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

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

An electronic version of this thesis is available at http://repository.tudelft.nl/.

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

Paris Loizides
Delft University of Technology
Delft, The Netherlands

ABSTRACT

This paper evaluates the performance of Large Language Models, specifically StarCoder 2, in non-English code summarization, with a focus on the Greek language. We establish a hierarchical error taxonomy through an open coding approach to enhance the understanding and improvement of Large Language Models in multilingual settings as well as identify the challenges associated with tokenization and influence by mathematical datasets. Our study includes a comprehensive analysis of error types, tokenization efficiency, and quantitative metrics such as BLEU, ROUGE, and Semantic Similarity. The findings highlight the importance of semantic similarity as a reliable performance metric and suggest the need for more inclusive tokenizers and training datasets to address the limitations and errors in non-English contexts.

KEYWORDS

LLMs, Multilingual, NLP, Tokenization, Greek language, Code summarization, Hierarchical Error Taxonomy, Evaluation metrics

1 INTRODUCTION

The integration of Large Language Models (LLMs) such as GPT[1] and BERT[2] into software development has significantly enhanced coding efficiency and productivity, particularly for English-speaking environments [28, 36]. At the same time, broader educational benefits have also been highlighted [4]. Despite their advantages, there is a notable performance disparity when these technologies are applied to non-English programming contexts, which limits their global applicability and effectiveness [26]. This raises important questions about the inclusivity and accessibility of AI tools in diverse linguistic landscapes.

This research is concerned with the main question: "How does the performance of Large Language Models vary across different comment generation tasks when applied to non-English languages?". We aim to investigate this by focusing specifically on the Greek language, exploring the application of LLMs like StarCoder 2 in comment completion and code summarizing tasks, and identifying the challenges and distinct patterns in the errors that occur most frequently. We aim to raise awareness and initiate discussion on issues faced in non-English code completion, that will eventually lead to an increased availability of multi-lingual AI tools, and thus extend and deploy the benefits of these models on a global scale.

To systematically analyze and categorize the types of errors, we employ an open coding approach. This method involves breaking down data, labelling them, and comparing for similarities and differences [14, 17]. By iteratively refining these labels, we develop a hierarchical error taxonomy that provides a structured overview of the errors.

Our study makes several key contributions by answering the following questions:

(1) What types of errors are most common in Greek and other non-English languages, and how can a hierarchical error taxonomy help guide future developments in LLM technology?
(2) How does the tokenization process of prompts affect the performance of LLMs in recognizing Greek and generating comments?
(3) What is the quantitative performance of StarCoder 2 in code summarization when prompted with Greek-documented code snippets?

2 RELATED WORK

A significant disparity exists in the natural language usage within code files, with English being overwhelmingly dominant compared to non-English languages. This disparity presents challenges for multilingual LLMs, which must be addressed to improve their performance and inclusivity across different linguistic contexts [26].

Recent advancements in enhancing the multilingual capabilities of LLMs have focused on overcoming resource disparities and improving performance across diverse languages. Different approaches have been proposed to improve the performance of under-resourced languages by utilizing the knowledge of models in resource-rich languages [12, 19, 34]. Additionally, multilingual sentence embeddings have been extended from monolingual models using knowledge distillation. This technique aligns vector spaces across multiple languages, ensuring accurate semantic understanding and enhancing the utility of sentence embeddings in diverse linguistic contexts. These efforts aim to address the challenges posed by multilinguality in LLMs [27].

The open coding approach is widely used in qualitative research to systematically categorize and label data. In the context of LLM evaluation, this approach facilitates a detailed analysis of model outputs by closely examining, and identifying similarities and differences [14, 17]. This method helps in identifying common errors and patterns, which is crucial for developing error taxonomies and improving model performance. The iterative process of open coding ensures that labels are refined and updated based on continuous analysis for error identification and correction. Notable research involves a hierarchical error taxonomy to systematically categorize errors produced by LLMs during code completion tasks, but also a related work that uses open coding for the qualitative analysis of code snippet translation to English comments [16, 21, 23]. These

[1] https://platform.openai.com/docs/models
[2] https://huggingface.co/docs/transformers/en/index

taxonomies aid in identifying common pitfalls and guiding model improvements.

In the field of Natural Language Processing (NLP), error taxonomies are commonly used to systematically identify and categorize errors [5, 9, 13, 15, 30–32]. These taxonomies are crucial for understanding the types and sources of errors in automatically generated text. Consequently, we adopt a similar approach to develop a hierarchical error taxonomy for multilingual comment completion, an area that remains under explored.

Tokenization is a critical aspect of training LLMs, significantly influencing their performance and training efficiency, especially for multilingual models [20]. Research highlights that the choice of tokenizer can substantially impact model performance and training costs, emphasizing its importance in the pre-training phase [1]. Factors such as the tokenizer's size, pre-tokenization regular expressions, and training data can affect generation speed, effective context size, memory usage, and overall model effectiveness [10]. Using a specialized monolingual tokenizer for each language can enhance performance compared to using a single multilingual tokenizer, particularly for adequately represented languages [29]. These insights underscore the necessity of selecting and optimizing tokenizers for better performance and efficiency especially in multilingual LLMs.

Notable research has been conducted for the evaluation of LLMs in programming tasks. Significant contributions in code evaluation are different proposed techniques and metrics to successfully measure the ability of a model to predict code [2, 7, 11]. Further evaluation metrics such as CodeBERTScore have been developed, effectively comparing a ground-truth reference with code snippets generated by models, by computing cosine similarity among the sequence of vectors generated from the tokens sequence of pretrained models [35]. A similar approach to extract semantic similarity of doc-strings and other code is applied in ReCode benchmark [33].

3 METHODOLOGY

3.1 Collaborative Efforts

This project involves a collaborative effort among five teammates, each working on a different language and LLM: Dutch, Chinese, Polish, and Greek. The goal of this collaboration is to identify error labels and patterns across these languages and models. Weekly meetings are held to discuss findings, iteratively update the Hierarchical Error Taxonomy, and ensure consistency in the experimental process. This collaborative approach helps answer the main research question by investigating the performance of a variety of models and languages, resulting in a more generalized conclusion and taxonomy of errors.

3.2 Experiment Setup and Pipeline

We selected StarCoder 2 for its impressive performance in various evaluation metrics and trained using the Fill-in-the-Middle (FIM) objective to generate code summarization predictions [3, 22]. The pipeline, used a max token length of 215 for Greek, motivated in subsection 4.4.

3.3 Open Coding

Open Coding Approach. Open coding approach is utilized for the qualitative analysis of the model's performance in order to answer research question 1. This approach is used to analyze qualitative data by labelling them iteratively, closely examining, and comparing them for similarities and differences [17]. For our experiment, the analysis is split into 2 steps:

(1) **Error Identification**: The outputs generated by the model are carefully reviewed to identify instances where the model's predictions diverge from the expected correct completions. These errors are updated weekly in every new iteration of the Error Taxonomy. Newly found errors are noted down for further analysis and comparison with similar error patterns by other models in different languages.
(2) **Hierarchical Taxonomy Development**: A hierarchical taxonomy of errors is developed to provide a structured overview of the types of errors encountered. This taxonomy categorizes errors into broad types and further subcategories based on specific characteristics.

Labelling. Labelling was a crucial part of the experimental process and open coding approach in order to ensure consistency among the research team. Throughout the Hierarchical Error Taxonomy iterations, different labelling schemas were adapted to ensure minimal bias from the dataset, but also to extract as many meaningful error labels as possible efficiently. The labelling schemas were as follows:

(1) **Initial Iteration:** 3 Error labels per comment with a detailed explanation for inclusion criteria for each label.
(2) **Second Iteration:** 3 Error labels per comment while at the same time limiting to a total of 5 labels per file to avoid overpopulation of the labelling pool by dominant files. Include any remarks and observations of new error patterns.
(3) **Final Iteration:** Only taking 1 randomly selected comment per repository, mark all Error labels identified and any remarks, for discussion with the research team.

3.4 Tokenization Experimental Approach

Tokenization for StarCoder 2. To address research question 2, we discuss the tokenizer for the StarCoder 2 model. According to the technical report of StarCoder 2, the tokenizer is a byte-level Byte-Pair Encoding (BPE) tokenizer trained on a small subset of The Stack v1[3] dataset. The pre-tokenization step involves using a digit-splitter and regex splitter from the GPT-2 pre-tokenizer. The tokenizer has a vocabulary size of 49,152 tokens [22].

While the tokenizer itself was not trained on the OpenWebMath dataset, the StarCoder 2 model was trained on a more extensive dataset that includes OpenWebMath which consists of 14.7 billion tokens of high-quality mathematical documents from the web [22]. This dataset features Greek letters in various mathematical documents[4] [25]. We want to explore the effect of mathematical context

[3] https://huggingface.co/datasets/bigcode/the-stack

[4] A mathematical document consists of fundamental mathematical content, including theorems, definitions, proofs, questions and answers, formal mathematics, or interdisciplinary documents that feature mathematical formulas in fields like physics, chemistry, biology, economics, and finance

in the performance of StarCoder 2 when dealing with the Greek alphabet in a natural language context.

The Stack v1 Dataset. Additional analysis of The Stack v1, on which the tokenizer for StarCoder 2 is built, reveals that 94% of the Python files used to extract docstrings and comments are in English, with only 12 files in Greek (el) [25]. This low frequency of Greek files means that the tokenizer has limited exposure to Greek text during training, potentially leading to sub-optimal tokenization for Greek letters.

Furthermore, the distribution of programming languages in The Stack v1 indicates a wide variety of languages, but very few support the use of the Greek alphabet natively in code. Most programming languages, such as HTML, JavaScript, Java, Python, C++, and others, primarily use the Latin alphabet. However, it is important to note that while the source code itself may not use the Greek alphabet, comments and documentation within the code can still include Greek letters, especially in mathematical contexts. This does not imply that the tokenizer is able to build tokens of Greek words for natural language processing since Greek is still underused in the dataset.

Experimental Evaluation. To evaluate the impact of this mathematical context on the tokenization of StarCoder 2, we designed the following experiment to gain insights into the information density of Greek tokens for StarCoder 2 in comparison to a Greek-based tokenizer and a mathematical-based tokenizer:

We compare three distinct tokenizers:

(1) **StarCoder 2 Tokenizer**
(2) **Meltemi-7B-v1**[5]: Tokenizer of the first Greek Large Language Model (LLM)
(3) **OpenWebMath Custom Tokenizer:**
    - Built on 50% of the OpenWebMath HuggingFace Dataset[6].
    - Pre-tokenized using a digit-splitter and regex splitter from the GPT-2 pre-tokenizer, similar to the StarCoder 2 tokenizer.
    - Utilizes byte-level Byte-Pair Encoding (BPE) like the StarCoder 2 tokenizer.

All extracted comments were encoded using the different tokenizers. We compared the length of tokenized comments to assess the information density of each tokenizer.

3.5 Quantitative Evaluation Metrics

To quantitatively assess the performance of the StarCoder 2 model and answer research question 3, we employ several well-established metrics in the field of natural language processing and n-grams comparison:

- **Accuracy**: This metric evaluates the correctness of the code completions provided by the model. Accuracy is the percentage of acceptable comments, determined by manually comparing the model's output to the expected correct completions and the ability of the model to correctly summarize and describe the related code snippet in Greek.
- **BLEU Score (Bilingual Evaluation Understudy)**: This metric is used to evaluate the quality of the generated code by comparing the n-gram overlap between the model's output and the reference code snippets. For the experiment, we are using the Sentence Bleu score, with the Smoothing Function technique Method 4, which scales inflated scores for short sentence translations. The use of the smoothing function leads to better correlation with human judgment [6]. A higher BLEU score indicates better performance [24].
- **ROUGE Score (Recall-Oriented Understudy for Gisting Evaluation)**[7]: This is used to measure the recall of the generated comment snippets, particularly focusing on how much of the reference comment is captured by the model's output. We are calculating the ROUGE-1 F score which compares the fraction of the n-grams in the prediction that are also in the reference.
- **Sentence transformer (semantic similarity)**: Multilingual Sentence Transformers extend monolingual models to multiple languages via multilingual knowledge distillation, aligning vector spaces across languages. Using XLM-R, pre-trained on 100 languages including Greek, it enables accurate multi-lingual semantic understanding [8, 27]. We first encode the reference and prediction and calculate the cosine similarity between the vector representations to obtain the semantic similarity score. Similar to the approach of established metrics like CodeBertScore and ReCode benchmark [33, 35].

4 DATA

4.1 Greek Language in NLP

The Greek language occupies a unique position among languages like Dutch, Polish and Chinese. It shares underlying semantic rules with other Latin-based languages while also presenting distinct challenges with its own alphabet and complexity. The Greek script, consisting of 24 letters with unique Unicode representations, is widely used in academia and programming, especially in mathematics, physics, and engineering. The language also features diacritical marks such as tonos (´) and diaeresis (¨) that modify pronunciation and meaning. Accurate handling of these marks is crucial for correct tokenization and text representation.

Greeklish is the practice of writing Greek using the Latin alphabet. While Greeklish can simplify typing, it introduces inconsistencies in spelling and can complicate NLP tasks due to the lack of standardized translation rules. For the purposes of this research, we focus exclusively on data written in the Greek alphabet, as the primary aim is to analyze and improve model performance on native modern Greek script.

Evaluation tasks have been developed targeting natural language inference, word sense disambiguation, and metaphor detection, underscoring the need for specialized datasets to advance the Greek NLP ecosystem [18].

4.2 Dataset Creation and Filtering

To conduct the experiments, a dataset consisting of Greek code snippets was prepared by extracting available Java files with Greek text from Github, using the top 2500 most frequent Greek words.

[5] ilsp/Meltemi-7B-v1
[6] open-web-math/open-web-math
[7] Understanding ROUGE score for NLP Evaluation

The Dataset filtering was necessary to improve the inference so that errors are identified correctly. The following dataset filtering was conducted on the files:

(1) Large files exceeding the context window of 8,192 tokens.
(2) Files that do not contain any comments.
(3) Duplicate files.

After extracting line comments and block comments using Regular Expressions[8], we filter out instances of comments that are not Greek using Fasttext-Langdetect[9].

4.3 Data Masking and Formating

Since StarCoder 2 has been trained using the Fill-In-the-Middle (FIM) objective, we need to maintain the same data input format for inference. Therefore, we will need to perform masking and preprocessing, resulting in a format that the model will understand [3]. The following operations are applied on the dataset to spanmask the comments using the FIM objective:

(1) Extract all block and line comments from the filtered dataset.
(2) Process content of the files by including FIM tags used by StarCoder 2 during training, as explained in the technical report [22]. The resulting file content will be as follows:

    <fim_prefix>*pre_code****<fim_suffix>****suf_code****<fim_mid>***

(3) To indicate the intended language of the predicted comment to the model, we retain the first 3 words of the original comment for block comments, and the first 2 words for line comments as part of the pre_code.

4.4 Dataset Statistical Analysis

To decide on the maximum token length for pipeline initialization, we tokenized the extracted original comments and calculated the total number of tokens for each comment. The mean number of tokens was computed to understand the average tendency of our dataset (*mean=75 tokens*). In the token length distribution, we observed a right-skewed shape, with notable spikes of lengths greater than 1500 tokens. This can be attributed to commented-out code snippets and long Greek text, unrelated to code, such as exercise descriptions and random text. We determined the ***95th*** percentile token length at ***215 tokens*** to achieve a balance between completeness and efficiency in the model's predictions.

5 RESULTS

5.1 Open Coding Taxonomy

Hierarchical Error Taxonomy. To answer the first research question, table 1 visualizes the final Hierarchical Error Taxonomy, which is the result of weekly labelling iterations and refinements of the different labels and categories. Table 1 lists the agreed taxonomy among the research team, as well as the frequency of the different errors in Greek comment generation for StarCoder 2.

Qualitative Results. The qualitative analysis aims to identify the most frequent error labels within the model's predictions. The

Table 1: Hierarchical Taxonomy of Error Labels for the Greek language on 300 labeled data points of the final iteration.

Error label and ID	Count
Comment Generation Errors	**719**
Syntax	16
(ST-IF) Incorrect Comment Format	16
(ST-IF1) Style Inconsistency	14
(ST-IF2) Omitted Identifier	2
Linguistic	48
(LG-IS) Incorrect Synonym	0
(LG-WL) Wrong Language	12
(LG-WL1) Undesired Translation	3
(LG-WL2) Incorrect Language	9
(LG-GR) Grammar	36
(LG-GR1) Plurality	1
(LG-GR2) Conjugation	0
(LG-GR3) Gendering	16
(LG-GR4) Spelling	2
(LG-GR5) Capitalization	1
(LG-GR6) Cohesion	16
Semantic	315
(SE-MD) Missing Details	22
(SE-TS) Too Specific	6
(SE-CS) Code Snippet Inclusion	195
(SE-CS1) Commented out Code	14
(SE-CS2) Code Intended to Run	181
(SE-HA) Hallucination	92
(SE-HA1) Misplaced Facts	4
(SE-HA2) Contextual Discrepancy	24
(SE-HA3) Educated Guess	64
Model Specific	222
(MS-IG) Incoherent Generation	16
(MS-CC) Copy Context	89
(MS-ET) Early Termination	7
(MS-LT) Late Termination	32
(MS-ME) Memorization	37
(MS-ME1) Contains PII	3
(MS-ME2) Contains URL	10
(MS-ME3) Verbatim Memorization	24
(MS-RE) Repetition	41
(MS-RE1) Pattern Repetition	12
(MS-RE2) Verbatim Repetition	29
Miscellaneous	30
Excluded	88

[8] https://docs.python.org/3/library/re.html
[9] https://pypi.org/project/fasttext-langdetect/

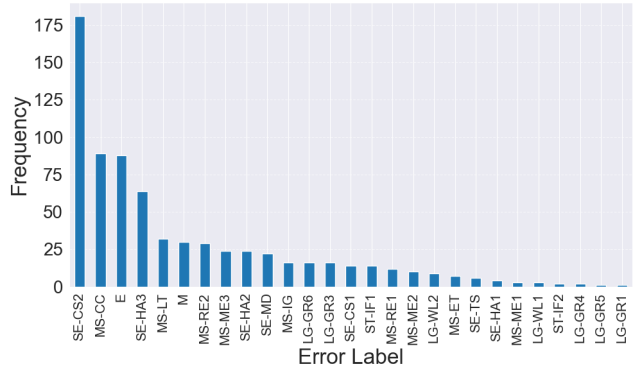

Figure 1: Error Labels Distribution of Final Iteration using 300 comment predictions.

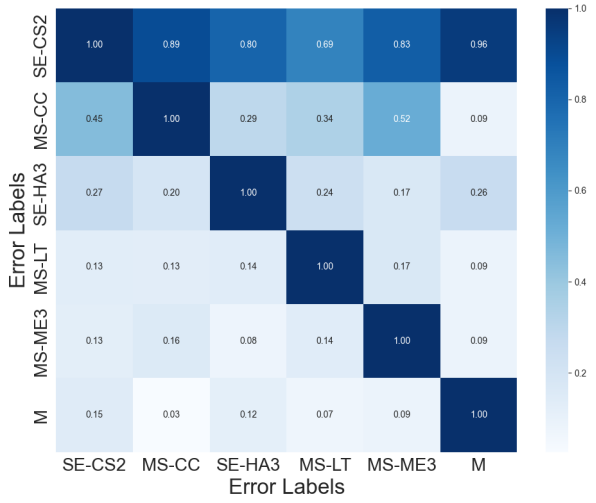

Figure 2: Heat Map of the top 6 most common errors and their percentage of asymmetric co-occurance.

distribution of error labels in the Greek dataset is depicted in Figure 1.

The most common error label identified was SE-CS (Includes Code Snippet), which appeared with the highest frequency of 181 occurrences. This was followed by MS-CC (Copying Context), E for Excluded labels due to extracted comment that do not provide meaningful insights when labelled, and SE-HA3 (Educated Guess) which is a type of Hallucination by the model. The excluded errors (E), may contain: **(1)** Commented out code snippet, **(2)** URLs, **(3)** Names, **(4)** File related information such as course name or author information, **(5)** Discussion comments among authors **(6)** "TODO:" comments.

A notable Error label was M (Miscellaneous) which occurred 30 times in the data. This label describes all errors that were not considered significant enough to be included in the Taxonomy due to their limited appearance when iterating over the Error taxonomy with the research team. For the Greek language and StarCoder2,

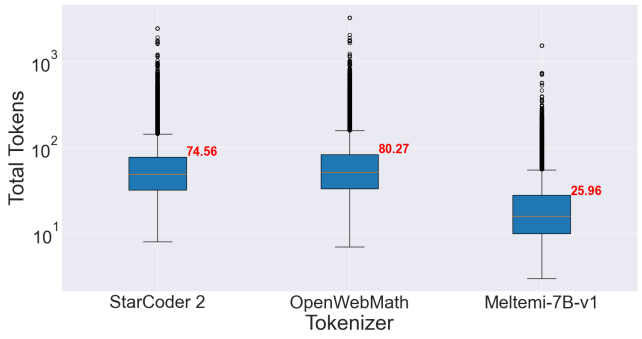

Figure 3: Box Plot comparison of the number of tokens for the StarCoder 2, Meltemi, and OpenWebMath Tokenizers on 14,524 Greek comments, the mean score for each tokenizer in red text, plotted on a logarithmic scale on the Y axis. A lower number of tokens is better.

these errors include the following: **(1)** Incorrect use of accents, **(2)** Declension[10] errors in grammar, **(3)** Predicted comment ends with a closing curly bracket (}), **(4)** Generation of in/out-of-context getters and setters methods, **(5)** Generation of Document Variables as jQuery[11] construct.

Further qualitative analysis is visualized in Figure 2. The co-occurrence heat map demonstrates the relation between the most common errors in the generated comments. It is obvious that due to the high frequency of SE-CS2 (Code intended to run), all errors are highly correlated with predicting code, highlighting the weakness of the model in correctly terminating the token generation when the comment is complete. Furthermore, a high correlation of 0.45 is noted among the pair SE-CS2 and MS-CC (Copy Context) as well as 0.27 for SE-CS2 and SE-HA3 (Educated Guess). This means that almost 50% of the times that the model predicts code snippet intended to run, it is just copying the context of the inferred file, while 27% of the times the generated code is a hallucination, grounded to the provided context. Lastly an important co-occurence would be that among MS-ME3 (Verbatim Memorization) and MS-CC (Copy Context) with a value of 0.52. This suggests that in more than half of the instances where memorization from the training data is observed in the generated Greek comment, the generated code is copied from the provided file. This could also explain that the copied context is memorized from the training data, indicating overfitting on files that contain Greek documented code and were part of the model's training set.

5.2 Tokenization Experimental Results

In order to answer research question 2, we present and analyze the experimental results of tokenization for different tokenizers, focusing on the analysis of Greek tokens. We compare the StarCoder v2 tokenizer, the Greek Tokenizer (Meltemi), and the OpenWebMath Tokenizer in terms of their efficiency and information density.

[10]Greek nouns, pronouns, and adjectives alter their form based on their grammatical role, a process referred to as declension.
[11]jQuery API Documentation

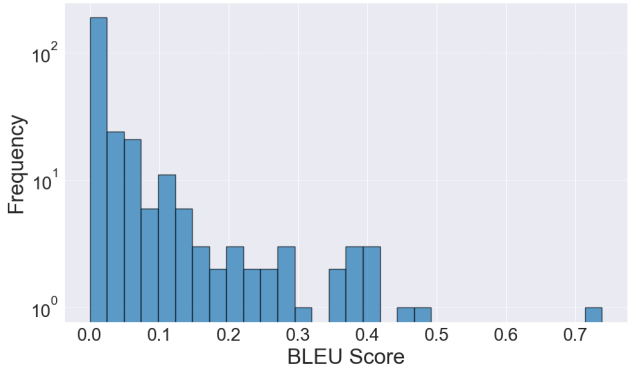

Figure 4: Distribution of BLEU Scores of 430 data points from the last iteration, random comment selection from all repositories, plotted on a logarithmic scale on the Y axis.

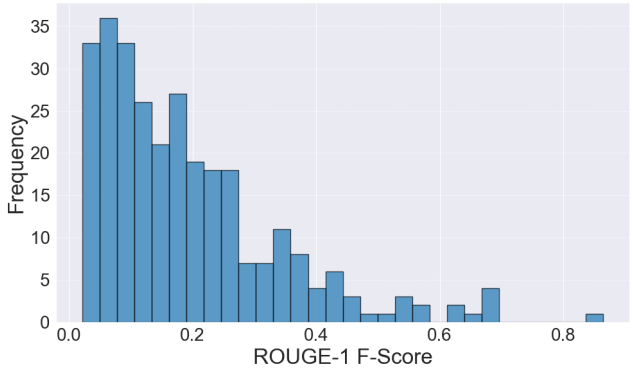

Figure 5: Distribution of ROUGE-1 F Scores of 430 data points from the last iteration, random comment selection from all repositories.

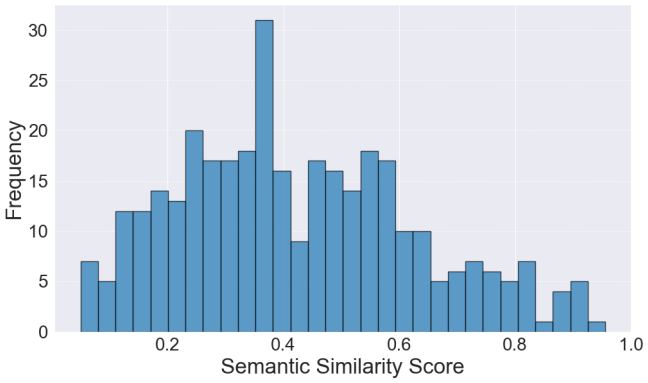

Figure 6: Distribution of Semantic Similarity Scores of 430 data points from the last iteration, random comment selection from all repositories.

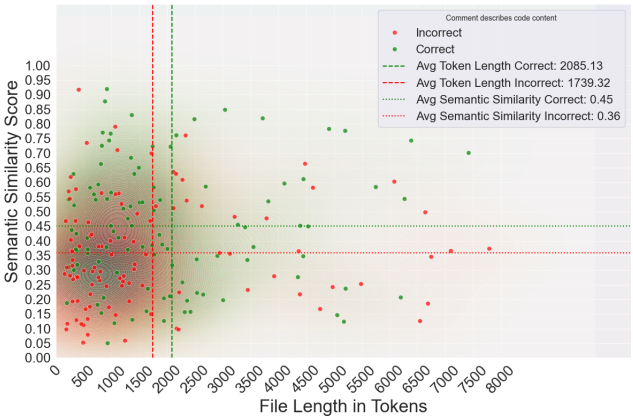

Figure 7: Scatter plot with Kernel Density Estimation (KDE) of Semantic similarity for 300 labelled Correct and Incorrect comments.

Greek Tokens Analysis. **StarCoder v2:** We analyzed the tokens of the StarCoder v2 tokenizer and found that out of a total of 49,152 tokens, only 39 are Greek, which translates to a ratio of 0.0008. All of the Greek tokens have a maximum length of 2 characters. These 39 tokens do not include Greek capital letters, which are split into two non-human readable tokens by the StarCoder 2 tokenizer.

Greek Tokenizer (Meltemi): This tokenizer extends the Mistral-7B tokenizer with Greek tokens. Out of a total of 61,362 tokens, 28,136 tokens are Greek, which is 46% of all tokens. Additionally, 95% of these Greek tokens are longer than 2 characters, leading to an efficient Greek tokenizer with meaningful information density per token.

OpenWebMath Tokenizer: Trained on 3 million mathematical documents, which is 50% of the entire dataset, this tokenizer yields a total of 90 Greek tokens out of 49,152, resulting in a ratio of 0.0018. Out of these Greek tokens, 85% have a length of less than or equal to 2 characters, most of which are mathematical symbols such as: ["Δx", "φ", "Δt", "π", "χ", "Δ", "μm", "ξ", " μs"].

Information Density of Tokenizers: Comparison. The box plot of Figure 3 comparing StarCoder 2, OpenWebMath, and Meltemi-7B-v1 tokenizers reveals significant differences in tokenization and information density for Greek comments. On average, StarCoder 2 produces longer token sequences (74.56 tokens) compared to Meltemi-7B-v1 (25.96 tokens), indicating that StarCoder 2's tokenizer tends to tokenize Greek text one letter at a time, resulting in inefficient representation. The higher number of outliers for StarCoder 2 suggests inconsistent tokenization patterns that could hinder performance in Greek contexts. In contrast, the similarity in average token lengths between StarCoder 2 (74.56 tokens) and OpenWebMath (80.27 tokens) indicates that both tokenizers handle mathematical and Greek content similarly. This similarity suggests that StarCoder 2's tokenization leads to long sequences of individual letter tokens that lack the information density necessary for efficient NLP tasks.

5.3 Quantitative Analysis

The results of the quantitative analysis answer research question 3 regarding the different metrics aimed to measure the performance of the model for Greek use cases.

Accuracy Rate. The accuracy of the model as defined in methodology subsection 3.5 is measured at 49% of correct comment predictions that correctly describe the code snippet being summarized.

BLEU. The log scaled distribution of BLEU scores (Figure 4) reveals a significant skew towards lower scores, with the majority of scores clustered around 0. This indicates that many predicted comments have low accuracy compared to their reference comments. Responsible for this data distribution and trend towards score zero, is also the extensive predictions by the model. Code snippets are also generated, thus lowering the BLEU score on most examples when comparing reference to model prediction. However, a few instances achieve higher BLEU scores, suggesting that the model occasionally produces highly accurate syntactic predictions.

ROUGE. Notably, the ROUGE-1 F score distribution (Figure 5), shows a peak at lower scores, with a gradual decline as the scores increase. This pattern aligns with the BLEU score distribution, reinforcing the observation that the model struggles with high recall-oriented accuracy in most cases but can achieve better results in specific instances.

Semantic Similarity. Similarly, the semantic similarity scores (Figure 6) measure the semantic equivalence between the predicted and reference comments, demonstrating how well the model captures the underlying meaning and context.

The semantic similarity score distribution (Figure 6) highlights a more varied performance, with a notable peak around the 0.3-0.5 range. This indicates that despite code snippets being part of the generated prediction, the model often captures the semantic essence of the comments.

Further analysis of the Correct and Incorrect data points, for Semantic Similarity and File Length in terms of tokens (Figure 7) demonstrates a correlation between the two metrics. The Kernel Density Estimation (KDE) visualizes the density of correct predictions around the 0.5 Semantic Similarity score with a file token length of 1000. On the other hand, Incorrect predictions are concentrated at around 0.3 Semantic Similarity Score with a file token length of 500 tokens. Average token length and semantic similarity score for both correct and incorrect predictions are also visualized in Figure 7, confirming the effect of Context token length of inference in the quality of the predictions.

Effectiveness of Quantitative Metrics Comparison. The box plot comparison (Figure 8) highlights the effectiveness of different metrics in evaluating the correctness of comments. Among the metrics, the semantic similarity score best differentiates between correct and incorrect comments, showing a larger gap and higher variance for correct predictions without any outliers. This indicates that semantic similarity is more representative of comment correctness. The low performance across all scores is attributed to the poor quality of Greek documentation which is used as ground truth, and the model's tendency to predict code snippets, which inherently lowers the scores. Taking into consideration these limitations, semantic similarity seems to be the most useful metric, effectively answering research question 3, regarding the performance of StarCoder 2 when prompted with Greek code snippets.

6 DISCUSSION

6.1 Implication

Our findings underscore the necessity for more inclusive and extensive research on multilingual LLMs designed for coding. The hierarchical error taxonomy developed through the open coding approach sheds light on common pitfalls and errors that code LLMs encounter in multilingual contexts. This taxonomy can serve as a valuable tool for guiding future research and development efforts to enhance the inclusivity and effectiveness of these models on a global scale.

The Greek language, in particular, presents unique challenges due to the influence of both the tokenizer and the training data. Our analysis revealed that models not specifically tailored to the Greek language exhibit unexpected behaviors. This can be attributed to the frequent use of Greek characters in mathematical contexts and the limited availability of purely Greek repositories that can result in learning Greek letter tokens in both mathematical and natural language processing contexts. The individual letter tokenization, highlights the lack of extensive Greek content in training datasets like The Stack v1, leading to suboptimal performance in Greek comment generation tasks.

Our evaluation of quantitative metrics revealed that BLEU and ROUGE scores are not reliable indicators of performance in this context due to the generation of code snippets in predicted comments and the poor quality of Greek documentation used as ground truth. Instead, semantic similarity scores, obtained through multilingual transformers, provided a more accurate measure of the quality of predictions. These results align with previous research on similarity scores, such as CodeBERTScore and the ReCode benchmark, which also emphasize the importance of semantic similarity for evaluating the quality of generated content [33, 35].

Additionally, we are hypothesizing that despite the lack of non-English documented datasets, the model's ability to summarize code in languages other than English can be attributed to the contextual similarities of code. This is due to the model mapping specific code snippet contexts to a shared space, allowing for effective transfer across different languages. Generalization capabilities indicate that a shared space is created that facilitates this transfer, thereby enhancing the model's performance even with limited training data for certain languages [26].

6.2 Recommendations

Based on our findings, we propose several recommendations to enhance the performance and inclusivity of LLMs used for multilingual coding tasks:

Development of Multilingual Tokenizers and Models: Greater emphasis should be placed on creating multilingual tokenizers that are inclusive of languages like Greek. At the same time models should be trained on a larger corpus of non-English data. Techniques such as knowledge distillation from resource-rich languages, the use of lexicons, translation of training data, self-translation, and

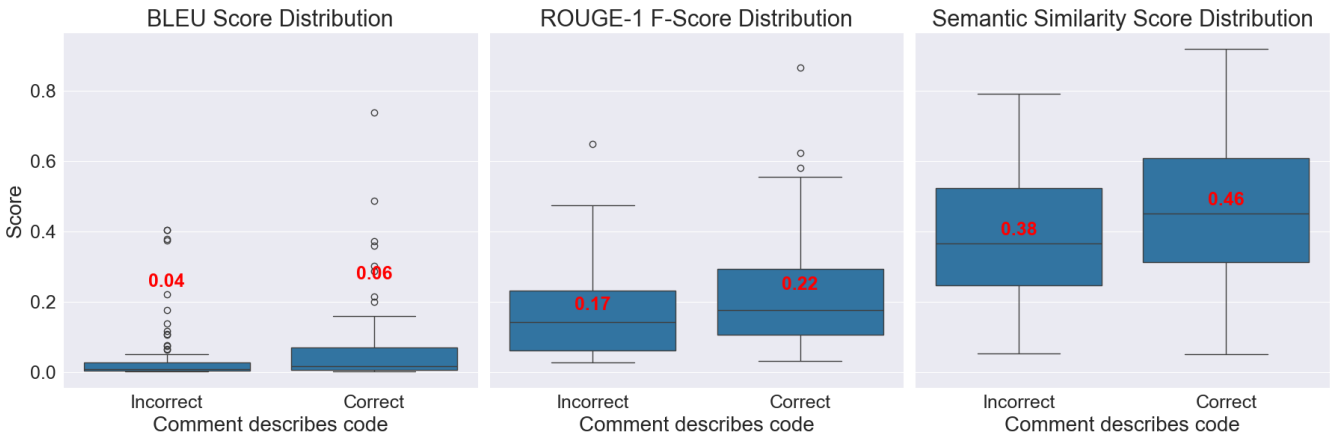

Figure 8: Box plot comparison of Correct and Incorrect predictions for BLEU, ROUGE, Semantic Similarity, and the mean score in red for each score.

alignment of vector spaces demonstrate potential for improving multilingual models performance [12, 19, 27, 34].

Prioritize Semantic Similarity Metrics: In multilingual experiments, semantic similarity metrics should be prioritized for evaluating the quality of generated comments for code snippets. This metric closely aligns with human evaluation of code completion tasks [33, 35]. This approach provides a more accurate and meaningful assessment of model performance compared to traditional metrics like BLEU and ROUGE.

6.3 Limitations

While this study provides valuable insights into the performance of LLMs in non-English programming contexts, several limitations must be acknowledged.

Due to time constraints, the data labelling process was less extensive than desired. Ideally, a larger dataset would be labeled to ensure comprehensive analysis and conclusions. Additionally, having multiple annotators label the same datasets for each language and model would help mitigate individual biases and improve the reliability of the error taxonomy.

Another limitation is the rotation of all discussed languages among the different models. To fully understand the performance variations across languages, it would be beneficial to evaluate each model with every language and model included in the study while at the same time creating an English dataset as a baseline of comparison. This would provide a more accurate view of model capabilities and shortcomings.

6.4 Future Work

In this subsection, we propose directions for future research to enhance our understanding of Greek-related issues in LLMs and extend the research to other non-English languages.

Further research could be conducted on the content of The Stack v2 to identify the total Greek corpus used during model training, especially focusing on Java files relevant to this study. Understanding the proportion of Greek data compared to English data in The Stack v2 could offer deeper insights into the tokenization challenges and performance disparities observed.

Additionally, while the hierarchical error taxonomy developed in this study is a valuable tool, its effectiveness could be enhanced with more iterations and refinements. Future work should aim to involve a more diverse range of languages and models to ensure the taxonomy's applicability across different contexts.

Finally, the exclusion of Greeklish data from this study, although intentional to maintain consistency, limits the applicability of this study to informal digital communication contexts. Future studies could explore the impact of Greeklish on LLM performance to provide a more comprehensive understanding of Greek language usage in coding environments.

Addressing this future research will help to refine the findings and contribute to the development of more inclusive and effective LLMs for non-English programming contexts.

7 CONCLUSION

This study has conducted an evaluation of Code LLMs, specifically StarCoder 2, in non-English programming contexts, focusing on the Greek language. Our research reveals significant disparities in Greek natural language usage within code files, highlighting the need for advanced techniques to enhance the multilingual capabilities of LLMs. By developing a hierarchical error taxonomy through an open coding approach, we identified and categorized common errors, providing a framework for improving model performance. The tokenization experiments further revealed critical insights into the impact of training data and tokenizer design on model effectiveness, particularly for underrepresented languages like Greek. The findings underscore the importance of using semantic similarity metrics over traditional metrics like BLEU and ROUGE for evaluating multilingual models. Our recommendations emphasize the need for more inclusive tokenizers and training datasets to bridge the resource gap between English and non-English contexts. Future work should focus on refining this taxonomy and expanding research to other underrepresented languages, ultimately contributing to the development of inclusive AI tools.

8 RESPONSIBLE RESEARCH

Conducting responsible research is important to ensure the integrity and ethical standards of our study. In our work, we have prioritized transparency, reproducibility, and ethical considerations.

By focusing on multiple languages through collaborative efforts, we aim to contribute to the inclusivity and accessibility of AI tools, addressing potential biases and limitations in current LLMs. Creating models that are accessible to all users regardless of language is crucial, particularly in educational settings where proficiency in English is limited. These tools can significantly aid in teaching coding to non-English speakers, making the technology more inclusive. Addressing language biases in LLMs ensures that people from diverse linguistic backgrounds can benefit equally, fostering a more equitable technological environment.

To ensure that our experiments are reproducible, we have made our methodology transparent and provided all necessary resources for replication. The code[12] used in our experiments is open-sourced and available on GitHub. The dataset[13] used as part of open coding approach can be accessed from Hugging Face, and the labeled data[14] is available in Google Drive.

Additionally, we did not train or finetune any models on the provided data, and excluded licenses encountered, thus adhering to ethical standards.

REFERENCES

[1] Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Buschhoff, Charvi Jain, Alexander Weber, Lena Jurkschat, Hammam Abdelwahab, Chelsea John, Pedro Ortiz Suarez, Malte Ostendorff, Samuel Weinbach, Rafet Sifa, Stefan Kesselheim, and Nicolas Flores-Herr. 2024. Tokenizer Choice For LLM Training: Negligible or Crucial?. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 3907–3924. https://aclanthology.org/2024.findings-naacl.247

[2] Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. 2022. A Framework for the Evaluation of Code Generation Models. Available online: https://github.com/bigcode-project/bigcodeevaluation-harness.

[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 e-prints*, Article arXiv:2207.14255 (July 2022), arXiv:2207.14255 pages. https://doi.org/10.48550/arXiv.2207.14255 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 (SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[5] Emanuela Boros, Maud Ehrmann, Matteo Romanello, Sven Najem-Meyer, and Frédéric Kaplan. 2024. Post-Correction of Historical Text Transcripts with Large Language Models: An Exploratory Study. In *Proceedings of the 8th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL 2024)*, Yuri Bizzoni, Stefania Degaetano-Ortlieb, Anna Kazantseva, and Stan Szpakowicz (Eds.). Association for Computational Linguistics, St. Julians, Malta, 133–159. https://aclanthology.org/2024.latechclfl-1.14

[6] Boxing Chen and Colin Cherry. 2014. A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU. In *WMT@ACL*. https://api.semanticscholar.org/CorpusID:7410732

[7] 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]

[8] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

[9] Angela Costa, Wang Ling, Tiago Luís, Rui Correia, and Luisa Coheur. 2015. A linguistically motivated taxonomy for Machine Translation error analysis. *Machine Translation* 29 (06 2015), 127–161. https://doi.org/10.1007/s10590-015-9169-0

[10] Gautier Dagan, Gabriel Synnaeve, and Baptiste Rozière. 2024. Getting the most out of your tokenizer for pre-training and domain adaptation. *arXiv e-prints*, Article arXiv:2402.01035 (Feb. 2024), arXiv:2402.01035 pages. https://doi.org/10.48550/arXiv.2402.01035 arXiv:2402.01035 [cs.CL]

[11] Hantian Ding, Varun Kumar, Yuchen Tian, Zijian Wang, Rob Kwiatkowski, Xiaopeng Li, Murali Krishna Ramanathan, Baishakhi Ray, Parminder Bhatia, and Sudipta Sengupta. 2023. A Static Evaluation of Code Completion by Large Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, Sunayana Sitaram, Beata Beigman Klebanov, and Jason D Williams (Eds.). Association for Computational Linguistics, Toronto, Canada, 347–360. https://doi.org/10.18653/v1/2023.acl-industry.34

[12] Julen Etxaniz, Gorka Azkune, Aitor Soroa, Oier Lacalle, and Mikel Artetxe. 2024. Do Multilingual Language Models Think Better in English?. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 550–564. https://aclanthology.org/2024.naacl-short.46

[13] Valentin Hartmann, Anshuman Suri, Vincent Bindschaedler, David Evans, Shruti Tople, and Robert West. 2023. SoK: Memorization in General-Purpose Large Language Models. arXiv:2310.18362 [cs.CL]

[14] A Huberman et al. 2014. Qualitative data analysis a methods sourcebook. (2014).

[15] Rudali Huidrom and Anya Belz. 2023. Towards a Consensus Taxonomy for Annotating Errors in Automatically Generated Text. In *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, Ruslan Mitkov and Galia Angelova (Eds.). INCOMA Ltd., Shoumen, Bulgaria, Varna, Bulgaria, 527–540. https://aclanthology.org/2023.ranlp-1.58

[16] Maliheh Izadi, Jonathan Katzy, Tim Van Dam, Marc Otten, Razvan Mihai Popescu, and Arie Van Deursen. 2024. Language Models for Code Completion: A Practical Evaluation. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[17] Shahedul Huq Khandkar. 2009. Open coding. *University of Calgary* 23, 2009 (2009), 2009.

[18] Konstantinos Kogkalidis, Stergios Chatzikyriakidis, Eirini Giannikouri, Vasiliki Katsouli, Christina Klironomou, Christina Koula, Dimitris Papadakis, Thelka Pasparaki, Erofili Psaltaki, Efthymia Sakellariou, and Charikleia Soupiona. 2024. OYXOY: A Modern NLP Test Suite for Modern Greek. In *Findings of the Association for Computational Linguistics: EACL 2024*, Yvette Graham and Matthew Purver (Eds.). Association for Computational Linguistics, St. Julian's, Malta, 311–322. https://aclanthology.org/2024.findings-eacl.21

[19] Fajri Koto, Tilman Beck, Zeerak Talat, Iryna Gurevych, and Timothy Baldwin. 2024. Zero-shot Sentiment Analysis in Low-Resource Languages Using a Multilingual Sentiment Lexicon. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, Yvette Graham and Matthew Purver (Eds.). Association for Computational Linguistics, St. Julian's, Malta, 298–320. https://aclanthology.org/2024.eacl-long.18

[20] Tomasz Limisiewicz, Jiří Balhar, and David Mareček. 2023. Tokenization Impacts Multilingual Language Modeling: Assessing Vocabulary Allocation and Overlap Across Languages. 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, 5661–5681. https://doi.org/10.18653/v1/2023.findings-acl.350

[21] Mario Linares-Vásquez, Gabriele Bavota, Michele Tufano, Kevin Moran, Massimiliano Di Penta, Christopher Vendome, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. 2017. Enabling mutation testing for Android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 233–244. https://doi.org/10.1145/3106237.3106275

[12] https://github.com/ploizides/LLM-of-Babel-Paris

[13] https://huggingface.co/parislo

[14] https://docs.google.com/spreadsheets/d/1mEqcouS4rPNqSDiX-IIgjm0gHj9ggudEzV-NgmNmnzs/edit?usp=sharing

[22] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. 2024. StarCoder 2 and The Stack v2: The Next Generation. *arXiv preprint arXiv:2402.19173* (2024).

[23] Junayed Mahmud, Fahim Faisal, Raihan Islam Arnob, Antonios Anastasopoulos, and Kevin Moran. 2021. Code to Comment Translation: A Comparative Study on Model Effectiveness & Errors. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, Royi Lachmy, Ziyu Yao, Greg Durrett, Milos Gligoric, Junyi Jessy Li, Ray Mooney, Graham Neubig, Yu Su, Huan Sun, and Reut Tsarfaty (Eds.). Association for Computational Linguistics, Online, 1–16. https://doi.org/10.18653/v1/2021.nlp4prog-1.1

[24] Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. 311–318.

[25] Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. OpenWebMath: An Open Dataset of High-Quality Mathematical Web Text. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*.

[26] Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How Multilingual is Multilingual BERT?. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 4996–5001.

[27] Nils Reimers and Iryna Gurevych. 2020. Making Monolingual Sentence Embeddings Multilingual using Knowledge Distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 4512–4525. https://doi.org/10.18653/v1/2020.emnlp-main.365

[28] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces (IUI '23)*. Association for Computing Machinery, New York, NY, USA, 491–514. https://doi.org/10.1145/3581641.3584037

[29] Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, Online, 3118–3135. https://doi.org/10.18653/v1/2021.acl-long.243

[30] Khetam Al Sharou and Lucia Specia. 2022. A Taxonomy and Study of Critical Errors in Machine Translation. In *Proceedings of the 23rd Annual Conference of the European Association for Machine Translation*, Helena Moniz, Lieve Macken, Andrew Rufener, Loïc Barrault, Marta R. Costa-jussà, Christophe Declercq, Maarit Koponen, Ellie Kemp, Spyridon Pilos, Mikel L. Forcada, Carolina Scarton, Joachim Van den Bogaert, Joke Daems, Arda Tezcan, Bram Vanroy, and Margot Fonteyne (Eds.). European Association for Machine Translation, Ghent, Belgium, 171–180. https://aclanthology.org/2022.eamt-1.20

[31] Liyan Tang, Igor Shalyminov, Amy Wong, Jon Burnsky, Jake Vincent, Yu'an Yang, Siffi Singh, Song Feng, Hwanjun Song, Hang Su, Lijia Sun, Yi Zhang, Saab Mansour, and Kathleen McKeown. 2024. TofuEval: Evaluating Hallucinations of LLMs on Topic-Focused Dialogue Summarization. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 4455–4480. https://aclanthology.org/2024.naacl-long.251

[32] Arda Tezcan, Véronique Hoste, and Lieve Macken. 2017. *SCATE taxonomy and corpus of machine translation errors.* https://doi.org/10.1163/9789004351790_012

[33] Shiqi Wang, Zheng Li, Haifeng Qian, Chenghao Yang, Zijian Wang, Mingyue Shang, Varun Kumar, Samson Tan, Baishakhi Ray, Parminder Bhatia, Ramesh Nallapati, Murali Krishna Ramanathan, Dan Roth, and Bing Xiang. 2023. ReCode: Robustness Evaluation of Code Generation Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 13818–13843. https://doi.org/10.18653/v1/2023.acl-long.773

[34] Yuanchi Zhang, Yile Wang, Zijun Liu, Shuo Wang, Xiaolong Wang, Peng Li, Maosong Sun, and Yang Liu. 2024. Enhancing Multilingual Capabilities of Large Language Models through Self-Distillation from Resource-Rich Languages. arXiv:2402.12204 [cs.CL]

[35] Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. 2023. CodeBERTScore: Evaluating Code Generation with Pretrained Models of Code. 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, 13921–13937. https://doi.org/10.18653/v1/2023.emnlp-main.859

[36] 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 (MAPS 2022)*. Association for Computing Machinery, New York, NY, USA, 21–29. https://doi.org/10.1145/3520312.3534864

A ERROR LABELS

All error labels used in the final iteration of labelling are listed below. Error label ID, label name and detailed inclusion criteria for each label used by the research team are included.

A.1 Syntax

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) ***(ST-IF1) Style Inconsistency***
   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) ***(ST-IF2) Omitted Identifier***
   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.2 Linguistic

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

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

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

(1) ***(LG-WL1) Undesired translations***
   Translations that are correct but undesired in the language because the words are seldomly used in that context.

(2) ***(LG-WL2) Incorrect language***
   The model predicts a comment (or significant part of it) in a language other than the target language.

Grammar. Language is correct, grammatical mistake was made.

(1) ***(LG-GR1) Plurality***
   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) ***(LG-GR2) Conjugation***
   Incorrect usage of conjugation rules

(3) ***(LG-GR3) Gendering***
   Incorrect gendering in case the language has gendered nouns

(4) ***(LG-GR4) Spelling***
   Incorrect spelling

(5) ***(LG-GR5) Capitalisation***
   Prediction capitalizes letters that grammatically is not correct to capitalize: e.g all capitals, every word begins with capital

(6) ***(LG-GR6) Cohesion***
   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."

A.3 Semantic

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) ***(SE-HA1) Misplaced Facts***
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) ***(SE-HA2) Contextual Discrepancy***
Hallucination not grounded in the provided context.
(3) ***(SE-HA3) Educated Guess***
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) ***(SE-CS1) Commented out code***
Code that resides in a code block.
(2) ***(SE-CS2) Code intended to run***
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.

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

A.4 Model Specific

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

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

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

Memorisation. Model recognised the code to some capacity

(1) ***(MS-ME1) Contains PII***
Personally identifiable information is included in the generated comment (fictional or did not occur in the original prompt)
(2) ***(MS-ME2) Contains URL***
URL for a file or repository is included
(3) ***(MS-ME3) Verbatim Memorization***
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) ***(MS-RE1) Pattern repetition***
Model generated a repeating pattern: eg. 1,2,3,4,5,6,
(2) ***(MS-RE2) Verbatim repetition***
Model generated verbatim repetition: eg. i am repeating i am repeating i am repeating

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

(MS-LT)Late Termination . 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.5 Miscellaneous

Anything that does not fall into any of the above categories

B USE OF CHATGPT

Throughout the creation of this report, ChatGPT was used to assist in different stages of the writing and formatting process. The AI provided valuable input on refining the language and ensuring clarity, coherence and conciseness, as well as the formatting of plots used in the analysis. Prompts used throughout the project are listed below:

- "Can you rewrite the following paragraph to be more concise and to the point, while adhering to conference paper use of language"
- "Given the code for the creation of 3 different Box plots, help me create the code that combines all 3 in one figure that shares the Y axis"
- "Can you provide me with phrases that I can use while explaining my results and graphs?"
- "In the following section what can be considered redundant content that could be removed to save space?"
- "Please review this text for any grammatical errors or inconsistencies."