed.

Research Questions

The main research question is: **How does Code Llama perform in comment generation in Java when applied in Polish language settings?** Sub-questions to add more structure to this research include:

- **(RQ1)** How often does Code Llama generate erroneous comments for code written in Polish?
- **(RQ2)** What are the most frequent types of limitations encountered in comment generation for code written in Polish?
- **(RQ3)** What is the influence of temperature parameter for code comment completion in Java for the Polish language?

[1] https://en.wikipedia.org/wiki/ChatGPT

[2] https://llama.meta.com/code-llama

[3] www.oracle.com/java

2 RELATED WORKS

Research on the evaluation of Large Language Models (LLMs) across multiple languages has highlighted several key areas.

Evaluations of LLMs for specific languages, not limited to code, reveal poor performance for Llama 2, which Code Llama is based on [26]. Llama was performing the worst in comparison to other models in Indonesian and African languages, probably due to its English and code-centric training corpus [12, 20].

New discoveries in the field highlight a substantial multilingual bias in non-English LLM code generation [28], which was previously underexplored. It was found that Code Llama 34B exhibits a big performance gap between code generation with instructions in English and Chinese, noting a performance decrease of 37.8% in the Pass@1 metric when Chinese instructions were used. The results indicate a significant bias in language models regarding their ability to understand various natural languages when generating code.

General evaluations of LLMs, not specific to code, were also comprehensively reviewed. The main takeaways indicate that LLMs are primarily evaluated on English data, which can lead to poor multilingual performance, particularly for non-Latin languages and resource-limited languages [6]. Additionally, research measuring the performance of LLMs across multiple non-English languages and ranking these languages based on LLM performance on them highlights several discoveries that align with the statement that LLMs perform sub-optimally in non-English languages [13]. The results show a strong correlation between Llama 2's performance in different languages and the proportion of the pre-training corpus. It is worth noting that Llama 2, the base model for Code Llama, has an 89.7% English training corpus [26]. Although not focused on code-specific assessment, this benchmark can be used to measure the performance of LLMs in a given language.

In contrast, evaluations of models for code have primarily focused on English, emphasising English-based benchmarks [1, 8, 30]. Findings from the evaluation of existing largest language models for code suggest that blending natural languages in the training corpus may enhance code language modelling [30]. Regarding Code Llama, research validating the quality of LLM-generated code indicates that it maintains 100% syntax correctness and 64% format correctness when generating documentation from code [1].

Similarly, evaluations of code comment completion, discovering the superiority of transformer architecture over an n-gram model [18], have also been based on English data. Error analysis in LLM-generated code, without a focus on non-English contexts, resulted in obtaining a taxonomy of the ten most frequent bug patterns [25]. Research on code explanation generation reveals that a big part of the automatically generated content is of good quality, however again with an English-centric approach.

Lastly, concepts and research related to code comments were also summarised, discussing automatic generation technology and the evaluation of comment quality, but focused solely on English [31].

This body of work highlights the gap in evaluating LLMs for code comment completion in non-English languages, which this study aims to address.

3 METHODOLOGY

To investigate the performance of Meta's Code Llama in generating comments for code written in Polish, this study employs a mixed-methods approach combining quantitative and qualitative research methods. This methodology ensures a thorough evaluation of the model's capabilities and limitations in a non-English programming context.

3.1 Data Preprocessing

The initial step involves preprocessing the dataset to ensure that it meets the requirements for analysis. Java files containing code written in Polish are collected from GitHub. These files are filtered based on the following criteria:

- **Exclude files that are too long** (exceeding 8192 tokens) to avoid surpassing the context window and giving the model too much irrelevant information that could degrade performance [16]. Although Code Llama was trained on sequences of 16k tokens [22], the 8192 tokens were used as the maximum length for consistency with all models, which have shorter context window than Code Llama.
- **Exclude files that do not contain comments for the code.**

3.2 Comment Masking and Fill-in-the-Middle Technique

To prepare the data for the Fill-in-the-Middle (FIM) task, we identify and mask out the comments in Java files using regular expressions. We retain the first three words of each block comment and first two words of each inline comment to provide the model with contextual information about the language to generate in. Afterwards, we mask the remainder with a special Code Llama token " **<SUF>**". For block comments, we ensure that the model recognises the comment boundaries by including the "*/" delimiter after the special token to signify the end of the comment block. For inline comments, we use "\n" as a delimiter. Subsequently, we add two more tokens, "**<PRE>** " at the beginning of the file and " **<MID>**" at the end of the file. This masking process is crucial for applying the Fill-in-the-Middle with prefix-suffix-middle technique, which allows the model to generate comments based on the surrounding context of the code [3]. This method utilises both left and right contexts, which is crucial for accurately predicting comments in a programming environment. The span masking process is visualised in Figure 1, which depicts an example Java file, and Figure 2 with special tokens highlighted in orange, which shows the same file after span masking has been applied to prepare it for FIM.

3.3 Random Selection

To perform manual labelling of errors and qualitative analysis, we narrowed down the dataset described in subsection 4.1, obtained so through the filtering and mapping outlined in previous subsections 3.1 and 3.2. To avoid biases that might arise from large repositories with many faulty files or misplaced comments, we sampled one random comment per repository. This approach limits the influence of single authors or repositories with numerous poor contributions. It also ensures a representative sample of block and inline comments,

```java
public class SimpleClass {

    /**
     * This method greets the person with the provided name.
     *
     * @param name The name of the person to greet.
     * @return A String containing a personalized greeting message.
     */
    public String sayHello(String name) {
      return "Hello, " + name + "!";
    }
}
```

Figure 1: Example Java File Without Processing

```
<PRE>public class SimpleClass {

    /**
     * This method greets <SUF>
     */
    public String sayHello(String name) {
      return "Hello, " + name + "!";
    }
}<MID>
```

Figure 2: Java File Prepared for FIM

as they were uniformly sampled from all detected comments, preserving the real world distribution of those two types of comments in code. The resulting dataset consists of 959 random comments from different repositories, and is released online[4]. Manual labelling and classification of encountered errors were conducted on this dataset.

3.4 Quantitative Analysis

Post-generation, we evaluate the performance of Code Llama using established quantitative metrics such as BLEU [21] and ROUGE-L [14]. BLEU calculates the precision of n-grams between the generated and original comments, while ROUGE-L is recall oriented and considers sentence-level structure similarity [29], capturing the overall meaning and context. We use a smoothing method 4 for BLEU computation [7]. BLEU-4 is less effective for comments shorter than 4 words. This can misrepresent the model's performance, especially for extremely short inline comments, which might receive lower scores despite being accurate in eyes of an expert [2]. Using these metrics, we aim investigate how the generated comments differ from the original comments, which are treated as ground truth. Based on these results, we will be able to partially answer (**RQ1**) by investigating how many of the generated comments closely resemble the original ones, based on their BLEU and ROUGE scores. To fully answer this question, we will further explore the correctness of the generated comments in the qualitative part of this study, described in Section 3.5.

3.5 Qualitative Analysis and Open Coding in Taxonomy Development

To answer (**RQ1**) and (**RQ2**), and to discover how LLMs fail in non-English settings, this research employs the open coding method in qualitative analysis. Open coding allows us to derive conclusions from qualitative data without any underlying assumptions about it beforehand. This approach enables insights to emerge naturally from the data rather than forcing it into predefined categories. The transparent process of open coding enhances the validity of this research by ensuring that the analysis is grounded in the data itself, therefore reducing researcher bias.

We iteratively create and improve the labels describing the encountered error types in the generated comments from the dataset described in Section 3.3 by manually evaluating them. As we gain more understanding from the data with manual labelling, we revise the categories during multiple rounds of discussion to reflect our findings. By these means, we develop a broad taxonomy of errors encountered in comments generated in multiple non-English languages. Continuously improving categories as new error types appear ensures their relevance to the data. The goal of this taxonomy is to provide a basis for understanding how LLMs fail in non-English comment generation, offering insights that can guide future improvements in multilingual model performance. Besides the error categories, we introduce additional label for good predictions, as well as an "Excluded" category for comments that were excluded from the analysis. As each file is an individual case, we decided to leave the decision about excluding the file from analysis to each of the experts. However, in the guidelines about file exclusion we recommend to exclude detected comments which **(a)** contain commented-out code, **(b)** are URLs, **(c)** only contain @author with the name, **(d)** are TODO comments, **(e)** are in a file with too little content to comment on, and **(f)** are in the file with no Polish context, or are not in Polish themselves. Examples of excluded comments can be seen in Appendix B.

Additionally, the analysis of the frequency of error categories, their distribution, popular patterns found, most interesting instances, and other qualitative aspects will be included in the results to provide a thorough understanding of the frequent errors.

3.6 LLM Temperature Experiment Setup

In LLMs, the temperature is the value that influences the probabilities of the model's output. It is used in activations functions, such as softmax, to normalise the output to a probabilistic distribution [19]. Adjusting the temperature parameter can make the output more deterministic (lower temperature) or more random (higher temperature).

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \quad (1) \qquad \text{softmax}(z_i, T) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \quad (2)$$

1: Basic Softmax Formula **2: Softmax with Temperature**

To answer (**RQ3**), we collected 120 comments which were correctly predicted by the model in the basis evaluation described in Section 3.5. A correct prediction indicates that the model can properly handle the files in the basic settings. Then, we re-run the generation on all 120 for five different temperature values: 0.1, 0.3,

[4]https://huggingface.co/datasets/mziem/predicts-label

0.5, 0.7, and 0.9. Then, we manually analyse the predicted content to measure the influence of the temperature parameter to the quality of generation, and discover potential dependencies.

3.7 Error Tracking and Iterative Refinement

Throughout the project, we iteratively refine the taxonomy in three cycles, incorporating new categories of errors in multiple languages as they emerge. This process accurately reflects the types of errors encountered in non-English programming settings.

To maintain consistency in error classification, we ensure evaluation is conducted with uniform parameters across multiple models and languages within broader research that this paper is a part of. This includes using the same maximum length for generated outputs (set at 95th percentile of all original comment lengths, which is 116 tokens in the case of the Polish dataset), using the same dataset mapping and filtering that is described in Section 3.1, and defining strict inclusion and exclusion criteria for each of the categories. These measures minimise bias during the labelling process. Consistency is crucial for accurately comparing the performance of different models and languages.

4 DATA

This section describes the dataset utilised for analysis, examining its structure and providing insights into its composition and characteristics. This dataset serves as the foundation for subsequent analyses and experiments conducted within this study.

4.1 Dataset

After performing all mapping and filtering outlined in the previous subsections (3.1 and 3.2) the final dataset emerges. That dataset serves as a basis for sampling of individual comments, described in subsection 3.3. It is accessible online[5] and contains 13,556 comments in Polish from 2,216 unique files extracted from the initial dataset. We have gathered a total of 9,643 block comments and 3,913 line comments. An examination of the dataset reveals some insights regarding comment length and their distribution. Average original comment length is 103.5 characters with a median length of 63 characters. That translates to average of 39.4 tokens per comment and median of 24 tokens per comment. A visual representation of the distribution of original comment token lengths is provided in Figure 3.

When performing further analysis on the dataset for the purpose of error labelling of generated comments, we discovered several shortcomings in some instances in the dataset. Comments that are too short and files that are lacking sufficient context for meaningful comments, are excluded from further consideration. Additionally, comments containing only auto-generated content such as licences, or single author tags are also excluded. This ensures the integrity and relevance of the dataset for subsequent analysis and conclusions drawn in the end.

4.2 Language

Polish language has several characteristics that distinguish it from other languages, potentially leading to errors in generation if Polish

[5]https://huggingface.co/datasets/mziem/all_results

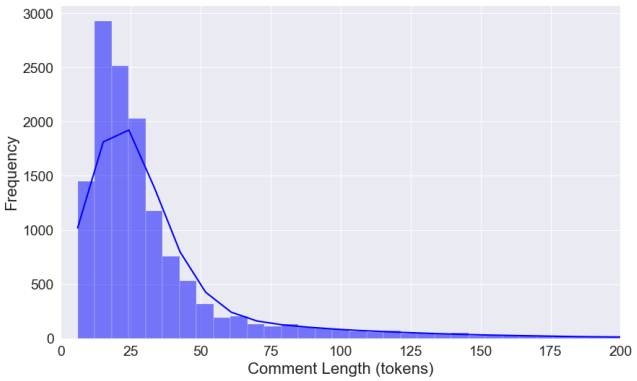

Figure 3: Comment Token Length Distribution

is underrepresented in the training dataset. One of the main differences is the extension of the standard Latin alphabet with nine additional characters (ą, ć, ę, ł, ń, ó, ś, ź, ż). The presence of these additional characters, which can change the meaning and pronunciation of the words, introduces complexity that may challenge the model. Many people, especially in informal communication or any other form that needs to only be practical or understandable, omit those diacritics in writing (e.g., changing "ą" with "a", "ó" with "o"). In this research it was noticed that substantial amount of people also omit them in code comments. This requires the model to learn additional relations between multiple tokens, which effectively encode the same meaning of the word in the training phase. Therefore, the model might encounter difficulties with generating grammatically correct comments and not replicate this "lazy" approach in writing, resulting in erroneous generation.

There are numerous differences in the grammar as well. Polish is characterised by a high degree of inflection, with seven cases (nominative, genitive, dative, accusative, instrumental, locative, and vocative). This contrasts with English, which relies more on word order and prepositions rather than cases. Moreover, Polish verbs have aspects (imperfective and perfective) that indicate whether an action is ongoing or completed, adding another layer of complexity not present in the same way in English, which needs to be captured by the model with already limited training data available in Polish. Additionally, Polish nouns are gendered (masculine, feminine, neuter) and the gender affects adjective and verb conjugation. Although the basic word order in Polish is SVO (subject-verb-object), it is much more flexible due to the inflectional nature of the language, which allows for variations in word arrangement. This flexibility and rich morphology of Polish may lead the model to be more prone to producing grammatical errors compared to languages with a more rigid word order.

Polish constitutes only 0.09% of Llama 2 training data [26], so this sparse occurrence in the dataset might lead to the model having difficulties in this language. Authors underline that Llama 2's proficiency in non-English languages is limited, hence Code Llama which is based on it, can encounter similar complications.

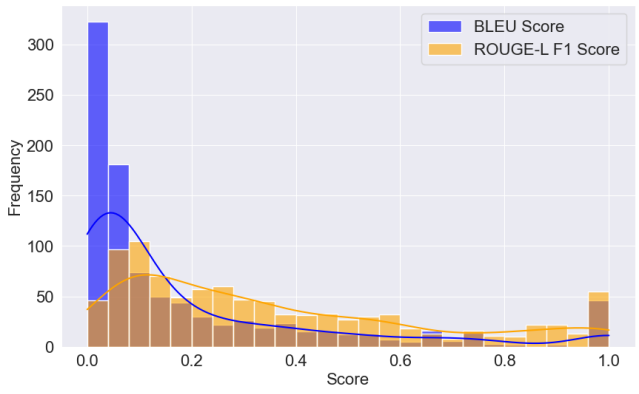

Figure 4: ROUGE-L F1 and BLEU Scores Distribution

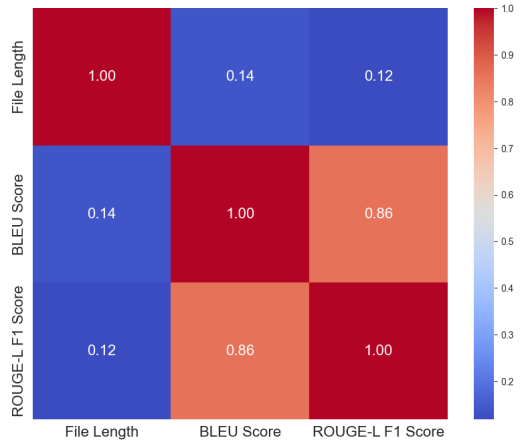

Figure 5: Correlation Heatmap Between Metrics and File Length.

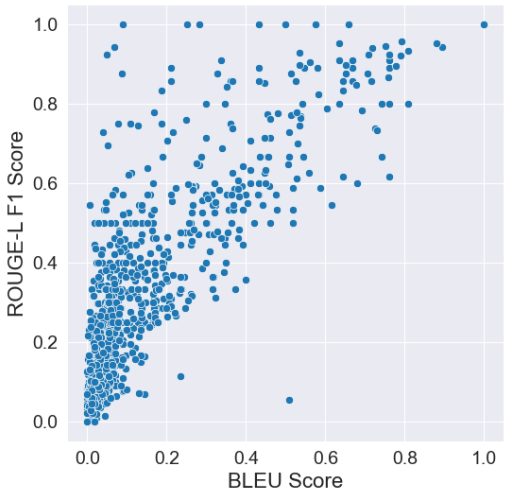

Figure 6: BLEU vs. ROUGE-L F1

5 RESULTS

This section covers the findings from both quantitative and qualitative analysis. All of the results are based on the dataset used for manual labelling, which was obtained through random sampling described in Section 3.3 and is accessible online[4].

5.1 Quantitative

BLEU and ROUGE-L scores, which distributions are depicted on Figure 4, compare the similarity between original comment found in the GitHub sample, and the comment predicted by the model. Scores range from 0 to 1, with values close to 1 indicating large similarity of two texts. Most BLEU scores are close to 0, while ROUGE-L scores are more uniformly distributed, although the majority being in [0, 0.3] range. BLEU measures the exact similarity between reference and candidate texts, which means that if the generated comment uses different sentence structure, it will receive a low score, even though it might still have the correct meaning. In contrast, ROUGE is commonly used for text summarisation, since it captures the important content of the candidate text, leading to higher scores in the distribution. In the tail of the distribution there is a significant break in the trend, with considerably many files have scores close to 1. This is due to relatively short, mostly inline comments which were predicted correctly, word to word with the reference.

Figure 5 shows a strong correlation between BLEU and ROUGE-L scores. This relation is further investigated in Figure 6, where a scatter plot illustrates the shape of it. Together, these visualisations show that BLEU and ROUGE-L metrics are complementary, jointly expressing the quality of the predicted comments. Higher scores in both metrics corresponding to higher-quality predictions and lower scores indicating poor predictions.

What is an interesting finding, is that both scores appear to have negligible correlation with length (in tokens) of the file. BLEU and ROUGE-L scores have 0.14 and 0.12 correlation coefficients with file length measured in tokens respectively, as also can be seen on Figure 5. This indicates, that Code Llama's performance does not deprecate with the length of the context given to the model, up to 8192 which is the upper boundary set in Section 3.1. **The size of the file does not influence the quality of the prediction in Polish negatively**.

5.2 Error Taxonomy

Through iterative collaboration we obtained the error taxonomy, which is visualised on Table 1. More elaborated version, with exact descriptions and inclusion criteria for each of the categories can be found in Appendix A. When manually analysing the predicted comments, the errors were classified to one of the 27 categories positioned as the leaf nodes in the presented taxonomy tree. Only the errors in the leaf nodes have the unique code, which is later used for analysis. The counts of errors in the table are based on the final iteration of labelling, which was done under the presented version of the taxonomy, and consisted of 420 comments.

As a result of manual analysis of the 420 comments from the dataset using an open coding approach, we determined the distribution of errors visualised in Figure 7. Errors that occurred only once were excluded from the plot for better readability. These include verbatim repetition, omission of identifier and plurality grammar

Table 1: Taxonomy of failure categories

Failure category plus label ID	Count
Model-Specific Errors	**229**
(MS-IG) Incoherent Generation	3
(MS-CC) Copy Context	72
Memorisation	12
(MS-ME1) Contains PII	11
(MS-ME2) Contains URL	1
(MS-ME3) Verbatim Memorisation	0
Repetition	63
(MS-RE1) Pattern Repetition	24
(MS-RE2) Verbatim Repetition	39
(MS-ET) Early Termination	8
(MS-LT) Late Termination	71
Linguistic Errors	**76**
Grammar	63
(LG-GR1) Plurality	1
(LG-GR2) Conjugation	11
(LG-GR3) Gendering	5
(LG-GR4) Spelling	12
(LG-GR5) Capitalisation	0
(LG-GR6) Cohesion	34
Wrong Language	6
(LG-WL1) Undesired Translation	3
(LG-WL2) Incorrect Language	3
(LG-IS) Incorrect Synonym	7
Semantic Errors	**218**
Hallucination	91
(SE-HA1) Misplaced Facts	18
(SE-HA2) Contextual Discrepancy	13
(SE-HA3) Educated Guess	60
Code Snippet Inclusion	88
(SE-CS1) Commented out code	14
(SE-CS2) Code Intended to Run	74
(SE-MD) Missing Details	28
(SE-TS) Too Specific	11
Syntax Errors	**11**
Incorrect Comment Format	11
(ST-IF1) Style Inconsistency	10
(ST-IF2) Omitted Identifier	1
Miscellaneous	**5**

errors. Verbatim memorisation and capitalisation, which did not occur at all, were also omitted. Additionally, 67 comments were excluded from the analysis due to bad quality of original comment, and 106 comments were classified as good predictions.

To answer (**RQ1**), we can draw conclusions from the qualitative analysis. **25.2% of predicted comments were manually classified as correct**.

From the distribution of encountered error categories we can derive the answer to (**RQ2**). **The three most frequent errors are**

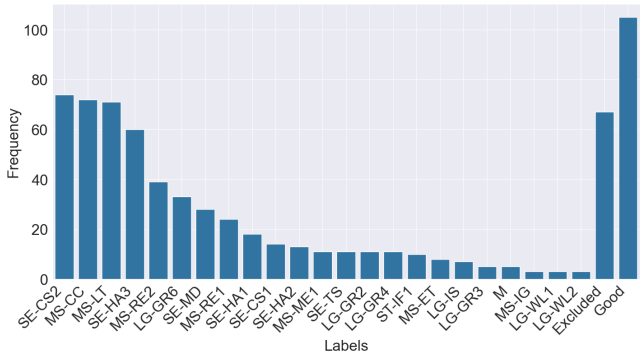

Figure 7: Distribution of Error Categories

generating code snippets, copying the surrounding context verbatim and late termination of prediction, with 74, 72 and 71 instances respectively. Aside from the categories omitted in the plot, Code Llama almost never has issues with errors in "Wrong Language" category, nor does it produce incoherent outputs.

We encountered many linguistic errors, especially in grammar, compared to other languages. Code Llama produced a comment with a grammatical mistake 63 times, hence in 15% of all predictions. More than half of these errors were related to cohesion. In comparison to other languages in our broader study of multiple languages, grammatical errors were generated the most often in Polish. This represents model's issues in handling an highly inflected language like Polish, which has a complex grammatical structure.

An interesting error pattern identified concerns the exclusion of a whitespace character at the end of the remained comment. By not including a space after the last word of the original comment, the model is prone to producing spelling mistakes. The decision not to include a space was made to avoid informing the model that it should proceed with a new word. Instead, we leave that decision up to the model, allowing it to decide to continue the last word. We experienced the issue in 25 instances, so 5.95% of the labelled comments in the final round, where the model continued the last word which always lead to erroneous generation. The issue was addressed before and is related to generating from the middle of a token [11]. This also affects the rest of the generation, often disturbing the cohesiveness of the sentence. The issue is not specific to Polish language, and should be treated as a model shortcoming in training. Examples of such generations can be seen in Appendix B.

5.3 Temperature Influence

To measure the influence of the temperature on the quality of generated comments and answer (**RQ3**), we first evaluated the comments with ROUGE-L F1 score, the same way it is described in Section 3.4. First, we analysed the results quantitatively only with ROUGE-L metric, as BLEU has proven itself to not be additionally informative. Figure 8 shows the distribution of ROUGE-L F1 scores for each temperature. We can observe the best performance for lowest temperature values, 0.1 and 0.3. For higher temperatures, ROUGE-L F1 scores tend to decrease.

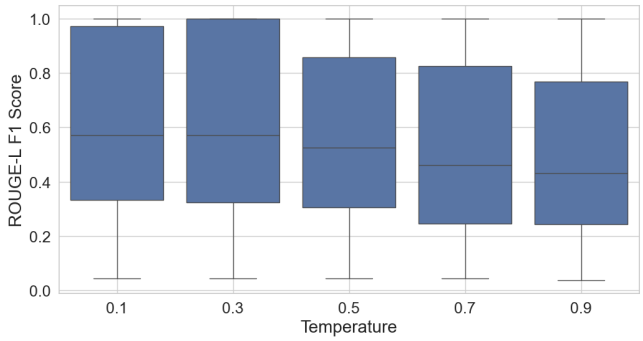

Figure 8: Box Plot of ROUGE-L F1 Scores Across Different Temperatures

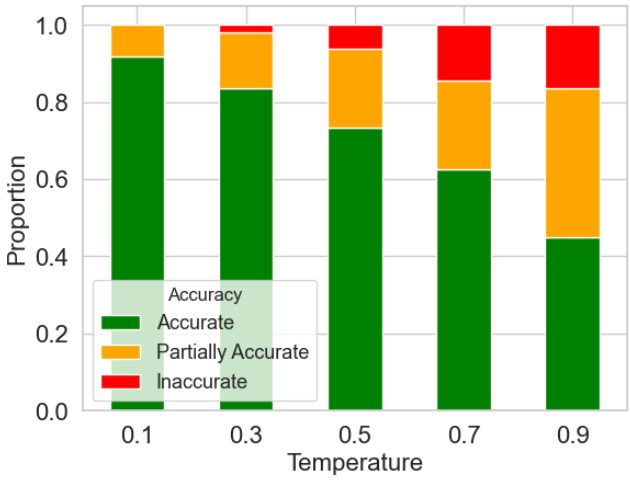

Figure 9: Accuracy Summary by Temperature

We can verify the findings with a qualitative analysis. Manual analysis of 50 comments with five different temperatures was conducted, resulting in 250 samples reviewed. We evaluate each of the comments as "Accurate", "Partially Accurate" or "Inaccurate", to give additional insight about the quality of the comment besides error categories. "Partially Accurate" category represents the comments that were of satisfactory quality, with minor errors that do not critically influence the usability of the comment (i.e. a minor grammatical mistake, containing code snippet that is useful in the context). "Accurate" comments are the ones without any error detected. "Inaccurate" category was assigned to comment with critical errors in them, which cannot be accepted as a useful prediction by a human (i.e. severe hallucination, copying context without purpose). From Figure 9 we can conclude that the tendency revealed with ROUGE-L metrics is in fact correct, as with increasing temperature, less accurate generations occur. **With increasing temperature value in generation, the quality of generated comments in Polish decrease.**

6 DISCUSSION

The results show that while the model demonstrates some proficiency in generating accurate comments, it also faces several challenges inherent to non-English languages, particularly Polish, which we discuss below.

6.1 Implication

One primary observation is that the taxonomy developed through iterative discussions covers various non-English languages, leading to categories that may not be relevant for Polish. For instance, the capitalisation (LG-GR5) category, common in Greek, was absent in the Polish dataset. On the other hand, grammatical mistakes occurred frequently for Polish, while none were generated for Chinese. This highlights the issue of certain errors being specific to individual languages, complicating the task of tailoring the taxonomy to ensure universality and accuracy for all non-English languages. Our taxonomy might not capture the intricacies in various languages, as African or Indonesian languages, indicating the need for a more diverse languages are needed.

Another critical issue is Code Llama's difficulty in determining the appropriate endpoint for comment generation. Instead of concluding with a meaningful comment, the model often generates irrelevant code or repeats itself. This likely stems from the training data, which mostly consists of uncommented code, leading to struggles with producing descriptive comments [24]. Among all code snippet generations (SE-CS2), 54.05% were associated with late termination (MS-LT) errors, and 52.7% were linked to repetition errors (MS-RE1 and MS-RE2). Furthermore, the Code Llama model frequently copies context (MS-CC) and exhibits "educated guessing" (SE-HA3), potentially due to a lack of extensive training data, leading to repeated or nonsense content. This issue compounds with the model's struggles with Polish grammar, particularly in longer and more complex comments. While language models handle English grammar well [15], Code Llama produced multiple grammatical errors, primarily incoherent sentences, followed by spelling mistakes.

When it comes to the quality of generated comments, mentioned issues can be critically detrimental, leading to decreased trust in LLMs and AI systems in general. Perceived performance and usefulness are the dominant factors in the acceptance of new AI technologies by users [9, 17]. As LLMs are still a relatively new and significant breakthrough in the AI field, it is crucial for model creators to build and maintain user trust. **The model's inability to produce meaningful comments, while hallucinating or copying context, severely degrades perceived usability and trust in such technology.**

Authors of Llama's and Code Llama's technical reports are not entirely transparent about the origins of the data used for training, making it unclear if the issue stems from it. However, Polish files encountered on GitHub during our evaluation often lacked descriptive comments or used a very simplified language which is not syntactically correct (e.g., parts of sentences in short forms, avoiding Polish diacritics like "ą," "ł," etc.). Code Llama's struggles in generating contextually appropriate comments suggest the lack of diverse and well-annotated training data. The lack of descriptive and technically correct Polish comments on platforms like

GitHub intensifies this problem and reveals a broader issue in AI and LLMs: the quality of the training data significantly impacts the model's performance, especially in less commonly used languages. If the training data contains flawed or overly simplified examples, the model will inevitably reproduce these errors. This underscores the importance of using high-quality and diverse datasets to train models. Improving data quality and transparency in AI training processes is crucial for developing more reliable LLMs.

One of the main findings in this research is the unsatisfactory efficacy of BLEU. Our expert evaluation has proven that 25.2% of comments were good predictions, without any errors. **The difference between BLEU scores, presented on Figure 4, and expert evaluations suggests that BLEU is not a suitable metric for assessing the quality of code comments in non-English languages.** BLEU is a universally used metric, which is well known in the field. However, this finding indicates a need for utilising more reliable evaluation metrics that can better reflect the quality of generated comments.

The percentage of correct comments (25.2%) demonstrates that there is substantial room for improvement in the model's performance in Polish. Enhanced training data and more accurate evaluation methodologies are necessary for non-English languages. What is more, the model's struggles with Polish grammar emphasise the importance of language-specific training and evaluation methodologies. Addressing these challenges is crucial for improving the model's performance in generating accurate and coherent comments not only for English.

6.2 Recommendations

Based on our findings, we recommend the following steps to improve the performance of models like Code Llama in generating code comments for non-English languages:

(1) **Improvement in Training Data:** Create more diverse and well-annotated training datasets, also for languages other than English. Including a multiple languages and ensuring the datasets contain accurate and descriptive comments will help the model learn to generate better comments.
(2) **Addition of Grammatically Sophisticated Data:** With the data which is not exclusively related to code and is more sophisticated, the trained models might encounter less grammatical issues in other languages, as it is the case for English.
(3) **Use of Alternative Evaluation Metrics**: Use other metrics for automated qualitative analysis of generated comments. While BLEU has proven unsuitable, and Rouge-L was only somewhat better, more effective metrics are needed to accurately assess the quality of generated comments.

6.3 Future work

To build on our findings, future work should extend the evaluation to other non-English languages. Additionally, it is crucial to encourage greater transparency from model creators regarding their training procedures and origins of data. This will aid in the reproduction and improvement of models, allowing researchers to better understand and address existing limitations.

Future models should also limit comments that contain code commented out in the training corpus. This will improve the quality of the training data by preventing the model from generating commented-out code, which is generally useless and incorrect.

Moreover, given the limitations of BLEU in our study, it is essential to develop alternative evaluation metrics that are better suited for assessing the quality of code comments. Encouraging researchers to create and adopt these new metrics will help establish more reliable standards for automated qualitative analysis.

6.4 Limitations

- **Language Scope:** This study focuses exclusively on Polish, with no English benchmark for comparison, which may limit the comparison between English and other languages.
- **Bias in Labelling:** There is potential bias from individual labellers, which could affect the accuracy of error classification and analysis. Although we implemented strict inclusion criteria for each error to minimise bias, complete mitigation may not be possible.
- **Model Out-of-Scope Usage**: The authors of Code Llama state that the use of the model in languages other than English is considered out of scope. This limits inclusivity and may impact the model's performance.

7 CONCLUSION

In this paper we study the performance of Meta's Code Llama 7B model in generating comments for Java code written in Polish.

Through qualitative analysis using an open coding approach, we identified common error categories in comment generation, resulting in a taxonomy of errors for non-English languages. The most frequent errors in 900 manually analysed generations included the inclusion of code snippets, copying context, and late termination. Polish language-specific challenges, such as frequent grammatical errors and the generation of incomplete or incoherent comments, underline the model's limitations in handling complex syntax.

We highlight the weakness of BLEU as a metric for evaluating code comment quality in non-English languages like Polish. Significant differences between BLEU scores and expert human analysis suggest the need for more suitable evaluation metrics.

Our findings encourage the inclusion of diverse languages in the training corpuses of large language models. Models predominantly trained on English data struggle to generate accurate and contextually appropriate comments in languages with distinct linguistic features. Notably, only 25.2% of the comments were classified as correct, leaving the space for a significant improvement.

8 RESPONSIBLE RESEARCH

In conducting this research we prioritised work integrity, transparency, and reproducibility. We used an open coding approach, which does not assume any preconceived notions about the analysed data. This methodology enables us to identify patterns and categorise data, ensuring that our findings are grounded in the data itself rather than any biases or assumptions of researchers.

To ensure the reproducibility of our research, we have made all datasets and code used in this study publicly available. By doing so, we provide the necessary resources for other researchers to replicate

our work, validate our findings, and build upon our research. The code used in this research is available online in a public GitHub repository[6].

The dataset used for evaluation only contains the publicly available code. We did not train any new models from the data. This approach ensures that our study remains within ethical boundaries and leverages freely available resources.

Our research aims to bridge the gap in the accessibility of LLMs for use cases outside of English. By focusing on generating comments for Java code written in Polish, we address the often overlooked needs of non-English programming environments. Our work propagates the importance of inclusive technology that supports a diverse range of languages and contexts.

ACKNOWLEDGMENTS

To our Supervisors, for their invaluable guidance during the entire project and their excellent feedback. To Mark Zuckerberg, for creating this awesome model, even though we know you're just trying to build an army of code-commenting robots. To my colleagues from the research group: thank you for our collaboration and striving towards this common goal. Also, to the Polish programming community for providing so many hilarious and unrelated comments in the GitHub files. You kept me very much entertained while conducting this mundane task of labelling.

REFERENCES

[1] Anisha Agarwal, Aaron Chan, Shubham Chandel, Jinu Jang, Shaun Miller, Roshanak Zilouchian Moghaddam, Yevhen Mohylevskyy, Neel Sundaresan, and Michele Tufano. 2024. Copilot Evaluation Harness: Evaluating LLM-Guided Software Programming. arXiv:2402.14261 [cs.SE]

[2] Bogdan Babych. 2014. Automated MT evaluation metrics and their limitations. , 464–470 pages.

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

[4] Brett A. Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[5] Lenz Belzner, Thomas Gabor, and Martin Wirsing. 2024. Large Language Model Assisted Software Engineering: Prospects, Challenges, and a Case Study. In *Bridging the Gap Between AI and Reality*, Bernhard Steffen (Ed.). Springer Nature Switzerland, Cham, 355–374.

[6] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* 15, 3 (2024), 1–45.

[7] Boxing Chen and Colin Cherry. 2014. A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, and Lucia Specia (Eds.). Association for Computational Linguistics, Baltimore, Maryland, USA, 362–367. https://doi.org/10.3115/v1/W14-3346

[8] 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]

[9] Hyesun Choung, Prabu David, and Arun Ross. 2023. Trust in AI and its role in the acceptance of AI technologies. *International Journal of Human–Computer Interaction* 39, 9 (2023), 1727–1739.

[10] Mohammad Fraiwan and Natheer Khasawneh. 2023. A Review of ChatGPT Applications in Education, Marketing, Software Engineering, and Healthcare: Benefits, Drawbacks, and Research Directions. arXiv:2305.00237 [cs.CY]

[11] M. Izadi, J. Katzy, T. van Dam, M. Otten, R. Popescu, and A. van Deursen. 2024. Language Models for Code Completion: A Practical Evaluation. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 956–968. https://doi.ieeecomputersociety.org/

[12] Fajri Koto, Nurul Aisyah, Haonan Li, and Timothy Baldwin. 2023. Large Language Models Only Pass Primary School Exams in Indonesia: A Comprehensive Test on IndoMMLU. *EMNLP 2023 - 2023 Conference on Empirical Methods in Natural Language Processing, Proceedings* (2023), 12359 – 12374.

[13] Zihao Li, Yucheng Shi, Zirui Liu, Fan Yang, Ninghao Liu, and Mengnan Du. 2024. Quantifying Multilingual Performance of Large Language Models Across Languages. arXiv:2404.11553 [cs.CL]

[14] 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

[15] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics* 4 (2016), 521–535.

[16] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the Middle: How Language Models Use Long Contexts. CoRR abs/2307.03172 (2023).

[17] Xin Luo, Han Li, Jie Zhang, and J.P. Shim. 2010. Examining multi-dimensional trust and multi-faceted risk in initial acceptance of emerging technologies: An empirical study of mobile banking services. *Decision Support Systems* 49, 2 (2010), 222–234. https://doi.org/10.1016/j.dss.2010.02.008

[18] Antonio Mastropaolo, Emad Aghajani, Luca Pascarella, and Gabriele Bavota. 2021. An empirical study on code comment completion. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 159–170.

[19] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *arXiv preprint arXiv:2402.06196* (2024).

[20] Jessica Ojo, Kelechi Ogueji, Pontus Stenetorp, and David Ifeoluwa Adelani. 2024. How good are Large Language Models on African Languages? arXiv:2311.07978 [cs.CL]

[21] 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 on Association for Computational Linguistics* (Philadelphia, Pennsylvania) *(ACL '02)*. Association for Computational Linguistics, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[22] 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. 2023. Code Llama: Open Foundation Models for Code.

[23] Max Schäfer, Sarah Nadi, Aryaz Eghbali, and Frank Tip. 2024. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. *IEEE Transactions on Software Engineering* 50, 1 (2024), 85–105. https://doi.org/10.1109/TSE.2023.3334955

[24] Demin Song, Honglin Guo, Yunhua Zhou, Shuhao Xing, Yudong Wang, Zifan Song, Wenwei Zhang, Qipeng Guo, Hang Yan, Xipeng Qiu, et al. 2024. Code Needs Comments: Enhancing Code LLMs with Comment Augmentation. *arXiv e-prints* (2024), arXiv–2402.

[25] Florian Tambon, Arghavan Moradi Dakhel, Amin Nikanjam, Foutse Khomh, Michel C. Desmarais, and Giuliano Antoniol. 2024. Bugs in Large Language Models Generated Code: An Empirical Study. arXiv:2403.08937 [cs.SE]

[26] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2:

[6] https://github.com/mziem/LLM-of-Babel

Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]
[27] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 353–355.
[28] Chaozheng Wang, Zongjie Li, Cuiyun Gao, Wenxuan Wang, Ting Peng, Hailiang Huang, Yuetang Deng, Shuai Wang, and Michael R. Lyu. 2024. Exploring Multi-Lingual Bias of Large Code Models in Code Generation. arXiv:2404.19368 [cs.SE]
[29] Wikipedia contributors. 2023. ROUGE (metric) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=ROUGE_(metric)&oldid=1187242316 [Online; accessed 10-June-2024].
[30] 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*. 1–10.
[31] Lele Zhao, Liping Zhang, and Sheng Yan. 2019. A Survey on Research of Code Comment Auto Generation. *Journal of Physics: Conference Series* 1345, 3 (nov 2019), 032010. https://doi.org/10.1088/1742-6596/1345/3/032010
[32] Albert Ziegler, Eirini Kalliamvakou, X. Alice Li, Andrew Rice, Devon Rifkin, Shawn Simister, Ganesh Sittampalam, and Edward Aftandilian. 2022. Productivity assessment of neural code completion. 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, 21–29. https://doi.org/10.1145/3520312.3534864

A TAXONOMY

This appendix contains the detailed descriptions of each error category in the taxonomy. For each error category, the inclusion criteria are given.

A.1 Model Specific Errors

Model specific errors are all errors which are related to the workings of LLMs.

Incoherent Generation (MS-IG). Model outputs random words which have no logic between them

Copy Context (MS-CC). Model copies the surrounding context verbatim

Memorisation. Model recognised the code to some capacity

(1) *Contains PII (MS-ME1).*
Personally identifiable information is included in the generated comment (fictional or did not occur in the original prompt)
(2) *Contains URL (MS-ME2).*
URL for a file or repository is included
(3) *Verbatim Memorization (MS-ME3).*
1) The model memorized the content verbatim
2) the text would not be generated if not for memorization

Repetition. Model generates and repeats what it has already said it in some capacity

(1) *Pattern repetition (MS-RE1).*
Model generated a repeating pattern: eg. 1,2,3,4,5,6,
(2) *Verbatim repetition (MS-RE2).*
Model generated verbatim repetition: eg. i am repeating i am repeating i am repeating

Early Termination (MS-ET). Model stops generating in the middle of prediction, while the comment is clearly not complete or did not generate anything

Late Termination (MS-LT). 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

A.2 Linguistic Errors

Linguistic errors are all errors related to the linguistic content of the generated text

Grammar. Language is correct, grammatical mistake was made.

(1) *Plurality (LG-GR1).*
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.")
(2) *Conjugation (LG-GR2).*
Incorrect usage of conjugation rules
(3) *Gendering (LG-GR3).*
Incorrect gendering in case the language has gendered nouns
(4) *Spelling (LG-GR4).*
Incorrect spelling
(5) *Capitalisation (LG-GR5).*
Prediction capitalizes letters that grammatically is not correct to capitalize: e.g all capitals, every word begins with capital
(6) *Cohesion (LG-GR6).*
a) Mistake in using a language that involves organizing words and phrases that don't make sense (incoherence).
b) Missing (or inappropriate usage of) a comma or a quotation mark
c) 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).
d) 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."

Wrong language. The model predicts a comment (or significant part of it) in a language other than the target language

(1) *Undesired translations (LG-WL1).*
Translations that are correct but undesired in the language because the words are seldomly used in that context.
(2) *Incorrect language (LG-WL2).*
The model predicts a comment (or significant part of it) in a language other than the target language.

Usage of incorrect synonym (LG-IS). Usage of a similar word with an incorrect meaning in context (e.g. home->house)

A.3 Semantic Errors

Semantic errors are all errors related to the semantics or meaning of the generated content.

Hallucination. Category for hallucination generations, i.e. factually incorrect or not related to input prompt.

(1) *Misplaced Facts (SE-HA1).*
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/finctional person.
(2) *Contextual Discrepancy (SE-HA2).*
Hallucination not grounded in the provided context.
(3) *Educated Guess (SE-HA3).*
a) Syntactically correct but semantically or factually incorrect
b) Grounded in the provided context.

Code Snippet Inclusion. Model generates actual code outside of comment

(1) *Commented out code (SE-CS1).*
Code that resides in a code block.
(2) *Code intended to run (SE-CS2).*
Code that the model intends to run.

Missing Details (SE-MD). Description does not fully describe the content of the summarized code. Current information does not describe the full functionality of code being summarized. Current information does not describe the entire purpose of the summarized code. Generated comment is too generic.

Too Specific (SE-TS). Model copies the surrounding context verbatim.

A.4 Syntax Errors

Syntax errors are all errors which are related to the syntax of the comments.

Incorrect comment format. Model uses outdated format of javadoc. Model uses comment format that is inconsistent with the standards. Errors with javadoc format.

(1) *Style Inconsistency (ST-IF1)*
a) Model uses outdated format of javadoc
b) Model uses comment format that is inconsistent with the standards
c) Model repeated auto-generated-comment like format which is not informative enough, instead of generating an actual description
d) Model does not follow the javadoc format that is present in the rest of the file (if present format is correct)
(2) *Omitted Identifier (ST-IF2)*
a) Model starts enlisting @params, but misses some of them
b) Generation started with a tag @return but then doesn't have @params
c) Generated @params, but does not have @return for a method that does not return void

A.5 Miscellaneous

Anything that does not fall into any of the above categories

B EXAMPLES OF CATEGORIES

This appendix contains the examples of selected error categories.

B.1 Errors Related to Token Splitting

(1) **Original comment:**
`// DFS na tablicy stanów - sprawdzenie wszystkich możliwości wzięcia setów.`
Masked-out comment:
`// DFS na <SUF>`
Predicted comment:
`// DFS nawetrzymuje <rest of the generation where late termination occured, omitted for clarity>`
The word "nawetrzymuje" does not exist in Polish and is a failed attempt of trying to generate something meaningful in the middle of the token.
(2) **Original comment:**
`//bank ma listę bankomatów i klientów`
Masked-out comment:
`//bank ma <SUF>`
Predicted comment:
`//bank matów`
The generated word does not make any sense in the provided context and is an incomplete word for this context. This erroneous generation is also caused by mid-token invocation.

B.2 Excluded comments

(1)
```
/**
* @author Marcin Miłkowski
*/
```
(2)
```
/**
* 获得rowid
*
* @param field rowid属性名
* @return RowId
*/
```
File contains Chinese characters.
(3) `// How big do we draw our circle?`
Comment is in English, not Polish.
(4) `//log(x) = z*(2+zˆ2*((2.0/3)+zˆ2*((2.0/5))))` Comment is a mathematical formula, with no point to predict to evaluate performance in Polish.