e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

TEXT TO PSEUDO CODE GENERATION

Aniruddha Sonawane*1, Tejas Bhosale*2, Siddhi Retharekar*3

*1,2,3B.Tech CSE, VIIT, Pune, India.

DOI : https://www.doi.org/10.56726/IRJMETS45500

ABSTRACT

In the rapidly evolving landscape of software development, bridging the gap between software requirements and implementation remains a critical challenge. This research paper explores applying Artificial Intelligence (AI) and Machine Learning (ML) techniques to automate the process of generating pseudo-code from natural language text. The proposed approach harnesses the power of deep learning models, such as Transformers, and leverages vast text corpora for training to facilitate accurate and context-aware pseudo-code generation.

Keywords: Pseudo code, Deep Learning, Transformers, Data, Artificial Intelligence.

I. INTRODUCTION

Software development has witnessed a rapid evolution over the years, with an ever-growing demand for efficient and productive methods to translate high-level textual descriptions into executable code. The need for automated solutions to text-to-code generation has become increasingly apparent as software projects grow in complexity, spanning diverse domains and industries.

Traditional methods often involve human developers deciphering and translating textual requirements or specifications into code, a process that is time-consuming, subjective, and susceptible to miscommunication.

This research paper addresses the aforementioned challenges by focusing on the application of Artificial Intelligence (AI) and Machine Learning (ML) techniques to the task of text-to-pseudo-code generation. The primary motivation for this research lies in the potential transformative impact such a technology can have on the software development lifecycle. By automating the generation of pseudo-code from natural language text, software developers can save significant time and effort, allowing them to concentrate on higher-level aspects of design and architecture. Additionally, automated text-to-pseudo-code generation can reduce the risk of errors, improve code consistency, and enhance the maintainability of software systems.

II. METHODOLOGY

Data Collection

For training and evaluating our text-to-pseudocode generation model, we utilized a diverse dataset containing pairs of natural language text and corresponding pseudocode. The dataset was collected from various sources, including open-source code repositories, programming forums, and textbooks, to ensure a wide range of programming styles and domains.

Table 1. Size of dataset used per stage.

Stages	Dataset	Programming Languages	Number of samples in the Dataset
Text to Code	MBPP	Python	974
Code to Pseudocode	Django	Python	16000

- As shown in Table 1, our research utilized two datasets for the Text to Code and Code to Pseudocode stages.
- The first dataset, MBPP, consisted of 974 samples and was used for the Text to Code stage.
- The second dataset, Django, consisted of 16,000 samples and was used for the Code to Pseudocode stage.

Data Preprocessing

Data preprocessing is an important step in any machine learning model, including transformer models. In the context of transformer models, data preprocessing typically involves converting the raw input data into a format that can be used by the model. Here are some common data preprocessing steps for transformer models:

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

1. Tokenization: Tokenization is the process of breaking up the input text into individual tokens or words. In transformer models, tokenization is typically done using a tokenizer object that is specific to the particular model architecture. For example, the BERT tokenizer is used for BERT models, and the GPT-2 tokenizer is used for GPT-2 models.

2. Padding: Transformer models typically require fixed-length inputs, which means that all input sequences must be the same length. To achieve this, shorter sequences are padded with a special token (usually a zero) so that they have the same length as the longest sequence in the dataset.

3. Encoding: Once the input text has been tokenized and padded, it needs to be converted into a numerical format that can be used by the model. This is typically done using the tokenizer object, which maps each token to a unique numerical ID.

4. Data splitting: The dataset is usually split into training, validation, and test sets. The training set is used to train the model, the validation set is used to tune the model hyperparameters and prevent overfitting, and the test set is used to evaluate the final performance of the model.

5. Data augmentation: Data augmentation techniques such as data cleaning, noise reduction, and data synthesis can be used to improve the quality and quantity of the data used to train the model.

Feature Selection/Engineering

1. Transformers:

Transformers are attention-based models that use multi-headed self-attention instead of the recurrent layers that are usually used in encoder-decoder designs. The encoder receives the input sequence's word embeddings. After that, the data is converted and sent on to the next encoder. All of the decoders in the decoder stack get the output from the last encoder in the encoder stack. The Transformer can be trained much quicker than designs based on recurrent or convolutional layers for translation jobs.

2. Sequence-to-Sequence Models:

Sequence-to-Sequence is a neural network architecture that converts one sequence of elements into another, such as the sequence of words in a phrase. An encoder and a decoder make up the model. In the form of a hidden state vector, the encoder captures the context of the input sequence and provides it to the decoder, who subsequently constructs the output sequence. Seq2Seq models excel in translation, which involves transforming a series of words from one language into a sequence of other words in another.

3. T5forConditionalGeneration:

T5 for Conditional Generation is an application of the T5 (Text-to-Text Transfer Transformer) model where the focus is on generating text based on specific conditions or input. In this context, T5 takes an input text (condition) and generates the corresponding output text. This approach allows T5 to be used for a variety of tasks, such as text summarization, translation, question answering, and more, where the generated text is dependent on the provided input. It's a versatile and powerful model for conditional text generation tasks in natural language processing.

III. MODELING AND ANALYSIS

Model Architecture

A fine-tuned T5 transformer is a variant of the T5 (Text-to-Text Transfer Transformer) model that has been trained on a specific downstream task using transfer learning. The T5 model itself is a transformer-based architecture designed for various text generation tasks, such as summarization, translation, and question-answering.

1. Load the Pre-trained T5 Model: Start by loading the pre-trained T5 model from a pre-trained model checkpoint. This checkpoint contains the weights and architecture of the pre-trained T5 model.
2. Tokenization: Tokenize the input text you want to convert into code using the T5 tokenizer. The tokenizer splits the text into individual tokens that the T5 model understands.
3. Model Inference: Pass the tokenized input through the fine-tuned T5 model to obtain the generated code. The model applies a series of transformer layers to process the input tokens and generate the desired output.
4. Decoding: Convert the model's generated output tokens back into human-readable code using the T5 tokenizer's decoding capabilities.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

Evaluation Metrics

Blue Score:

- BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate's translation of the text to one or more reference translations.
- Although developed for translation, it can be used to evaluate text generated for a suite of natural language processing tasks.
- A perfect match results in a score of 1.0, whereas a perfect mismatch results in a score of 0.0.

Performance Analysis

- The Bleu score for the first stage of the project denotes understandable to good translations, which is evident from the syntactical correctness of the generated Python code. However, there may be logical errors in the Python code due to insufficient training data.
- The Bleu score for the second stage points to high-quality translations, with the generated pseudocode exhibiting a high level of logical and structural correctness.

IV. RESULTS AND DISCUSSION

Text To Code: Bleu score: 0.4

Code To Pseudocode: Bleu score: 0.74

The trained model demonstrates promising results in generating pseudocode from a diverse set of natural language inputs. However, challenges related to code complexity, rare commands, and handling variable input lengths remain. Post-processing techniques to improve code readability and structure are under investigation.

V. CONCLUSION

This research paper addresses the challenging task of generating code from natural language text, which is a significant undertaking in the realm of big systems. Instead, it serves as a foundation upon which future efforts can build to create a more reliable system. These recommendations highlight the potential for further advancements in this field.

In summary, this research paper aims to contribute to the field of software engineering by investigating and presenting a comprehensive approach to automating the text-to-pseudo-code generation process using AI and ML techniques. While this research represents a modest step towards achieving this goal, it is important to acknowledge that the initial prototype has yielded promising results at a granular level. However, it is essential to emphasize that the proposed model is not presented as an absolute solution to this complex problem. Instead, it serves as a foundation upon which future efforts can build to create a more reliable system. It addresses the critical need for efficient and accurate methods in the software development workflow and offers a promising avenue for improving the productivity and reliability of software development processes.

VI. REFERENCES

[1] Iyer, S., Konstas, I., Cheung, A., Zettlemoyer, L.: Mapping language to code in programmatic context. arXiv preprint arXiv:1808.09588 (2018).

[2] Saeki, M., Horai, H., Enomoto, H.: Software development process from natural language specification. In: Proceedings of the 11th International Conference on Software Engineering, pp. 64{73 (1989).

[3] Husain, H., Wu, H.-H., Gazit, T., Allamanis, M., Brockschmidt, M.: CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436 (2019).

[4] Lu, S., Guo, D., Ren, S., Huang, J., Svyatkovskiy, A., Blanco, A., Clement, C., Drain, D., Jiang, D., Tang, D., et al.: CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. arXiv preprint arXiv:2102.04664 (2021).

[5] Saeki, M., Horai, H., Enomoto, H.: Software development process from natural language specification. In: Proceedings of the 11th International Conference on Software Engineering, pp. 64–73 (1989).

[6] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., et al.: CodeBERT: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155 (2020).

[7] Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K.-W.: Unified pretraining for program understanding and generation. arXiv preprint arXiv:2103.06333 (2021).

[8] Norouzi, S., Tang, K., Cao, Y.: Code generation from natural language with less prior and more monolingual data. arXiv preprint arXiv:2101.00259 (2021).

[9] Parvez, M.R., Ahmad, W.U., Chakraborty, S., Ray, B., Chang, K.-W.: Retrieval augmented code generation and summarization. arXiv preprint arXiv:2108.11601 (2021).

[10] Oda, Y., Fudaba, H., Neubig, G., Hata, H., Sakti, S., Toda, T., Nakamura, S.: Learning to generate pseudo-code from source code using statistical machine translation. In: 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 574–584 (2015). IEEE.

[11] Alhefdhi, A., Dam, H.K., Hata, H., Ghose, A.: Generating pseudo-code from source code using deep learning. In: 2018 25th Australasian Software Engineering Conference (ASWEC), pp. 21–25 (2018). IEEE.