LARGE LANGUAGE MODEL FOR CODE TRANSLATION

Manal, Raghad, Eman, Shahad

- Introductin

- **Manual code conversion between programming languages is time-consuming, error-prone, and requires deep expertise due to differences in syntax, semantics, and language features.**

- **Automated code translation techniques, like transcompilers, reduce errors, improve efficiency, and enable seamless integration of code from different languages, enhancing codebases and promoting code interoperability**

Literature Review

- ## Large Language Model
  - Definition: type of artificial intelligence (Deep Learning) that has the ability to imitate human intellect.
    - Applications: Text Generation – Summarization – Sentiment Analysis.
    - Limitations: require large computational capabilities.

- ## Deep Learning
  - Definition: a subfield of machine learning, relies on artificial neural networks(ANNs) as its foundation.
    - Applications: Supervised – Unsupervised – Reinforcement
    - Limitations: limited to knowledge from information in training data.

Literature Review

- ## Code Translation

  - Definition : embodied in the concept of a transcompiler or source-to-source translator.

    - Applications: Code reusability- software conversion to other language.
    - Limitations:Low-Resource Programming Languages- small change in tokens can drastically change its meaning.

Related Work

1. Understanding the Effectiveness of Large Language Models in Code Translation

- **Problem**
  - The current LLMs for code translation did not achieve the required efficiency, in addition to the high rate of incorrect translation.
  - There is no systematic classification and identification of roots causing translation error.
  - The weakness in the current prompt techniques is considered one of the causes of translation errors.

1. Understanding the Effectiveness of Large Language Models in Code Translation

- **Contribution**

  - Taxonomy of translation bugs.
  - Improving the results of translation by offering suitable contexts.
  - Develop an iterative prompt crafting.

1. Understanding the Effectiveness of Large Language Models in Code Translation

- ## Proposed solution

The proposed solution serves translation between five PLs(C, C++, Go, JAVA, Python).
emprical study conducted and used three datasets, two real world projects, and seven LLMs.

outcome solution

-Taxonomy o f 14 bugs categories.
-New iterative prompt is created.

1. Understanding the Effectiveness of Large Language Models in Code Translation

- ## Critical Review

  - 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.

  - It couldn’t take advantages from more case studies about bug taxonomy in other works.

2. MrianCG: a code generation transformer model inspired by machine translation

- **Problem**
  - Challenge of code generation from natural language descriptions.
  - The gap that exists between instructions that can be executed by machines and human languages.
  - Vulnerabilities found in previous models that efficiently converts natural language descriptions into executable Python code.

2. MrianCG: a code generation transformer model inspired by machine translation

- **Contribution**
  - Acknowledgment of limitations, specifically a lack of flexibility in language detection and translation restrictions to Python.
  - Emphasis on the complexity involved in creating a model capable of accurately converting natural language into executable Python code.

2. MrianCG: a code generation transformer model inspired by machine translation

- **Proposed solution**

  - Fine-tuning pre-trained language model (MarianMT) using two datasets.
  - Development and implementation of MarianCG to solve the code generation problem.
  - Three-step experimentation and development process to enhance model performance.
  - Utilization of BLEU score and exact match accuracy as evaluation metrics.

2. MrianCG: a code generation transformer model inspired by machine translation

- ## Critical Review
  - Risk of misinterpretation or misclassification of language constructs.
  - MarianCG is confined to translating into Python, lacking support for other languages.
  - Lack of flexibility in distinguishing programming from natural language.

3. Large Language Models Are State-of-the-Art Evaluators of Code Generation

- **problem**

  - Difficult to develop evaluation metrics that align with human judgment.
  - The utilization of human-written test suites to evaluate functional correctness can be challenging in domains with low resources.

3. Large Language Models Are State-of-the-Art Evaluators of Code Generation

- ## Contribution

propose a new evaluation framework based on the GPT-3.5 for code generation assessments.

3. Large Language Models Are State-of-the-Art Evaluators of Code Generation

- ### Proposed solution

Evaluates the framework of four programming languages (Java, Python, C, C++, and JavaScript) from two aspects :

- Human-based usefulness.
- Execution-based functional correctness.

by comparing its performance with the state-of-the-art CodeBERTScore metric

3. Large Language Models Are State-of-the-Art Evaluators of Code Generation

- ## Critical Review

it is still uncertain whether LLMs can be effectively employed to evaluate other tasks related to source code beyond code generation.

4.Code Translation and Multilingual Code Co-Evolution

- ### Problem

- The main problem addressed in the paper is focuses on the task of automatically translating code changes from one programming language to another.

- ### Contribution

The contribution is the introduction of the ***Codeditor*** model, which is designed to address the challenge of automatically translating code changes from one programming language to another.

4.Code Translation and Multilingual Code Co-Evolution

- Proposed solution

Codeditor

01. built on CoditT5

02. Use LLMs

03 encoder-decoder framework

Codeditor demonstrates proficiency in handling the complexities associated with translating code

4.Code Translation and Multilingual Code Co-Evolution

- ## Critical Review

  - **Codeditor** showcases promise in handling longer code, yet there's a need for a closer look at its efficacy with shorter snippets.

  - Proposed integration with generation models boosts accuracy but warrants careful consideration due to potential complexities.

THANK YOU FOR LISTENING