Code Explanation Matching: Leveraging NLP and Pseudocode Generation for Improved Code-Text Association

Dr. Sanjay Chatterji, Yashraj Singh, Pallav Saxena, Shubham Kumar

Abstract

In this research paper, we present a novel approach for matching code explanations with the corresponding code snippets. The task at hand involves associating natural language explanations with their corresponding code, which is a crucial step in various software engineering tasks such as code comprehension and documentation generation. Traditional methods relying solely on vector comparisons between Intermediate Representation (IR) and explanation representations yielded poor results due to the dissimilarity of vector spaces.

To address this challenge, we propose a hybrid approach that leverages Natural Language Processing (NLP) techniques and vector similarity. Initially, we explored comparing vector representations of LLVM-generated IR using IR2Vec with explanation vectors generated using sBERT. However, due to the disparate vector spaces, this approach did not yield satisfactory results.

To bridge the gap between the code and explanation representations, we employed language transformation models, including GPT2, FLAN, T5, and BART, to convert the code snippets into pseudocode. Among these models, BART produced the highest BLEU score, indicating superior performance in generating pseudocode. By converting the code into textual pseudocode, we aimed to extract more comprehensive textual information from the code snippets.

Subsequently, we utilized sBERT to generate vector representations of the pseudocode and compared them with the sBERT encodings of the explanations. This combination of NLP and vector similarity proved to be highly effective, resulting in significant improvements in matching code explanations with the appropriate code snippets.

The experimental results demonstrated the efficacy of our proposed approach, showcasing its ability to achieve accurate code-explanation associations. We believe that this research contributes to advancing the field of code comprehension and can find applications in various software engineering tasks, including code documentation generation, code understanding, and developer assistance tools.

Keywords: Code Explanation Matching, Pseudocode Generation, Natural Language Processing, Vector Similarity, sBERT, BART,

Introduction

For natural language processing, understanding code snippets in any of the programming languages like C, C++, Python is a difficult task as there are no such strict grammar rules, instead, the code dependency is the logic of the code.

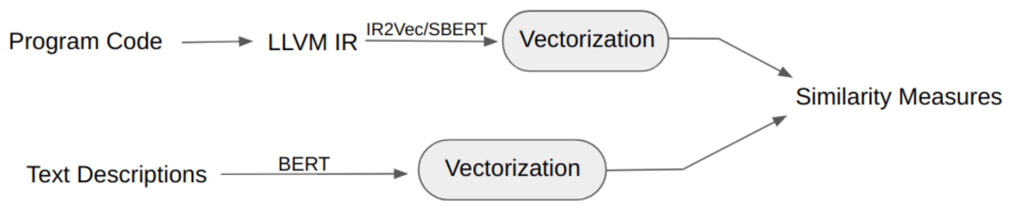
Therefore, to comprehend the code snippets, we shall develop such a model system, which is able to understand the semantics of the code, based on the core logic of each program. Hence, establishing a correspondence between the code and its description in natural language, allows to map and develop the major aspects of programming languages to model the relationship of logic blocks to its corresponding semantics described in a well known natural language (referred to as textual description in the rest of the report).

Previous Work

Previously, to establish the correspondence between C/C++ codes and their textual descriptions, code snippets were processed using LLVM framework for compilers to obtain Intermediate Representations (IR) of the C/C++ code.

Following which, three different paths were taken to produce comparisons and make deductions for the most optimal way to follow.

1. Direct comparisons (as a base case): using SBERT to perform vectorization and produce encodings for both code and textual descriptions; following similarity measures calculations.
2. Use LLVM to obtain IR of the code and map the instructions to form a raw natural language document; followed by SBERT encodings on IR document and textual descriptions; followed by similarity measures calculations.
3. Use a tool, developed specifically for code to produce meaningful vector embeddings, called IR2Vec. Since SBERT is trained for natural languages, accuracy of vectorization is improved, when using IR2Vec toolchain to obtain vector encodings for code, and using SBERT only to obtain vector encodings of the textual description; followed by similarity measures calculations.



Major drawback of each method is that the obtained vector encodings of code and textual descriptions belong to entirely different vector spaces. Since, the encodings are (sampled) from such distinct spaces, similarity measures came out very poor.

The obtained conclusion posed the motivation for an entirely alternate way to approach the problem for more meaningful results with a robust model system.

Present Flow

The initial subtask of the current section was to determine a robust way to establish the semantics of code with textual description without compromising the correlation between the vector spaces.

After careful consideration of a set of procedures, the most optimal way out was to perform some transformations over the code snippets, such that the correlations in the vector spaces is enhanced, preserving the syntactic and semantic characteristics of the code.

Transformations were nothing but using large language transformer models to perform code to pseudo code transformations, which essentially would convert the code (programming language) into pseudo code (natural language). Since now, both processed code and textual descriptions belong to the domain of natural language, vector spaces sampling the encodings for both the sets would now carry strong correlation and similarities would now have meaningful comprehension.

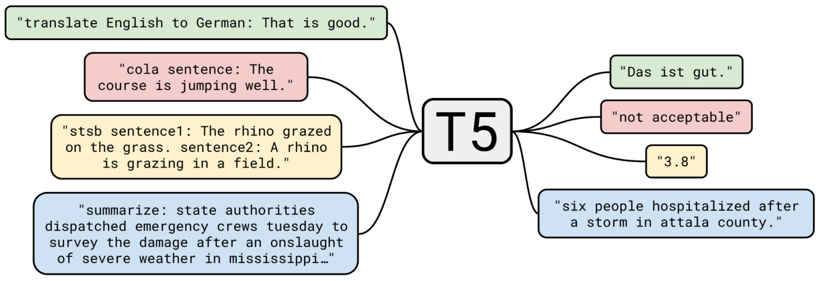
Models

For the code to pseudo code transformations, large language transformer models were chosen depending on their individual performance over specific tasks after fine tuning and ability to process and model multimodal language corpus.

Three transformer models were chosen, namely

1. T5 (Text-To-Text-Transfer-Transformer) model

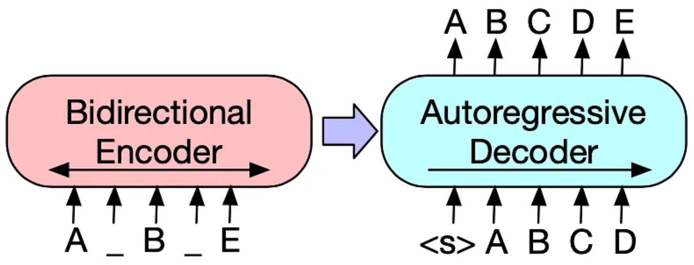
   T5, developed by google is a variant of transformer architecture that specializes in text generation taks. The input and output sequences are both represented as text, and the model is trained to transform one sequence into another. This architecture was exactly what was needed for the transformation task at our hand.


"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

2. BART (Bidirectional and Auto-Regressive Transformer) model

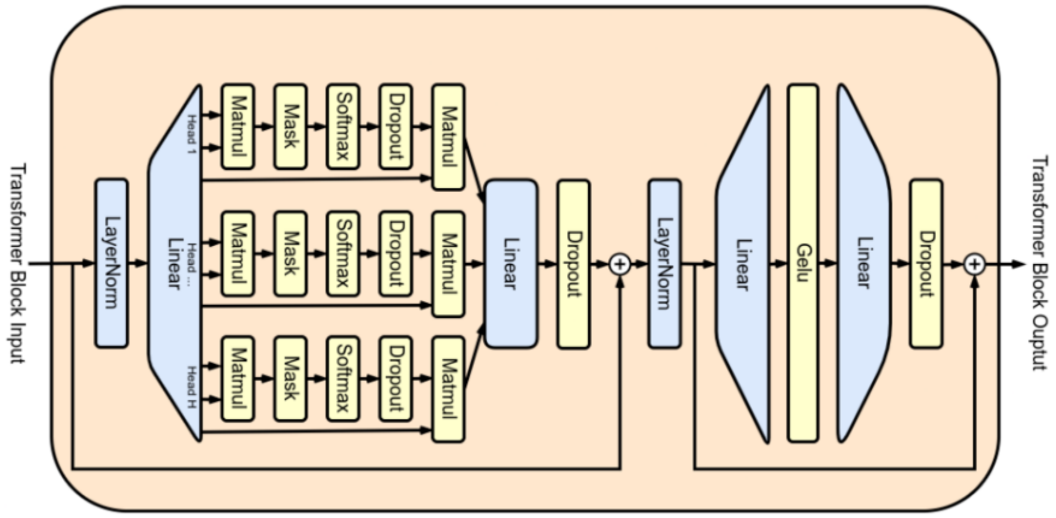
   BART is a sequence-to-sequence (Seq2Seq) model architecture for NLP tasks, introduced by Facebook AI Research (FAIR) in 2019 with state-of-the-art results on various NLP benchmarks, such as machine translation, summarization, and question-answering.

   The model used for training, BART-base and 140 Million parameters and 6 encoder/ decoder layers. The encoder takes the input sequence and processes it in both forward and backward directions and the decoder then generates the output sequence one token at a time, using the representation from the encoder and the previously generated tokens.



3. GPT-2 (Generative Pretrained Transformer) model

GPT-2 is a LLM, trained to perform language inferences and modeling using the pretrained transformer architecture. The model works with an attention mechanism to find the relevant topics and words from the training corpus, and performs modeling over them to make predictions. The model built over 1.5 billion parameters, performs excellently on generative tasks given inductive attention to the architecture.



Inference

Models were fine tuned over SPoC dataset of C++ codes with its corresponding pseudo code. A set of *100k* premise-hypothesis pairs of SPoC data points were used to fine tune each of them for the use case. To test the models and to rank against themselves, BLEU and Rogue scores were calculated over a set of testing dataset.

1. **T5**

   BLEU score: *0.6498*

   ROUGE-1 score: *0.7065*

   ROUGE-2 score: *0.4583*

2. **BART**

   BLEU score: *0.6388*

   ROUGE-1 score: *0.7153*

   ROUGE-2 score: *0.4788*

3. **GPT-2**

   BLEU score: *0.3727*

   ROUGE-1 score: *0.3098*

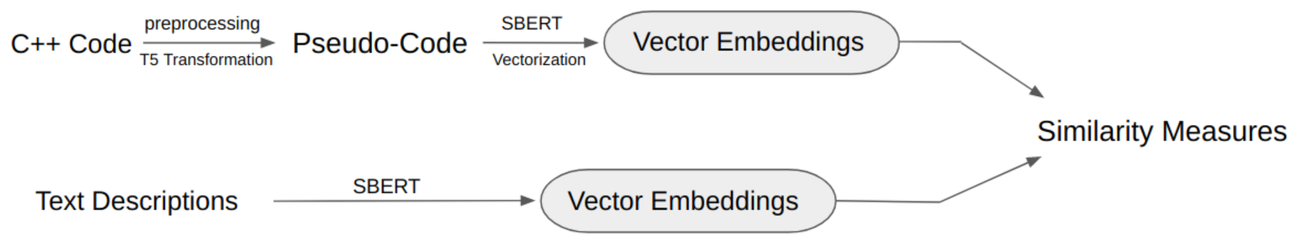
   ROUGE-2 score: *0.1214*

Collating the results, the decision was to develop the project flow using T5 model architecture for preprocessing the code snippets, i.e. performing the code to pseudo-code transformations.

Encodings

Once the preprocessing is done on the code snippets, the obtained Pseudo-Code, and Text Descriptions are encoded using SBERT, performing vectorization. The obtained embeddings are defined over similar vector spaces of the natural language.

Following is the final implementation of the project flow with optimal results



Inference

Once the encodings are obtained, similarity measures are calculated over the pairs of pseudocode and textual descriptions. We prepared a set of *1000* data points, each <pseudo-code, description, label> after fine tuning the T5 model and using a testset of *1000* data points.

Then we sampled *50* points from the prepared set, to perform the similarity measures. Each pseudo-code in the set of *50* samples was tested against the entire set of *50* descriptions (mapped one-to-many), and top *3* among matches were selected on the basis of cosine similarity.

Results

Collating the top *3* results for each datapoint in the test set, following are the aggregated similarity scores (Cosine Similarity and Euclidean Distance)

Cosine Similarity: *0.7365*

Euclidean Distance: *14.2613*

Conclusion

In conclusion, our research presents a hybrid approach that combines NLP techniques and vector similarity to address the challenge of matching code explanations with code snippets. By converting code snippets into pseudocode using the BART model and comparing sBERT vector representations, we achieved significant improvements in accurately associating code explanations with the corresponding code. This approach outperformed traditional methods based solely on vector comparisons. The results highlight the potential of our approach in enhancing code comprehension, code documentation generation, and developer assistance tools. Future work could explore the scalability and applicability of our method to different programming languages and datasets. Overall, our hybrid approach shows promise in advancing code comprehension and software engineering tasks.