Research Report

Large Language Model for Code Translation

Students names:
Manal Alkhudhairy:451214899
Raghad ALmisned: 451213970
Eman ALharbi:451213973
Shahad ALmisned:451213971

Submission date: 12/2/2023

TABLE OF CONTENT

1. Introduction

For a long time, the programming language and software engineering (PLSE) community has struggled with source-to-source translation. Converting code manually from one programming language to another is an intricate task that requires meticulous rewriting and restructuring of the code to align with the syntax, semantics, and idiomatic conventions of the target language. Not only is this process time-consuming and labor-intensive, but it also has the potential for bugs and inconsistencies if not executed with precision.

Manual code conversion is a major challenge due to the differences between programming language paradigms and features. Each programming language has its own syntax rules, data types, control structures, and libraries that may not have direct equivalents in the target language. Consequently, a deep understanding of both the source and target languages, coupled with expertise in the involved programming paradigms, is crucial.

Moreover, manual code conversion is often accompanied by the translation of intricate algorithms and data structures. Ensuring the preservation of original functionality and performance characteristics while adapting the code to the target language proves to be a daunting task, demanding careful consideration of language-specific nuances, optimizations, and potential pitfalls.

The challenges of manual code conversion are further compounded by human errors. Bugs and unexpected behavior in the translated code can be caused by mistakes in syntax, logic, or comprehension of the original code. Debugging and troubleshooting such issues requires substantial expertise in both languages, which adds to the overall complexity.

To address these challenges, developers and researchers have explored automated code translation techniques. Automation through tools, frameworks, and machine learning approaches is used to reduce the likelihood of human errors and inconsistencies while also improving the efficiency and accuracy of code conversion.

The conversion of source code from one high-level language such as C to another, such as Python, is seamless with the use of an automated process known as transcompiler, transpiler, or code translation [1]. Code translation is distinct from traditional compilers because it operates between high-level languages with similar abstraction levels instead of translating code to lower-level languages for execution [2].

Code translation is significant across multiple domains, enabling developers within companies or open-source projects to efficiently integrate new code from different languages. Furthermore, it facilitates the modernization of codebases that were written in earlier languages and provides a quick and simplified method to enhance their functionality.

Code interoperability is another application, where code translation allows for the seamless integration of components or modules written in different languages by translating interfaces and function calls between them. Code translation can also facilitate code reuse and adaptation, as code snippets or libraries can be translated to meet the requirements of a different language or framework. [3]

2. Literature Review

2.1 Large Language Models

2.1.1 Large Language Models Overview

Large language models, also known as LLMs, are a type of artificial intelligence (Deep Learning) that has the ability to imitate human intellect. Using statistical models, they analyze large quantities of data and grasp the relationships and patterns among words and phrases, allowing them to create new written content, such as essays or articles, that closely resembles the style of a particular author or genre.

A large amount of data, including books, journals, and web pages, is supplied to train this large language model. This approach allows the model to detect complex relationships and patterns between words. When the model is trained with more data, it becomes more efficient at creating new content.

The large language model can generate new content by adhering to user-specified parameters after the training phase. As an example, if you desire to compose an article in the style of Shakespeare, you would provide the Large language model with a prompt, such as a sentence or paragraph, and it would generate the rest of the article based on the patterns and connections it has learned from analyzing
Shakespeare's works. [4]

2.1.2 Large Language Models Applications

Large language models have transformed the fields of artificial intelligence and natural language processing, offering up a wide range of applications and opportunities. These models, powered by advanced deep learning techniques, have the ability to understand, generate, and manipulate human language in a remarkably sophisticated manner. With their vast knowledge and language capabilities, large language models have found a wide range of applications across various domains, promising to enhance productivity, creativity, and communication in numerous fields. From assisting in content creation to aiding in language translation, answering questions, and even simulating human-like conversations, the applications of large language models continue to expand and reshape how we interact with technology and harness the power of language. In the following sections, we will explore some of the key applications where large language models are making a significant impact.

Text Generation

Text generation is a process of generating written text that appear like have written by human, AI system try to emulate human styles and patterns in writing during this process. As part of this process, the goal is to produce language that is relevant and consistent with how people naturally communicate. Text generation has gained significant importance in wide range of fields, including content creation, natural language processing, customer service, customer service, coding assistance. Text generation generates an output text by taking input data into language models and algorithms in order to process them. It includes training AI models on large datasets of text to learn grammar, contextual information, and patterns. These models then employ this acquired knowledge to produce new text in response to provided instructions or situations. [5]

Text generation heart is language models, which represent a crucial part in the process, such as GPT (Generative Pre-trained Transformer) and Google’s PaLM, which have been trained on large amounts of text data from the internet.By leveraging deep learning techniques, particularly neural networks, these models understand the intricacies of sentence structures and generate text that is both coherent and contextually appropriate and relevant. [5]

In text generation process, the input data will be entered. Input can be sentence or keyword. Then the next probable word or phrases will be predicted by the AI model using it's learned knowledge from input

data. The model keeps producing text while taking context and coherence into account until a certain length or condition is found. [5]

The most effective text generating models are highlighted in the list below:

1. GPT-4. OpenAI's (and the world's) most advanced system, which generates responses that are both safe and useful.
2. Claude. A next-generation AI assistant developed by Anthropic, designed to be helpful, honest, and harmless.
3. ChatGPT. This model is a lot like InstructGPT, but it's trained to follow prompts and provide detailed responses.
4. Nous-Hermes. A state-of-the-art language model that's been fine-tuned on over 300,000 instructions, developed by Nous Research.
5. Falcon LLM. This is a foundational large language model with 40 billion parameters trained on one trillion tokens developed by TII.
6. PaLM 2. The next-generation large language model that builds on Google's legacy of groundbreaking research in machine learning and responsible AI.
7. LLaMA. A foundational and state-of-the-art open-source 65 billion parameter large language model developed by Meta AI. [5]

It is important to take in mind the limitation of text generation to better understand the challenges and potential drawbacks of this technology. By recognizing these limitations, we can work towards addressing them and developing more robust and reliable text generation systems.Lack of contextual understanding: Text generation models often struggle with comprehending the broader context and nuances of language. They generate text based on patterns in the training data without truly understanding the meaning or intent behind the words. This can lead to inaccuracies, ambiguity, or nonsensical outputs.

- Overreliance on training data: The variety and quality of training data is one of the most important part on generating text generation models. If training data is biased or limited or doesn't represent full range of language diversity, the output text will not be as expected and contains some faults.
- Difficulty in handling rare or unseen scenarios: These models can produce incorrect predictions because they deal with scenarios that rarely occur, or were not present in the training data, or even did not have similar scenarios during the model's learning phase.
- Ethical considerations: Text generation models must be constantly monitored, so they can sometimes produce incorrect text or content or cause harm, which can then be used in the wrong direction and spread harm. [5]

Summarization

Summarization involves the process of compacting a given text into a shorter form while retaining essential information and preserving the intended meaning. Since text summarization by human consumes a lot of time and effort, the task's automation is becoming more and more common, which is a significant driving force behind academic study. [6]

With the advent of the big data era, there has been an unprecedented surge in the volume of text data originating from diverse sources. There should be more research and development of NLP in the field of text summarization, due to the ever-increasing amount of available documents. Automatic text summarization is the task of generating a concise and fluent summary without any human help while keeping the meaning of the original text document. [6]

Machine learning has given rise to various models designed for addressing the task of text summarization. Broadly speaking, there are two primary approaches employed in automatic summarization models: extraction and abstraction. [6]

Using a scoring algorithm, extractive summarization selects phrases from the source text to create a meaningful summary. This technique works by highlighting key passages in the text, clipping them out, and then piecing the remaining information together to create a condensed version. [6]
Abstractive summarization methods strive to generate concise summaries by interpreting the text using sophisticated natural language techniques, resulting in the creation of a new and condensed text that captures the essence of the original content parts of which may not appear as part of the original document, that conveys the most critical information from the original text, requiring rephrasing sentences and incorporating information from full text to generate summaries such as a human-written abstract usually does. A good abstractive summary really includes the information essential details and is linguistically flexible. [6]

Sentiment Analysis

Sentiment analysis, also called opinion mining, is the field of study that analyzes people's opinions, sentiments, evaluations, appraisals, attitudes, and emotions towards entities and their attributes.In the realm of sentiment analysis, various types exist, such as advanced sentiment analysis that can extract the emotional state expressed by the author of a text. More basic sentiment analysis aims at determining the polarity of a text: whether it delivers a negative or positive sentiment, and this task considered as classification because numerically, polarity may be gauged by a polarity score that takes values in some continuous range, typically on the interval between -1 and 1 and then assign each text to one of the categories "negative," "neutral," or "positive". [7]
Sentiment analysis can be applied on three levels: document level, sentence level, or aspect level.
Document level: Task task at this level is known as document-level sentiment classification. This type determines if the whole document represents positive or negative sentiment. So the document is specific only to one product, not multiple products.
Sentence level:This type is focus on the sentence level and specify weather this sentence represents a positive,negative,or neutral opinion, this type is near to something called subjectivity classification, which briefly differentiate sentences those represent factual information from sentences those represent subjective views and opinions.
Entity and Aspect level: Old name of this level is called feature level (feature-based opinion mining and summarization). The two previous types of sentiment analysis, document-level and sentence-level analyses cannot provide information about what exactly people liked and did not like. So this type of analysis can be said to provide more accurate and useful insights. [7]
Sentiment Analysis also has certain limitations and issues:
Although sentiment words (words that are used to express positive or negative sentiments) and phrases are essential parts of sentiment analysis, merely relying on them is not enough. The issue is considerably more intricate. In other words, we can say that the sentiment lexicon is necessary but not sufficient for sentiment analysis. Below, we list a number of issues:

1- Sentiment word may have a negative meaning in one situation and a positive meaning in another, and vice versa. For example, "This camera sucks", In this sentence, the "sucks" word has a negative meaning, but when we say, "This vacuum cleaner really sucks," it reflects a positive sentiment.

2- Question (interrogative) sentences and conditional sentences may cause one type of issue because they form a sentence with sentiment words but do not represent negative or positive sentiment. For example, in "Can you tell me which Sony camera is good?" and "If I can find a good camera in the shop, I will buy it.", "good" neither expresses a positive or negative opinion. Although this exception exists, some of the question and conditionsl sentences represent sentiment.
3- This point of issues normally comes in political opinions; these opinions depend on sarcastic sentences with or without sentiment words, which make it difficult to analyze them. For example, "What a great car! It stopped working in two days.".

4- Sentences written without sentiment words but reflecting an opinion are also considered one of the issues. these sentences cannot be neglected. An example of this sentences are "This washer uses a lot water", this sentence has not sentiment word, but gives a negative sentiment because it wastes and consumes a lot of water. [7]

2.1.3 Large language Model Limitations

Large language models are subject to a number of limitations. While these models possess outstanding capabilities, it is crucial to understand their constraints and consider the potential challenges that may arise. By acknowledging these limitations, we can gain a more comprehensive understanding of the capabilities and boundaries of large language models. The following section outlines some of the key limitations that researchers and practitioners have identified, shedding light on the areas where these models may fall short or require further refinement. It is important to note that these limitations should not undermine the significant advancements made in the field, but rather serve as a reminder of the complexities involved in developing and deploying large language models.

The first limitation comes with respect to speed and cost. LLMs require large computational capabilities, hence high response times. This problem increases as the input document gets longer. In addition, these models require a lot of specific GPUs and processing power, more than any normal deep learning models. For all of these requirements. LLMs are not supposed to be the best choice for specific (HR) tasks due to their cost. Secondly, large language models (LLMs) are often considered black boxes, as their output can be ambiguous even to their developers, making it difficult to explain their behaviour. This lack of interpretability raises concerns about transparency and trustworthiness, as it hinders understanding and addressing potential biases or unintended consequences. Thirdly, LLMs main objective is to produce language preceives like "natural" like humans. They are not designed to generate trustworthy information. And this can lead to serious consequences, like the example of ChatGPT, which provides wrong information about COVID-19 vaccination. [8]

2.2 Deep Learning

2.2.1 Deep Learning Overview

Deep learning, a subfield of machine learning, relies on artificial neural networks(ANNs) as its foundation. Artificial neural networks also known as deep neural networks (DNNs). Neural networks are inspired by the structure and function of the human brain’s biological neurons. ANN learns and process from the input data based on "neurons", which represent layers of interconnected nodes with capable of learning patterns and complex relationships between the data. In a fully connected deep neural network, there exists an input layer followed by one or more hidden layers that are sequentially interconnected. Each neuron in these layers receives input from the neurons in the previous layer or the input layer itself. The output of one neuron serves as the input to other neurons in the subsequent layer, and this pattern continues until the final layer generates the network's output. Through a series of nonlinear transformations, the layers of the neural network facilitate the transformation of input data, making the network to learn intricate representations of the input data. The key factor that has made deep learning popular and widely used is its ability to operate without the need for explicit programming. Additionally, the availability of large datasets has significantly contributed to the increased popularity of this technology. [9]

2.2.2 Deep Learning Applications

Deep learning encompasses a versatile range of applications, including supervised, unsupervised, and reinforcement machine learning. It employs diverse methodologies to process and analyze data in these different learning paradigms.

- Supervised Machine Learning: Supervised learning is based on labeled datasets. Neural network learns to make prediction or classify data depending on these labels. The input features and target variables must be included in dataset as a input. Learning process is done by the cost or error that comes from the difference between actual and predicted value , which called back propagation process. Deep learning algorithms such Convolutional neural networks, Recurrent neural networks are used for supervised tasks like image classifications and recognization, sentiment analysis. [9]
- Unsupervised Machine Learning: Unlabeled dataset without target variables is used in this learning technique in order to find the patterns or cluster the dataset. Deep learning algorithms like autoencoders and generative models are used for unsupervised tasks such as anomaly detection, dimensionality reduction. [9]
- Reinforcement Machine Learning : Agent in this learning type takes an action and observing the resulting rewards in order to interact with the environment. This agent aims to maximizes a reward signal by learning how to make a decision. Deep reinforcement learning algorithms such as Deep Q networks is used to reinforce tasks like robotics and game playing. [9]

2.2.3 Deep Learning Limitations

There is no denying how promising deep learning has been and how successful it has been, but it also has limitations. These limitations can impact the performance, generalizability, and interpretability of deep learning models. For deep learning techniques to be effectively utilized and robust solutions to be developed, it is necessary to understand these constraints. Below, we will examine some of the important limitations of deep learning, highlighting areas in which further research and advancement are needed. Recognizing these limitations allows us to use deep learning's full potential while respecting its limitations.

Firstly, deep learning model is limited to knowledge from the information existing in training data, hence this model will not be suitable to be applied or used in other or larger functional areas. Secondly, the DL models are sensitive to biased data, if the model trained on not balanced data then it will make also biased results. Thirdly, Due to the learning pace, deep learning models may face significant challenges. If the rate is too high, the model will converge too rapidly. If the pace is too slow, it may become stuck and difficult to find a solution. [10]

2.3 Code Translation

2.3.1 Code Translation Overview

Code translation, embodied in the concept of a transcompiler or source-to- source translator, plays a crucial role in converting source code from one high-level programming language to another. This process is particularly valuable for achieving interoperability, updating codebases from obsolete languages to modern ones, and integrating diverse code written in different languages [2].

This process is typically performed to achieve interoperability, migrate legacy systems, or leverage the strengths of a different language. Translating code requires understanding both the source and target languages, handling syntax and semantic differences, and ensuring the functionality is preserved during the conversion. Automated tools, like compilers and transpilers, often play a crucial role in this translation process [11].

2.3.2 Code Translation Application

Code translation plays a vital role in the dynamic landscape of software development, especially when migrating systems from one programming language to another. This process is particularly beneficial during the transition of legacy systems to more contemporary languages or platforms. Moreover, it facilitates cross-

platform development by adapting code initially written for a specific platform to seamlessly operate on another platform with a different programming language, thereby enhancing overall compatibility. Additionally, code translation promotes code reusability, allowing the transformation of code snippets or modules from one language to another for efficient reuse across various projects or environments. Another critical aspect is language interoperability, which empowers communication and integration between software components or systems that are originally written in different languages, fostering a cohesive and interconnected development ecosystem [11].

2.3.3 Code Translation Limitation

Code translation achieves great importance, but at the same time it faces difficulties and gaps that must be mentioned, which are as follows:

Rule Base Approach:

Conventional rule-based approaches are costly in terms of human labor, time, and expertise because they require static analysis, conversion to
structured objects, and handcrafted rewrite rules. This creates a gap in terms of scalability and efficiency for large-scale code translation tasks.

Code Translation Limitation using LLMs

The biggest challenge facing code translation using LLMs is that its data is incomplete and therefore provides quality unsecured results which make LLMs less applicable, especially in situations where data for historical or modern programming languages is scarce. [1]

Peculiarities of Programming Languages:

Programming languages are different from natural languages in that a small change to a word or token in a code snippet can drastically change its meaning. So,Adapting LLMs to the intricacies of programming languages needs innovative techniques to guarantee precise and meaningful code translations. [1]

Low-Resource Programming Languages:

Low-resource programming languages have limited bilingual or even monolingual training data. Since these are the languages where the need for translation is greatest, filling in the data shortage for both recent and ancient languages becomes essential. [1]

3. Related Work

Given the importance and benefit of automatically converting source code from one programming language to another, the author in [12], conducted an empirical study of code translation using a large language model(LLM) and found a number of problems needs to be solve, including that LLM based code translation did not reach the required level of efficiency based on a number of programming languages and benchmarks, especially in real-world projects. Also, the rate of incorrect translation is relatively high, ranging from 52.7% to 97.9%. In addition there is a need to identify and classify the ambiguity of underlying roots causing translation errors. Also, try to identify effective prompt-crafting techniques that can increase the success rate of translation.

This study contributes to the field of code translation using large language model by presenting the first taxonomy of bugs that occur in the translation process, this study contributes a comprehensive categorization of 14 bug types. These bug types were manually labeled and extracted from the code translation process performed by LLMs. This work also offers suitable contexts that greatly impact the improvement of effective results in code translation using LLMs. The evaluation of LLM for code translation involves applying it to a real world project and assessing its performance based on three benchmarks. Also they make all resources and tools available to public for future research in addition of automation scripts that facilitate the assessment the performance of LLMs.

The authors proposed solution to develop automatic translation of between different programming languages includes (C, C++, GO, JAVA, and python), using LLMs by understanding the current limitations related to this technology, so they conducted an emprical study. It began by selecting appropriate datasets. The authors used three data sets and two open source projects for real word projects part. The authors relied on seven LLMs to evaluate their performance in translation based on weather the translated code compile or not, fails with a runtime error or not,and existing test on translated code fail or not. The four pieces of information are included in each prompt template of LLMs. The results showed poor performance in all models except GPT-4 and StarCoder. The authors performed manual analysis and investigation that led to build a taxonomy with 14 bugs categories organised into four groups. In order to mitigating these bugs and improve the performance, the additional contextual information is added to the prompt using iterative technique. the iteration and the mitigation process terminate based on increase of successful translation is less than five percentile.

While the paper addresses the important task of code translation using large language models, there are critical aspects that need to be addressed to strengthen the validity and impact of the findings. The evaluation of LLMs in this paper relies solely on compilation and execution, which provides a practical assessment. However, it is also essential to include static metrics to compare the performance of the models with related works. In addition to the valuable bug taxonomy provided in this paper, it couldn't take advantages from more case studies about bug taxonomy in other works.

Other research "*Code Translation and Multilingual Code Co-Evolution* " [13] and talk about the code translation and multilingual code co-evolution, recent research has emphasized of large language models (LLMs) to address the challenge of automatically translating code changes from one programming language to another.The problem addressed in the document is the translation of code from one programming language to another. The authors propose a new approach using large language models (LLMs) and introduce a model called '*coeditor*' to tackle this challenge. so this reaserch paper,"Multilingual Code Co-Evolution Using Large Language Models" introduces a novel LLM named '*coeditor*'. to address the problem .

Their contribution is coeditor, which is built on encoder-decoder framework using a transformer-based encoder and decoder.It is initialized with a trained language model called CoditT5, it like a helpful starting point with the right built-in knowledge for modeling edits.coeditor learned to align and make changes on the old version of the code in the selected programming language, which helped it become more adept at the multilingual coediting task.Based on the modifications made on the source code, it create edit sequence to update the selected code. A dataset was used from open-source Java and C# projects is used to train and assess the model. The outcomes shows that *coeditor* performs better than existing models and achieve high

accuracy in translating code changes between programming languages.

The results:

1. The Code Change Histories: The impact of code change histories on multilingual code is examined in this study. Code change history is available to" history-aware models", which perform better than "history-agnostic models". The rule-based model CopyEdits performs relatively to the history-agnostic model CodeT5-Translation, so the contextual information provided by code change histories in multilingual code is important.
2. coeditor, an edit-based model, excels over generation-based models in multilingual code co-editing.
3. Combining between the *coeditor* and generation models is optimizing accuracy, so *coeditor* excelling for longer code and generation models for shorter ones, improving accuracy by 6%.“ [13]

The paper's investigation of combining *coeditor* with generation-based models demonstrates its adaptability. The *coeditor* excels in longer code sections, while generation models handle shorter ones more efficiently. It appears that using this combination of models leads to an improvement in the accuracy of code translation. Overall, the workflow of Codeditor, which involves training, generating edit sequences or plans, and performance evaluation, offers a comprehensive approach to code co-editing.
To summarize, the paper provides a promising solution to the code translation across diverse programming languages by the introduction of the coeditor model. Codeditor outperforms existing approaches, including rule-based translation tools and generation-based models, on multiple metrics. The particular emphasis on the integration of *coeditor* with generation-based models underscores its adaptability and the potential for achieving heightened accuracy in outcomes.

In this paper “MarianCG: a code generation transformer model inspired by machine translation
“ [14], The author discusses the challenge of code generation from natural language descriptions.The main problem is specifically how to create a code generation model (MarianCG) that can efficiently and precisely convert descriptions in natural language into executable Python code. In the fields of programming and artificial intelligence, this is a major challenge because it requires bridging the gap between human language and instructions that a machine can follow. So, primary issue that the author addresses has to do with code generation and the use of language models.Accordingly, The solution proposed is the development and implementation of a Transformer language model called MarianCG. The field of code generation from natural language descriptions is the intended application for this model. The author's approach involves fine-tuning the pre-trained language model (MarianMT) with specific datasets, namely CoNaLa and DJANGO.
MarianCG seeks to solve the problem of code generation by applying the pre-trained language model's skills to the particular task of converting descriptions of natural language into Python code with the use of evaluation metrics like the BLEU score and exact match accuracy to assess the quality of the generated code.
The result of the study showed that after the three experiments of improvement and development, the MarianCG model achieved a 34.43 BLEU score with a 10.2% and exact match accuracy. In terms of exact match accuracy and BLEU score, MarianCG ranked as the top model due to its accurate predictions. This model's compact size, speed, and accuracy are its main advantages. But it still lacks flexibility in terms of detecting whether the language used is programming or natural language and then allowing it to be translated into a language other than Python. [14]

Other paper is talk about the use of LLMs as reference-free evaluators for Natural Language Generation (NLG) tasks has gained attention among researchers. The author in [15] focuses on the evaluation of code correlations with functional correctness and human preferences. The paper evaluates the framework of four programming languages (Java, Python, C, C++, and JavaScript) from two aspects (human-based usefulness and execution-based functional correctness) by comparing its performance with the state- of-the-art CodeBERTScore metric, which relies on a pre-trained model and summarizes findings as follows: LLMs demonstrate a profound understanding of source code and functionalities in different programming languages, and LLMs can assess generated code with reference code. In LLM-based evaluation, high-quality references are essential. In addition to improving the accuracy and efficiency of code generation evaluation, zero-shotCoT improved the reliability of LLM-based evaluation by capturing and reasoning about source

code logic and structure. The author uses the prompt-based evaluation method, inspired by G-EVAL, which comprises two main components:

1) defining the task, evaluation criteria, and detailed steps.
2) providing a specific problem and the corresponding generated code snippet for evaluation.

This approach allows a comprehensive assessment of code quality by considering the human judgment of code usefulness and execution-based functional correctness.

The author describes two conducted experiments to explore the correlation between performed evaluations by a language model (LLM) and human preference and functional correctness. They compare the performance of LLM-based evaluations with seven automatic evaluation metrics, including a state-of-the-art metric called CodeBERTScore. For measuring the correlation with human preference, they use the CoNaLa dataset and human annotations on generated code from different models. For measuring the correlation with functional correctness, they utilize the HumanEval-X dataset. The authors exclude the evaluation option of distinguishability, as prior research has shown it to be an unreliable metric.

Through the solution proposed by the author, most code generative models do not consider code formatting, leading to unformatted code that requires additional formatting for comprehension and execution. Existing evaluation metrics for code generation rely on language-specific program parsers, but recent research suggests that Language Models (LLMs) can understand programming context even without proper formatting. Therefore, in the evaluation process the problems and generated code (and reference code, if available) are inputted to assess the LLMs' understanding and performance. [15]

4. References

[1] D. K. e. a. Ruchir Puri, "Code Translation," 2021. [Online]. Available: https://research.ibm.com/projects/code-translation. [Accessed 15 12 2023].

[2] M. A. Lachaux, B. Roziere, L. Chanussot and G. Lample, "Unsupervised Translation of Programming Languages," *arxiv,* vol. 3, p. 21, 2020.

[3] "Deep learning to translate between programming languages," Meta, 2020.

[4] "Large Language Model," boost.ai, [Online]. Available: https://www.boost.ai/blog/llms-large-language-models. [Accessed 15 11 2023].

[5] A. A. Awan, "What is Text Generation?," datacamp, 5 2023. [Online]. Available: https://www.datacamp.com/blog/what-is-text-generation. [Accessed 15 11 2023].

[6] L. Goncalve, "Automatic Text Summarization with Machine Learning — An overview," medium, 4 2020. [Online]. Available: https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25. [Accessed 16 11 2023].

[7] B. Liu, Sentiment Analysis and Opinion Mining, Morgan & Claypool, 2012.

[8] M. Rotaru and K. Kok, "Seven limitations of Large Language Models (LLMs) in recruitment technology," textkernel, [Online]. Available: https://www.textkernel.com/technology/seven-limitations-of-large-language-models-in-recruitment-technology/. [Accessed 17 11 2023].

[9] saumyasazena, "Introduction to Deep Learning," geegsforgeeks, 4 2023. [Online]. Available: https://www.geeksforgeeks.org/introduction-deep-learning/. [Accessed 17 11 2023].

[10] Zaveria, "What is Deep Learning, its Limitations, and Challenges?," analyticsinsight, 19 1 2023. [Online]. Available: https://www.analyticsinsight.net/what-is-deep-learning-its-limitations-and-challenges/. [Accessed 17 11 2023].

[11] L. Qiu, "Programming Language Translation," p. 14.

[12] A. R. I. R. K. e. a. Rangeet Pan, "Understanding the Effectiveness of Large Language Models in Code Translation," *arxiv,* p. 14, 2023.

[13] J. Zhang, P. Nie, J. J. Li and M. Gl, "Multilingual Code Co-Evolution Using Large Language Models," *Cornell Uneversity,* vol. 2, p. 13, 2023.

[14] A. S. Soliman, M. M. Hadhoud and S. I. Shaheen, "MarianCG: a code generation transformer model inspired by machine translation," *Journal of Engineering and Applied Science,* p. 23, 2022.

[15] T. Y. Zhuo, "Large Language Models Are State-of-the-Art Evaluators of Code Generation," *Cornell University,* vol. 1, no. 2304.14317, p. 39, 2023.