Source Code Summarizer

Nisha Jacob (8915179886), Shreeram N (1937566881), Muskaan Parmar (1660937440)

USC Viterbi
School of Engineering

University of Southern California

Introduction

- Code summarization is a task that tries to comprehend code and generate descriptions from the source code.

- This summary would include a description of the logic and the function of the code. Programmers can save time and effort in describing what their code performs by making use of the code summarizer.

- It would also allow users to look up codes written by another developer to understand how they had approached the problem.

Dataset

- Making use of the 251820 Python code files present in CodeSearchNet Corpus dataset.

- Preprocessed data stored in JSON lines format

- Each line in the uncompressed file contains an example with the following information: *id*, *repo*, *path*, *func_name*, *original_string*, *language*, *code*, *code_tokens*, *docstring*, *docstring_tokens*, *sha*, *partition*, *url*

- Two columns that are heavily used for the purpose of code summarization are *code_tokens* and *docstring_tokens*.

An example of how the tokenized code looks is as follows :

```
code   :          'def get_vid_from_url(url):\n'
          '       """Extracts video ID from URL.\n'
          '       """\n'
          "       return match1(url, r'youtu\\.be/([^?/]+)') or \\\n"
          "         match1(url, r'youtube\\.com/embed/([^/?]+)') or \\\n"
          "         match1(url, r'youtube\\.com/v/([^/?]+)') or \\\n"
          "         match1(url, r'youtube\\.com/watch/([^/?]+)') or \\\n"
          "         parse_query_param(url, 'v') or \\\n"
          "         parse_query_param(parse_query_param(url, 'u'), 'v')"

code_tokens  : ['def', 'get_vid_from_url', '(', 'url', ')', ':', 'return',
                'match1', '(', 'url', ',', "r'youtu\\.be/([^?/]+)'", ')',
                'or', 'match1', '(', 'url', ',',
                "r'youtube\\.com/embed/([^/?]+)'", ')', 'or', 'match1',
                '(', 'url', ',', "r'youtube\\.com/v/([^/?]+)'", ')', 'or',
                'match1', '(', 'url', ',',
                "r'youtube\\.com/watch/([^/?]+)'", ')', 'or',
                'parse_query_param', '(', 'url', ',', "'v'", ')', 'or',
                'parse_query_param', '(', 'parse_query_param', '(',
                'url', ',', "'u'", ')', ',', "'v'", ')']
```

Existing PreTrained Models

- Salesforce/CodeT5 is a unified pre-trained encoder-decoder Transformer model that leverages code semantics conveyed from developed assigned identifiers.
- Other existing methods employ conventional NLP pre-training techniques on the source code by regarding it as a sequence of tokens like NL.
- The model builds on a framework with the same architecture as T5 and specifically focuses on identifiers that reserve rich code semantics, fusing the information into a Seq2Seq model.

Tokenization

- We use the RoBERTa Tokenizer in our model implementation.
- RoBERTa has the same architecture as BERT but uses a byte level BPE as tokenizer.
- Special tokens: "<pad>","<s>","</s>","<unk>","<mask>"
- RoBERTa tokenizer treats spaces differently and encodes the words differently
- A combination of token and position embedding is applied on this input.

Our Implementation

Network Architecture

- Implemented a pre-trained model CodeT5 (60M) that builds on an encoder-decoder framework with the same architecture as T5.

- Consequently, we have used a Roberta Model (250M) as our encoder and have designed a custom Transformer decoder that consists of a self-attention layer followers by an encoder-decoder attention mechanism and a position-wise feed forward network with 2 Linear layers followers by ReLU activation and a dropout layer with a dropout rate of 0.1.

- Our model is a sequence to sequence model which has an encoder-decoder architecture followed by 2 Linear layers and a Softmax.

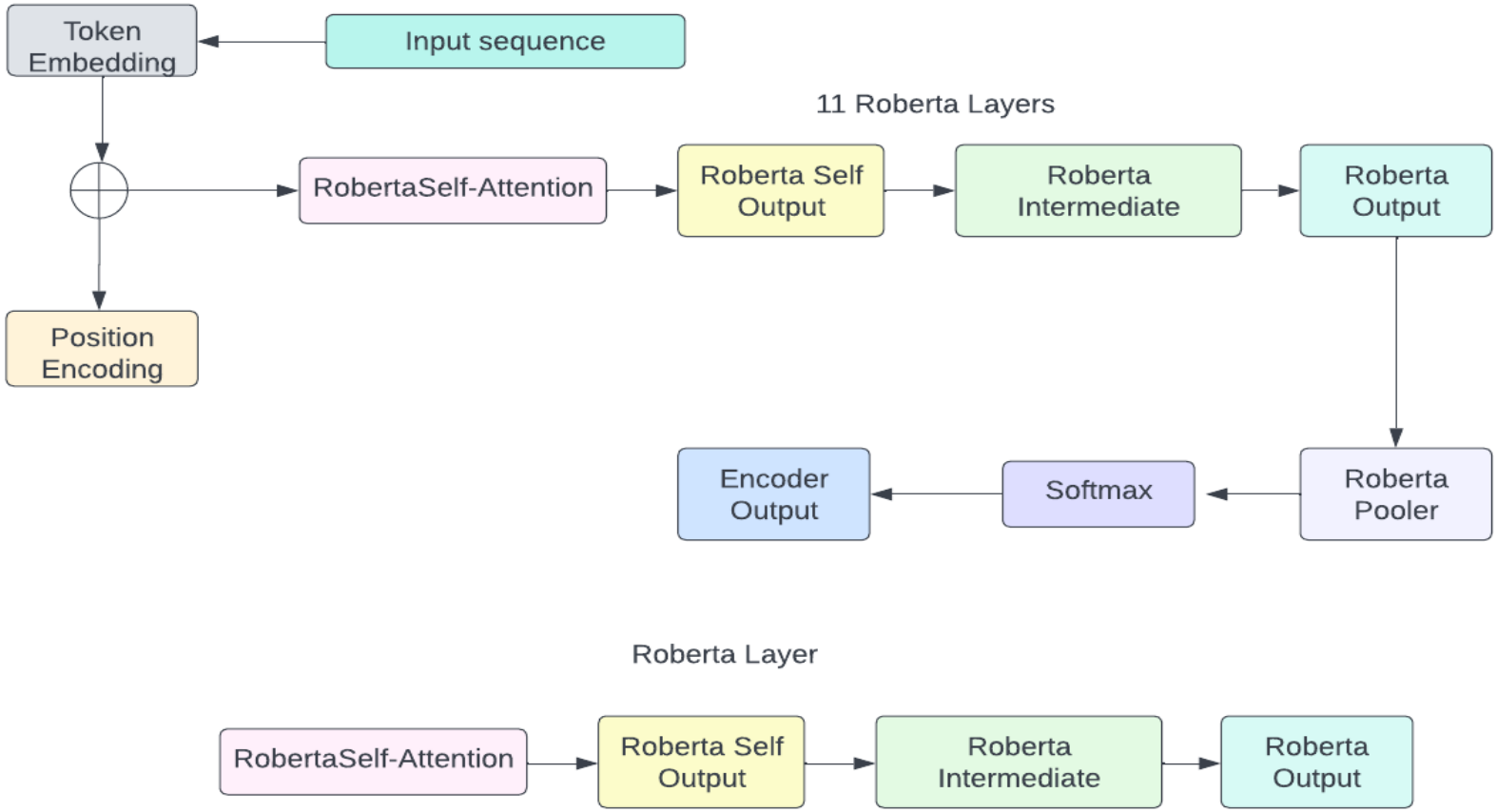
Encoder

- We use a pre-trained “Roberta-base” model with the *RobertaForMaskedLM* architecture for the encoder.

- MaskedLM is used as a pre-training objective in our encoder model which masks out certain words during training and the words are reconstructed from the surrounding information.

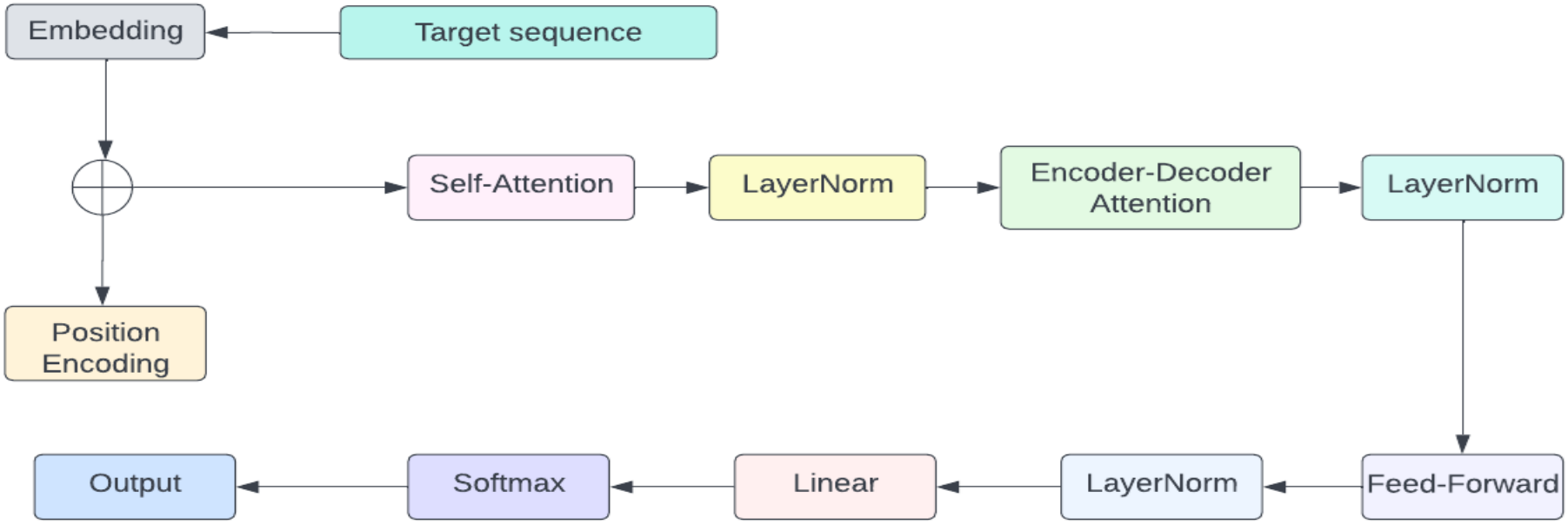
- We then get the embeddings for the target_ids i.e. the labels which in turn is fed to the decoder along with the encoder outputs.

Encoder Architecture



Decoder

- We are employing a Transformer Decoder with a self attention and an encoder-decoder attention head followers by a feed-forward network and a softmax.
- Our decoder architecture is as follows:



Training Details

- The networks were trained for 3 epochs. The optimizer used was Adam with weight decay and a batch size of 4.
- We set a constant learning rate of 5e-5 throughout the training process.
- We use the Cross Entropy Loss as our loss function.
- For training the pre-trained and custom designed networks we used an NVIDIA A100 40GB GPU.

Results

Input Code Snippet:

```
    def load_asf ( self , source , ** kwargs ):
        if hasattr ( source , 'read' ):
            p = parser.parse_asf (source, self.world, self.jointgroup, **kwargs)
        else:
            with open (source) as handle:
                p = parser.parse_asf(handle, self.world, self.jointgroup, **kwargs)
                self.bodies = p.bodies self.joints = p.joints self.set_pid_params(kp=
0.999 / self.world.dt)
```

Pre-trained Model Output: Load an asf from a source file.

Results

Input Code Snippet:

```
async def set_typing ( self , set_typing_request ):
    response = hangouts_pb2.SetTypingResponse()
    await self._pb_request('conversations/settyping', set_typing_request, response)
    return response
```

Pre-trained Model Output: Sets the typing of the conversations.

Results

Input Code Snippet:

```
def _serialize_items(self, channel_metadata_items):
    return json.dumps(self._prepare_items_for_transmission(channel_metadata_items),
                      sort_keys=True).encode('utf-8')
```

Pre-trained Model Output: Serialize items for transmission.

Results

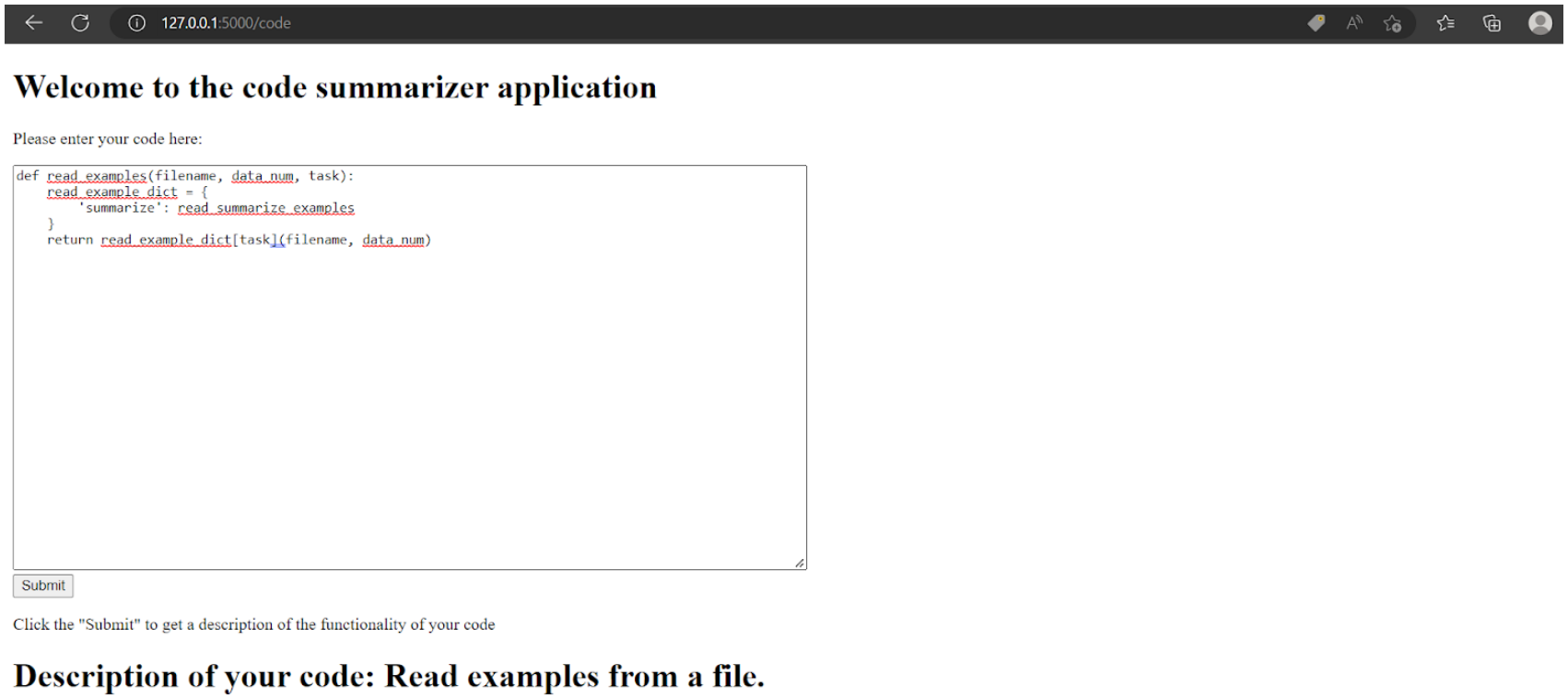
Scoring metric	CodeT5	Roberta-Base	Roberta Modified
Bleu-4 score	19.92	17.07	Training ongoing
EM score	1.6356	0.3419	Training ongoing

ML-OPS

- We plan to factor in automation by giving the users the ability to input a python file and obtain docstrings from our language model.

- We have written an automation script that takes in an input python file from the user, reads all of the functions inside it as text and is fed to the Roberta Model to get back a short description of the functionality of the code.

ML-OPS


127.0.0.1:5000/code
Welcome to the code summarizer application
Please enter your code here:
def read_examples(filename, data_num, task):
read_example_dict = {
'summarize': read_summarize_examples
}
return read_example_dict[task](filename, data_num)
Submit
Click the "Submit" to get a description of the functionality of your code
Description of your code: Read examples from a file.

References

- Wang, Yue, Weishi Wang, Shafiq Joty, and Steven CH Hoi. "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation." arXiv preprint arXiv:2109.00859 (2021).

- Ahmad, Wasi Uddin, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. "Unified pre-training for program understanding and generation." arXiv preprint arXiv:2103.06333 (2021).

- Ahmad, Wasi Uddin, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. "A transformer-based approach for source code summarization." arXiv preprint arXiv:2005.00653 (2020).

- Iyer, Srinivasan, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. "Summarizing source code using a neural attention model." In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 2073-2083. 2016.

Thank You