ISBN: 978-628-96613-1-6. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2025.1.1.458

Evaluating Language Dependency in Large Language Models: A Study on Programming Queries in English and Spanish

1st Farman Ali Pirzado
School of Engineering and Sciences
Tecnológico de Monterrey, Mexico
Monterrey, Mexico
A00836551@tec.mx

2nd Awais Ahmed
School of Computer Science
China West Normal University
Nanchong, China
ahmedawais@cwnu.edu.cn

3rd Gerardo Ibarra-Vázquez
School of Engineering and Sciences
Tecnológico de Monterrey, Mexico
Monterrey, Mexico
gerardo.ibarra.v@tec.mx

4th Hugo Terashima-Marin
School of Engineering and Sciences
Tecnológico de Monterrey, Mexico
Monterrey, Mexico
terashima@tec.mx

***Abstract*—As the integration of Artificial Intelligence tools, such as large language models (LLMs), into computing education increases, understanding their impact on students' learning becomes crucial. According to recent research, LLMs perform well when processing input in the English language. Still, they struggle when processing input in other languages or inputs containing non-English syntax or symbols, such as different languages and programming queries. Therefore, this study evaluates whether programming queries, particularly code generation queries in Spanish, a widely spoken language other than English, present challenges similar to those in code generation tasks compared to English queries. By doing this, this study aims to identify accuracy differences in the code generated by LLMs (Codex and Copilot) for English and Spanish input on a set of programming problems sourced from LeetCode. The study compares the performance of LLMs on three complexity levels of tasks, including basic, medium, and advanced code generation tasks. The results show that both Codex and Copilot show a significant decline in accuracy for Spanish as compared to English, particularly as task complexity increases from basic to advanced level. The Codex shows a significant decline in accuracy for Spanish inputs (85%) compared to English (92%). Similarly, Copilot shows a significant increase in accuracy for English inputs (93%) compared to Spanish (87%), with higher error rates across syntax, runtime, and logical errors in both. By comparing the results across multiple languages, the findings show that LLMs perform better on English-language inputs for code generation. Additionally, it demonstrated that Copilot also has superior adaptability and reliability in handling multilingual programming tasks compared to Codex. These results serve as a foundation and further emphasize the need for improvement in multilingual capabilities, as well as the language-dependent limitations of LLMs.**

***Index Terms*—Large Language Models, Multilingual Queries, Code Generation, Spanish, English.**

I. Introduction

The emergence of Large Language Models (LLMs), such as ChatGPT, Codex, Copilot, Palm, etc., has brought significant advancement to programming education, enabling students to receive assistance in code Debugging, code generation, and programming error message explanation [1], [2]. Undoubtedly, these models are considered to perform best when processing input in English due to their training and optimization, and they are highly resourced [3]. In addition, these models encounter multiple challenges when programming prompts due to input containing code snippets and programming queries containing special characters [4]. It has been reported that when prompting these models containing non-English input, such as coding structure, the outputs frequently include several mistakes, reflecting limitations in LLMs' ability to handle such inputs effectively [1]. However, students also seek more interactive and better explanations of errors, including the ability to ask clarifying questions about mistakes in their code and receive conversational responses from AI coding assistants [5]. No doubt, LLMs show potential for enhancing error explanation quality, such as improving compile-time error descriptions, but further advancements are needed to refine these capabilities, particularly for runtime error explanations [4]. Additionally, it is being discussed that current debugging capabilities are hindered by LLMs' inability to learn from past mistakes or adapt their responses based on user feedback, which limits their effectiveness in several problem-solving scenarios [6]. Although promising for improving programming error messages, significant work remains to be done in developing methods that enhance the quality and contextual relevance of LLM-generated responses [7].

The growing discussion about adopting LLMs in educational settings highlights the need to evaluate their effectiveness in generating accurate content and offering assistance in languages other than English. This is particularly relevant for non-English programming query contexts, where linguistic diversity significantly shapes learning experiences [3]. There-

fore, the literature highlights multiple gaps in LLMs' generalization and reasoning capabilities when faced with diverse, unseen inputs. Prior research emphasizes the importance of examining these capabilities to uncover potential biases, such as those related to multilingual inputs, particularly in handling queries in languages other than English [8].

Such limitations in LLMs pose a significant challenge for students and developers attempting to interact with these tools in non-English languages, particularly in programming contexts where coding syntax is already part of the prompt. As discussed earlier, programming inputs are inherently complex due to their structured nature, which includes code snippets and symbols. When combined with non-English prompts, this creates a dual challenge for LLMs to consider, highlighting their difficulty in processing inputs such as comments, variable names, and function descriptions with intricate, complex code syntax [9], [10]. These intermingled challenges underline the need for advancements in LLMs to handle the unique demands of multilingual and programming-specific contexts.

Therefore, this research investigates the performance of LLMs, with a primary focus on Codex and Copilot, when handling programming queries in non-English languages, specifically Spanish, compared to English. While LLMs are generally optimized for English-language prompts, their ability to generate accurate and efficient code from non-English instructions remains underexplored. By comparing responses to equivalent programming tasks in both languages, this study aims to identify potential performance gaps and assess their impact on equity and accessibility in programming education. The findings could inform LLM development, guiding improvements that better support non-English programming input and provide educators with practical insights for incorporating these tools in diverse language contexts.

To guide this investigation, we focus on the following research questions:

- **RQ1:** How do LLMs perform on programming tasks when inputs are provided in Spanish compared to English?
- **RQ2:** What is the difference between code accuracy, i.e., syntax, logical, and runtime errors, in programming queries when prompted in Spanish compared to English?
- **RQ3:** How does the accuracy of LLMs (Codex and Copilot) differ between English and Spanish programming queries across various levels of task complexity (basic, medium, advanced)?

II. Background and Related work

Recently, the performance of LLMs in responding to programming-related prompts has been a growing area of research, particularly their ability to produce correct output from input containing coding snippets [11]–[13]. A recent study investigated OpenAI's Codex and GPT-3.5 models, focusing on their ability to interpret students' programming queries within an online programming course at Aalto University in Finland. The study analyzed help requests and code samples, revealing that LLMs often struggle with inputs containing data beyond standard English letters, such as code snippets or symbols. These findings highlight the necessity of specially designed LLMs to assist computer science students in programming education [2]. In addition, evaluation of OpenAI's GPT models has also shown that while these models generate and explain programming code very well, they perform significantly better on queries framed in everyday English than those involving code snippets or symbolic data. According to a study that evaluated Python multiple-choice questions (MCQs), LLMs perform better on plain English language queries compared to prompts involving coding syntax or special symbols [3]. Also, MCQz with coding snippets in the questions have been noticed as less successfully answered than those written entirely in English, and debugging tasks involving code inputs often lead to difficulties for LLMs [2], [3], [14]. Another similar study found that these models face limitations when processing coding prompts or questions containing special symbols, often resulting in inaccurate or incomplete outputs, which further emphasizes the dependency of LLMs on language and input structure [1], [14].

Although interacting with LLMs in English doesn't seem to have a significant impact on non-English users, other factors may contribute to a less inclusive learning environment. For instance, these students often report incredible self-doubt and hesitation in seeking assistance from LLMs in non-native languages [15], [16]. Conversely, many non-native English-speaking students find learning programming in their native language more comfortable and believe it enhances their overall experience [17]. However, native-language instruction does not significantly influence learning outcomes for non-native English-speaking students [15], [18]. Recent studies have also highlighted the capability of tools like ChatGPT-3.5 to generate programming problems in various languages, including Tamil, Spanish, and Vietnamese [19].

A recent study examines how generative AI tools like ChatGPT can support programming education for non-native English speakers. By solving Prompt Problems in their native languages, focusing on three languages (Arabic, Chinese, and Portuguese), students explored multilingual interactions with AI. Portuguese and Chinese students achieved high success rates (over 90% and 63%, respectively), while Arabic speakers faced challenges (27% success) due to limited AI training data. Although native-language prompts improved expressiveness, English prompts often delivered better results, highlighting the dominance of English in programming syntax and AI performance [20]. This limitation can ultimately cause students various problems, such as students often face challenges in crafting effective prompts for accurate code generation, especially when non-English content is involved [2]. It would be difficult for students to interact with these tools in their native languages, particularly when it comes to programming queries. Therefore, this study emphasizes the critical need to evaluate and address the performance of LLMs when processing programming queries in languages other than English. The research highlights the challenges of LLMs in handling non-English languages by analyzing the accuracy and correctness

of code generated from Spanish-written programming queries and comparing it with English inputs. Given that English is a high-resource language where LLMs generally perform well, this comparison serves as a benchmark to uncover specific limitations in LLMs' multilingual capabilities.

III. Methodology

This section outlines the systematic approach used to evaluate the performance of LLMs (Codex vs. Copilot) in handling programming queries in English and Spanish. Several studies chose these platforms for similar analysis because they are specially designed tools for programming and are fine-tuned on large amounts of programming data [21]–[24]. To ensure a comprehensive analysis, the following subsection details the experimental setup, including data preparation, task design, evaluation metrics, and calculations of performance metrics. Figure 1 depicts the proposed methodology diagram. The methodology consists of three components: the input module, the code generation and accuracy analysis module.

First, a set of code generation problems was obtained from LeetCode, which were initially written in English. Each code generation query was then carefully translated into Spanish. The input module ensures semantic equivalence between the English and translated Spanish versions of a set of code generation questions sourced from LeetCode. Next, in the code generation module, these code generation queries were prompted to two different LLMs (Codex and Copilot) for code generation.

At last, in the accuracy analysis module, the resulting outputs were evaluated and compared to examine variations in the functional correctness of the code generated for both inputs. To assess functional accuracy, the generated code was tested on LeetCode, with a focus on syntax, runtime, and logical errors. Leetcode was chosen due to its standardized problems and automated feedback on syntax, runtime, and correctness, making it a widely accepted benchmark for evaluating LLM-generated code [25]–[27].

A. Data Preparation

Dataset curation were essential part of the methodology, to achieve designed objectives, question sets were taken randomly from LeetCode, a commonly used dataset for assessing the performance of LLMs on programming inputs. Various studies have used this dataset to evaluate the performance of LLMs in handling programming queries [26], [27], [27], [28]. To conduct our experiment, we collected a set of 100 tasks, divided into three difficulty levels: 40 basic-level, 30 medium-level, and 30 advanced-level code generation queries. To prepare a parallel dataset in Spanish, we used Google Translator to translate the complete dataset into Spanish, ensuring that the technical integrity of the problems was maintained during the translation process. Additionally, one of our Native Spanish authors was assigned to verify the translation for correct interpretation.

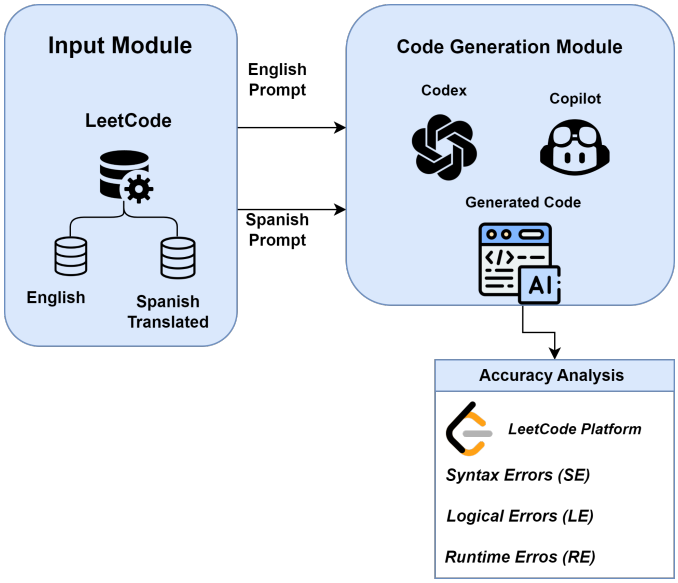


Fig. 1: Methodology.

B. Prompting and Code Generation

For each task, two prompts, one in English and one in Spanish, are created, and a task ID has been assigned, including the task description as shown in Figure 2. LLMs such as OpenAI Codex, which support both languages, are used, with Python as the programming language. The query sends requests to the Codex and Copilot APIs, which return generated code in both Spanish and English. The solutions are manually executed on the Leetcode website to report and analyze the errors. Only accuracy metrics are evaluated through a manual assessment.

```
# Define tasks in English and Spanish
tasks = [
    {
        "task_id": 1,
        "english_prompt": "Write a function to calculate the factorial of a number.",
        "spanish_prompt": "Escribe una función para calcular el factorial de un número."
    },
    {
        "task_id": 2,
        "english_prompt": "Given an array of integers, return indices of the two numbers such that
        they add up to a specific target. You may assume that each input would have exactly one solution,
        and you may not use the same element twice.",
        "spanish_prompt": "Dada una matriz de enteros, devuelve los índices de los dos números que suman un
        objetivo específico. Puedes asumir que cada entrada tendrá exactamente una solución y no puedes usar
        el mismo elemento dos veces."
    },
```

Fig. 2: Prompt Example.

C. Evaluation Metrics

The accuracy metric assesses an LLM's ability to generate error-free code that meets the requirements and passes all unit tests. It encompasses various errors that require developers or students to spend additional time and effort correcting them. LLMs that produce precise code with fewer errors help reduce development time and enhance student-team productivity. The generated code is evaluated based on accuracy, checking for syntax, runtime, and logical errors. These metrics are consistent with those used in prior studies analyzing LLM-based code generation and multilingual prompt handling [1], [29], [30] Three metrics are used in this study to assess the accuracy of the generated code.

- **Syntax Errors (SE):** These errors occur when the code does not follow the grammatical rules of the programming language. These errors typically prevent the program from running. They often involve incorrect punctuation, missing keywords, or improper structure, such as unmatched parentheses or semicolons.
- **Runtime Errors (RE):** Runtime errors happen while the program is executing. These errors are responsible for crashes or unexpected behavior during the runtime of the generated code. Some examples are dividing by zero and accessing an undefined variable.
- **Logical Errors (LE):** These errors occur when the code runs without crashing but produces incorrect output, most of the time caused by faulty logic or reasoning behind the code. Some examples include using the wrong formula, misplacing a loop condition, or mismanaging data, which can lead to logic errors in the generated code.

The formulas utilized in the calculations are provided below.

1) **Accuracy:** Accuracy was determined as the percentage of correctly solved tasks:

$$\text{Accuracy} = \frac{\text{Number of Correct Solutions}}{\text{Total Number of Tasks}} \times 100$$

2) **Error Rates:** For each error type, the rate was calculated to understand the frequency of syntax, runtime, and logical errors:

$$\text{Error Rate (Type X)} = \sum_{i=1}^{n} \text{ErrorCount}_i$$

3) **Performance Difference:** The deviation in performance between English and Spanish was calculated as:

$$\text{Difference (Metric)} = \text{ER of English} - \text{ER of Spanish}$$

IV. Results Analysis

To analyze the effectiveness of solving the same programming question in different languages, we conducted an experiment using the regular expression matching problem. This problem requires a function that matches an input string against a pattern. The problem supports using special characters, including* (matches any character) and * (matches zero or more occurrences of the preceding element). Figure 3 depicts the problem definition. An example solution pair for both English and Spanish input is summarized in a sequence of Figures, from Figure 4 to Figure 6, showcasing the problem-solving approach in English and Spanish, the implementation process, and the results.

The first step involves preparing the input for the LLMs. Figure 4 presents both English and Spanish versions of the same programming problem used as input, illustrating how the problem statement, examples, and constraints were provided in both languages. Initially, we presented the problem in English, as shown in Figure 4a, which is commonly used in programming contexts. This version was straightforward for both students and developers to process. In contrast,

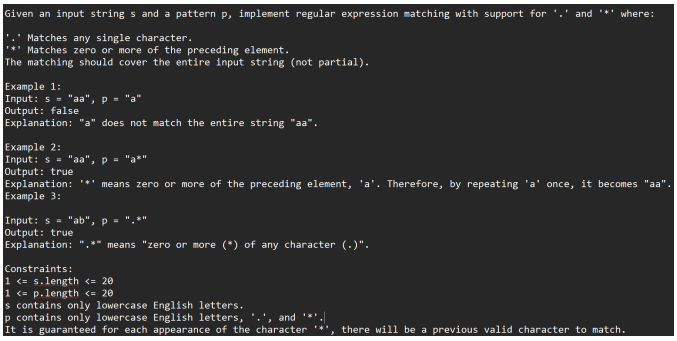


Fig. 3: A Snapshot of LeetCode selected question for result discussion.

the Spanish version presented in Figure 4b uses equivalent translation, allowing us to analyze language-related variations in model behavior. The second step focuses on code generation. Figure 5 shows how Codex responds to the same problem in both languages. For the English input, Codex generates syntactically correct and contextually appropriate code presented in Figure 5a. However, when given the Spanish version of the prompt, the output changes notably and lacks correctness, as seen in Figure 5b.

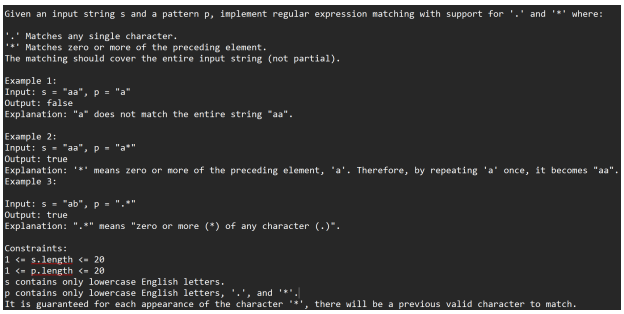
In the final step, we evaluated the functional correctness of the generated code by submitting it to LeetCode. Figure 6 summarizes these results. The English-generated solution was accepted by the platform shown in Figure 6a, confirming its syntactic and functional validity. In contrast, the Spanish-generated code produced a syntax error during submission, presented in Figure 6b. This reflects a limitation in the model's ability to accurately parse and respond to prompts in Spanish.

These examples show that while LLMs like Codex and Copilot handle English prompts effectively, their performance declines with Spanish inputs. This suggests current models may not generalize well across languages, raising concerns about their reliability in multilingual educational settings.

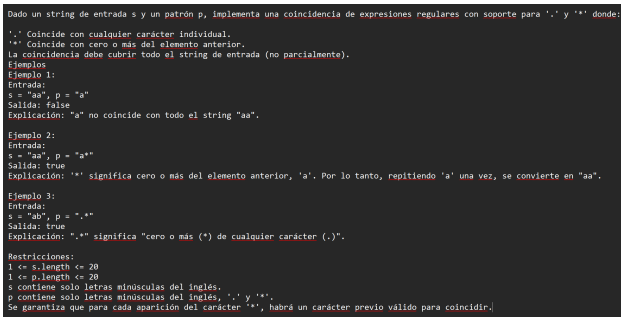
A. Key Observations

The experiment highlights the adaptability of problem-solving across languages, emphasizing that while the underlying logic remains the same, how a problem is described and understood can influence the implementation process. For example, it took considerable care to translate technical jargon while maintaining the intended meaning in Spanish. Despite these difficulties, the findings show that solutions can be successfully modified for the multilingual situation, with accurate translation and a detailed problem definition. Furthermore, Table I shows the performance of Codex and Copilot on English and Spanish programming inputs. The results presented in the table provide a detailed comparison of the performance of two different LLMs in handling English and Spanish programming inputs across three task levels: basic, medium, and advanced.

The results show that the performance of both Codex and Copilot is notably stronger with English inputs compared to



(a) Input in English



(b) Input in Spanish

Fig. 4: Two variants of inputs are adopted in our work such as sub-figure a)Showing English input, while b)Depicting Spanish Input.

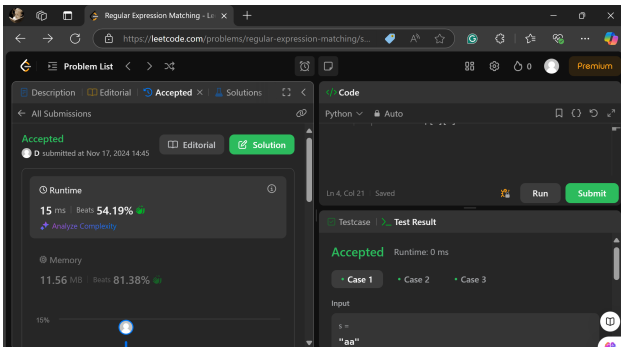
```python
class Solution(object):
    def isMatch(self, s, p):
        """
        :type s: str
        :type p: str
        :rtype: bool
        """
        # Create a DP table with dimensions (len(s) + 1) x (len(p) + 1)
        m, n = len(s), len(p)
        dp = [[False] * (n + 1) for _ in range(m + 1)]

        # Base case: empty string matches empty pattern
        dp[0][0] = True

        # Handle patterns with '*' that can match zero preceding elements
        for j in range(2, n + 1):
            if p[j - 1] == '*':
                dp[0][j] = dp[0][j - 2]

        # Fill the DP table
        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if p[j - 1] == s[i - 1] or p[j - 1] == '.':
                    # Direct match or '.' wildcard
                    dp[i][j] = dp[i - 1][j - 1]
                elif p[j - 1] == '*':
                    # '*' matches zero or more of the preceding character
                    dp[i][j] = dp[i][j - 2] or (dp[i - 1][j] and (s[i - 1] == p[j - 2] or p[j - 2] == '.'))

        return dp[m][n]
```

(a) Codex generated solution code using English input on LeetCode.

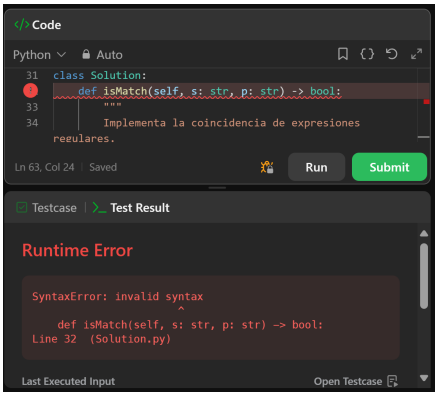
```python
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        """
        Implementa la coincidencia de expresiones regulares.
        :param s: string de entrada
        :param p: patrón con '.' y '*'
        :return: True si hay coincidencia, False en caso contrario
        """
        m, n = len(s), len(p)
        # Crear tabla DP con dimensiones (m + 1) x (n + 1)
        dp = [[False] * (n + 1) for _ in range(m + 1)]
        # Caso base: string vacío coincide con patrón vacío
        dp[0][0] = True
        # Manejar patrones con '*'
        for j in range(2, n + 1):
            if p[j - 1] == '*':
                dp[0][j] = dp[0][j - 2]

        # Rellenar la tabla DP
        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if p[j - 1] == s[i - 1] or p[j - 1] == '.':
                    # Coincidencia directa o '.'
                    dp[i][j] = dp[i - 1][j - 1]
                elif p[j - 1] == '*':
                    # '*' puede coincidir con cero o más del carácter anterior
                    dp[i][j] = dp[i][j - 2] or (dp[i - 1][j] and (s[i - 1] == p[j - 2] or p[j - 2] == '.'))

        # Devolver el resultado para el string completo y el patrón completo
        return dp[m][n]
```

(b) Codex generated solution code using Spanish input on LeetCode.

Fig. 5: Snapshots of Code-generated solutions based on English and Spanish versions of the same LeetCode problem.



(a) A snapshot of accepted submission using Codex-generated solution from English input on LeetCode.



(b) A snapshot showing a syntax error when submitting the Codex-generated solution from Spanish input to LeetCode.

Fig. 6: Decision: Comparison of results using Codex-generated solutions: (a) from English input (successful), and (b) from Spanish input (syntax error).

Spanish across all levels of problem complexity. Codex shows a 7% drop in accuracy when shifting from English to Spanish, while Copilot exhibits a slightly smaller decline of 6%. Additionally, both models produce more syntax, runtime, and logical errors when processing Spanish inputs. For instance, syntax errors increase by two on average in Spanish, and logical errors rise by up to 1.67 for Codex and 1.00 for Copilot. Therefore, it can be argued that both LLMs are better optimized for English, and their reliability decreases when handling non-English programming prompts.

Additionally, as problem complexity increased from basic to advanced, both Codex and Copilot showed a consistent decline

TABLE I: Performance Comparison of LLMs (Codex vs. Copilot) on English and Spanish Inputs with problem complexity of Basic, Medium, and Advanced.

Task Details	Metric	Codex (English)	Codex (Spanish)	Perf Diff (Codex)	Copilot (English)	Copilot (Spanish)	Perf Diff (Copilot)
Basic: 40 Problems	Accuracy	95%	88%	-7%	96%	90%	-6%
	Syntax Errors	2	4	+2	1	3	+2
	Runtime Errors	1	3	+2	1	2	+1
	Logical Errors	1	2	+1	1	1	0
Medium: 30 Problems	Accuracy	90%	83%	-7%	92%	86%	-6%
	Syntax Errors	3	6	+3	2	4	+2
	Runtime Errors	2	4	+2	1	3	+2
	Logical Errors	1	3	+2	1	2	+1
Advanced: 30 Problems	Accuracy	88%	80%	-8%	90%	83%	-7%
	Syntax Errors	4	7	+3	3	5	+2
	Runtime Errors	3	5	+2	2	4	+2
	Logical Errors	2	4	+2	2	3	+1
Overall: 100 Problems	Accuracy	92%	85%	-7%	93%	87%	-6%
	Syntax Errors	3	5	+2	2	4	+2
	Runtime Errors	2	4	+2	1	3	+2
	Logical Errors	1.33	3	+1.67	1.00	2.00	+1.00

in performance. Accuracy drops progressively across complexity levels for example, Codex's accuracy in English decreases from 95% (Basic) to 88% (Advanced), while Copilot's drops from 96% to 90%. This trend is also noticed in Spanish, where performance declines are even more pronounced. Syntax, runtime, and logical errors all become more frequent with increasing task difficulty. The findings underscore that both models struggle more with more complex programming tasks, and this is exacerbated when the input language is not English.

Finally, when comparing all 100 problems across complexity levels, Copilot slightly outperforms Codex in terms of overall accuracy and fewer logical errors. Copilot achieves 93% accuracy with English inputs and 87% with Spanish, whereas Codex scores 92% and 85%, respectively. Both models exhibit consistent patterns of higher error rates in Spanish, with increases of +2 in syntax and runtime errors, and over +1 in logical errors. These results reinforce the conclusion that Copilot maintains a modest but consistent edge in performance and is slightly more robust than Codex across both languages and task types.

In conclusion, both Codex and Copilot demonstrate stronger capability in code generation; however, their performance declines when handling queries in Spanish compared to English. Codex experiences a more significant reduction in overall accuracy and higher error rates. Particularly as task complexity increases, indicating challenges in adapting to non-English mode generation inputs. In contrast, Copilot exhibits greater robustness and adaptability to multilingual inputs. However, it too shows a noticeable performance gap between English and Spanish queries, although smaller than that of the Codex. Given its lower error rates across categories and reduced overall performance rate, Copilot emerges as a more reliable solution for multilingual programming tasks. These findings highlight the need for further advancements in LLMs to effectively address the dual challenges of language diversity and programming complexity, ensuring consistent performance in a multilingual programming education context.

B. Students Learning Implications

Firstly, the drop in LLMs' performance on code generation input into multilingual context has been observed, highlighting the need for educators to be aware of language barriers when incorporating these tools into programming education, which is the current focus of the research [31].In addition, the study also highlights the importance of incorporating multilingual LLMs, which can assist students in programming courses, emphasizing their potential to support diverse student populations. However, this requires targeted efforts to fine-tune LLMs for non-English languages, ensuring equitable access to these tools in programming education [2].

Overall, it is recommended that students who interact with LLMs in a language other than their native language should consider providing additional materials, such as glossaries for technical terms, to bridge potential comprehension gaps caused by linguistic nuances. In addition, educators could also design assignments that encourage students to evaluate AI-generated code, thereby improving their ability to analyze output generated by these systems. It will also enhance their programming skills using modern tools and reduce their over-reliance on LLM-generated outputs. Furthermore, there is a need to train students to formulate questions in their native languages, which can effectively prompt LLMs and improve their interaction with them, ultimately enhancing their overall learning experience. This is the ultimate goal of integrating these tools into traditional education [32].

Also, by following reflective practices, such as comparing LLM-generated code across languages and assessing its correctness, students can better understand the limitations of LLMs and develop critical thinking and debugging skills. In the end, the study's findings also recommended future curriculum development that focuses on multilingual programming education and the ethical integration of AI tools, an essential focus of several recent studies [33]. Undoubtedly, equipping students with the skills to navigate language-specific challenges and leverage AI tools effectively can promote inclusivity and prepare them for diverse, globalized programming and computing education environments.

V. Discussion

This study examined how well LLMs (Codex and Copilot) perform when generating code in response to input queries in a multilingual environment, specifically comparing English and Spanish. By comparing the differences in accuracy, focusing on the functional correctness (including syntax, logical, and runtime errors) of the generated code for both inputs, this study sheds light on LLMs' challenges when dealing with programming inputs, particularly in Spanish. The findings reported an increase in syntax and runtime errors in generated code for Spanish input, which is one of the study's most notable findings. Similarly, syntax errors were consistently higher for Spanish queries, with two additional errors observed compared to English at every task level in both LLMs. In addition, the same pattern has been noticed in the increase of runtime errors in the code generated via Spanish queries, resulting in more failures than their English counterparts.

These findings suggest that these tools may have been primarily trained on English-language programming data, which could explain their superior performance with English inputs. LLMs may struggle to interpret and generate code effectively when encountering the linguistic nuances and structure of Spanish programming queries, particularly in syntax and error handling. The findings provide valuable insights into the language-dependent nature of LLMs, particularly in coding tasks, offering a foundation for future research on enhancing LLM performance for multilingual use cases.

Regarding RQ-1, the performance comparison of Codex and Copilot on programming inputs in English and Spanish reveals a noticeable decline in performance when the queries are presented in Spanish. Both LLMs exhibited a reduction in overall accuracy when working with Spanish inputs, with Codex showing a 7% drop in accuracy and Copilot showing a 6% decline across various task complexities (basic, medium, and advanced). This suggests that LLMs are better optimized for English, likely due to the predominance of English-language datasets used in their training. The performance drop for Spanish inputs is consistent across all error categories, including syntax, runtime, and logical errors, indicating a challenge in understanding or generating code for non-English queries.

These findings align with those of [26], who reported that open-source models tend to outperform ChatGPT in text-to-code generation, particularly for specific languages. However, our study provides a more nuanced perspective by highlighting the challenges LLMs face when handling non-English inputs, which may require further fine-tuning or adjustments to improve their multilingual capabilities. The results related to RQ-2 suggest that both models, Codex and Copilot, tended to make more errors when responding to Spanish-language queries. Specifically, syntax errors were higher in Spanish, with Codex and Copilot experiencing an increase in the number of syntax errors when programming queries in Spanish. For instance, Codex had four syntax errors in Spanish compared to only two in English. Copilot showed similar behavior, rising from two syntax errors in English to four in Spanish.

Finally, considering RQ-3, our analysis also echoes [27], who found that LLM-generated code performed well for simpler tasks, but struggled with more complex problems. It has been noticed that, according to our results across all task complexities, basic, medium, and advanced, the accuracy of Codex and Copilot dropped when queries were provided in Spanish. A decline in accuracy was observed in both models, but it was more substantial for Codex. Specifically, Codex's accuracy dropped by 7% for basic and medium tasks and 8% for advanced tasks, while Copilot's accuracy dropped by only 6% across all complexities.

In summary, the study highlights the need to refine LLMs to handle non-English programming languages, especially Spanish, better, as both models demonstrate challenges in parsing and generating code in this language. While Copilot shows slightly better performance, as task complexity increases, the results underscore the importance of enhancing multilingual datasets and error-handling mechanisms to address linguistic nuances and improve LLMs' effectiveness in diverse linguistic contexts.

VI. Conclusion and Future Work

This study assessed the language-dependent performance of LLMs, specifically Codex and Copilot, in generating code from code generation input queries in English and Spanish. The results show performance disparity favoring English, with Codex showing a significant decline in accuracy for Spanish input, especially as task complexity increases. However, Copilot demonstrated better adaptability to multilingual prompts.

The possibilities for future work include assessing additional models, along with further metrics such as quality and performance metrics of LLMs, to conduct a more comprehensive analysis. One worth mentioning is the future work of incorporating an original Spanish dataset instead of a translated version, which would further help models to understand prompts effectively. In addition, comparing code generated from Spanish and English queries with code generated by humans could also yield valuable insights. Lastly, analyzing the limitations of LLMs in other languages, such as Chinese and Arabic, and developing techniques to optimize LLMs for multilingual programming contexts will be essential.

References

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[2] A. Hellas, J. Leinonen, S. Sarsa, C. Koutcheme, L. Kujanpää, and J. Sorva, "Exploring the responses of large language models to beginner programmers' help requests," *arXiv preprint arXiv:2306.05715*, 2023.

[3] M. S. Orenstrakh, O. Karnalim, C. A. Suarez, and M. Liut, "Detecting llm-generated text in computing education: A comparative study for chatgpt cases," *arXiv preprint arXiv:2307.07411*, 2023.

[4] A. Taylor, A. Vassar, J. Renzella, and H. Pearce, "Integrating large language models into the debugging c compiler for generating contextual error explanations," *arXiv preprint arXiv:2308.11873*, 2023.

[5] B. Kimmel, A. Geisert, L. Yaro, B. Gipson, T. Hotchkiss, S. Osae-Asante, H. Vaught, G. Wininger, and C. Yamaguchi, "Enhancing programming error messages in real time with generative ai," *arXiv preprint arXiv:2402.08072*, 2024.

[6] F. A. Sakib, S. H. Khan, and A. Karim, "Extending the frontier of chatgpt: Code generation and debugging," *arXiv preprint arXiv:2307.08260*, 2023.

[7] J. Leinonen, A. Hellas, S. Sarsa, B. Reeves, P. Denny, J. Prather, and B. A. Becker, "Using large language models to enhance programming error messages," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 2023, pp. 563–569.

[8] H. Tian, W. Lu, T. O. Li, X. Tang, S.-C. Cheung, J. Klein, and T. F. Bissyandé, "Is chatgpt the ultimate programming assistant–how far is it?" *arXiv preprint arXiv:2304.11938*, 2023.

[9] M. Grandury, "The# somos600m project: Generating nlp resources that represent the diversity of the languages from latam, the caribbean, and spain," *arXiv preprint arXiv:2407.17479*, 2024.

[10] I. Plaza, N. Melero, C. del Pozo, J. Conde, P. Reviriego, M. Mayor-Rocher, and M. Grandury, "Spanish and llm benchmarks: is mmlu lost in translation?" *arXiv preprint arXiv:2406.17789*, 2024.

[11] F. A. Pirzado, A. Ahmed, R. A. Mendoza-Urdiales, and H. Terashima-Marin, "Navigating the pitfalls: Analyzing the behavior of llms as a coding assistant for computer science students-a systematic review of the literature," *IEEE Access*, 2024.

[12] D. Cambaz and X. Zhang, "Use of ai-driven code generation models in teaching and learning programming: a systematic literature review," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 2024, pp. 172–178.

[13] N. Raihan, M. L. Siddiq, J. C. Santos, and M. Zampieri, "Large language models in computer science education: A systematic literature review," in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1*, 2025, pp. 938–944.

[14] J. Savelka, A. Agarwal, C. Bogart, and M. Sakr, "Large language models (gpt) struggle to answer multiple-choice questions about code," *arXiv preprint arXiv:2303.08033*, 2023.

[15] V. Agarwal, Y. Chuengsatiansup, E. Kim, Y. LYu, and A. G. Soosai Raj, "An analysis of stress and sense of belonging among native and non-native english speakers learning computer science," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1*, 2022, pp. 376–382.

[16] I. V. Molina, A. Montalvo, B. Ochoa, P. Denny, and L. Porter, "Leveraging llm tutoring systems for non-native english speakers in introductory cs courses," *arXiv preprint arXiv:2411.02725*, 2024.

[17] A. G. S. Raj, K. Ketsuriyonk, J. M. Patel, and R. Halverson, "What do students feel about learning programming using both english and their native language?" in *2017 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*. IEEE, 2017, pp. 1–8.

[18] A. G. Soosai Raj, K. Ketsuriyonk, J. M. Patel, and R. Halverson, "Does native language play a role in learning a programming language?" in *Proceedings of the 49th ACM technical symposium on computer science education*, 2018, pp. 417–422.

[19] M. Jordan, K. Ly, and A. G. Soosai Raj, "Need a programming exercise generated in your native language? chatgpt's got your back: Automatic generation of non-english programming exercises using openai gpt-3.5," in *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*, 2024, pp. 618–624.

[20] J. Prather, B. N. Reeves, P. Denny, J. Leinonen, S. MacNeil, A. Luxton-Reilly, J. Orvalho, A. Alipour, A. Alfageeh, T. Amarouche *et al.*, "Breaking the programming language barrier: Multilingual prompting to empower non-native english learners," *arXiv preprint arXiv:2412.12800*, 2024.

[21] F. Deriba, I. T. Sanusi, O. O Campbell, and S. S. Oyelere, "Computer programming education in the age of generative ai: Insights from empirical research," 2024.

[22] Z. Fan, X. Gao, A. Roychoudhury, and S. H. Tan, "Improving automatically generated code from codex via automated program repair," *arXiv preprint arXiv:2205.10583*, 2022.

[23] E. Fajkovic and E. Rundberg, "The impact of ai-generated code on web development: A comparative study of chatgpt and github copilot," 2023.

[24] B. Yetiştiren, I. Özsoy, M. Ayerdem, and E. Tüzün, "Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt," *arXiv preprint arXiv:2304.10778*, 2023.

[25] W. Hou and Z. Ji, "Comparing large language models and human programmers for generating programming code," *Advanced Science*, vol. 12, no. 8, p. 2412279, 2025.

[26] L. Mayer, C. Heumann, and M. Aßenmacher, "Can opensource beat chatgpt?–a comparative study of large language models for text-to-code generation," *arXiv preprint arXiv:2409.04164*, 2024.

[27] T. Coignion, C. Quinton, and R. Rouvoy, "A performance study of llm-generated code on leetcode," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, 2024, pp. 79–89.

[28] U. Lai, "Chatgpt's code suggestion accuracy evaluation," 2024.

[29] D. Kang, Z. L. He, and D. Hendrycks, "Code switching: Evaluating code generation capabilities across natural languages," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

[30] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le *et al.*, "Program synthesis with large language models," *arXiv preprint arXiv:2108.07732*, 2021.

[31] S. Elbanna and L. Armstrong, "Exploring the integration of chatgpt in education: adapting for the future," *Management & Sustainability: An Arab Review*, vol. 3, no. 1, pp. 16–29, 2024.

[32] K. Alshahrani and R. J. Qureshi, "Review the prospects and obstacles of ai-enhanced learning environments: The role of chatgpt in education," *International Journal of Modern Education and Computer Science*, 2024.

[33] L. Leng, "Challenge, integration, and change: Chatgpt and future anatomical education," *Medical Education Online*, vol. 29, no. 1, p. 2304973, 2024.