PMDL final project. Commit message generation with seq2seq language modeling.

Sergeev Nikita BS20-AI n.sergeev@innopolis.university

GitHub link: link

This project aims to create a model for the automatic commit message generation, given the code changes made in the commit.

Repository structure

- **CommitChronicle preprocessing** - Python script for the common preparation of the data before the training. This script transforms data from its initial raw format, presented in the dataset into the appropriate way, in which it can be passed to the model as an input and loss function can be calculated.
- **data processing** - This folder contains all the code for the data parsing from the GitHub API and it's preprocessing. This code wasn't used for the main model training as CommitChronicle is the way better. Nevertheless, this data can be used for the final model evaluation and measuring the model performance on the real data.
- **Evaluation** - This folder contains the scripts for the models evaluation and comparison. More precisely, in this folder you can find the measure of the BLEU score and BERTScore for the fine tuned version of the CodeT5+ model. It also contains the comparison of the model performance with the SOTA model, presented in the paper [1] from the references list.
- **Inference** - In this folder I implemented 2 scripts for the model.
  1) First one is - `get_commit_text.py`, which is the supplementary script. It's used to get the code changes in the commit from the github link. Further output of this script can be used as an input for the model.
  2) Second one is - `model_url_inference.py`. This script is end-to-end model inference by the github link. It takes the github link as an input and returns the commit message, generated by the model and the Original commit message, which was written by the author of the commit.
- **Interim Reports** - This folder contains the interim reports, which were written during the project development.
- **Model_train** - This folder contains the scripts for the model training.
- **tokens** - I used this folder to contain wandb and huggingface tokens.

Development process

Data collecting: As the first step, I decided to collect the data for the model training. I started by parsing the data from the GitHub API. In total I parsed ~ 180K commits. This data includes commits from 100 most starred GitHub repos in python.

Literature review: After the data collection, I started to read the papers, which were related to the topic of the project. There are several approaches to the commit message generation. The first one is the usage of the seq2seq transformer-based models described in [1-3]. And the second one is the usage Graph Neural Networks [4]. For the purpose of this project I decided to use the first approach, as it's more suitable for the commit message generation task. There are also some approaches, which uses retrieval to leverage the commit message generation, like [5], but I do not have enough time to implement it. In the literature I also found some collected datasets for the commit message generation task. The best one is CommitChronicle, presented in [1], as it contains commits for all the most popular programming languages. And furthermore, the amount of samples in it is ~ 8.5M, which is much more than I have collected. So, I decided to use this dataset for model training.

Data preprocessing: For the data preprocessing I performed the following steps:

- Removed the useless features from the dataset, like commit hash, author, etc.
- Initially code changes in the dataset were presented as a list dicts, where each dict contains the information about the code change in the commit for the specific file. I transformed this data into a single string, it represents the parsed version of the code changes for all the files in the commit.
- Add the special tokens to the code changes string, which indicates the addition, deletion of the code lines and the file name.

Model training: For the model training I used the codeT5+ model from the HuggingFace library. I used this model, as it's the most suitable for the commit message generation task. As we can interpret the commit message generation as the Neural Machine Translation task, where the code changes are the source language and the commit message is the target language. So, I needed some model which would have good information on the code. And the CodeT5+ model is the best one for this purpose.

First step before the model training was the tokenization of the data. I used the tokenizer from the CodeT5+ model for this purpose. As I have added some special tokens to the code changes string, I needed to add them to the tokenizer vocabulary and the model embeddings. Then I tokenized the code changes string and the commit message string and passed them to the model as an input and target respectively. Important thing to mention here is that during the commit message tokenization I need to change the padding token from the default one to some unknown token. It's needed to prevent the model from learning the padding token as a part of the commit message and calculating the loss on it.

The details of the training can be found in the previous report. It's stored in the GitHub repo `interim_reports/Project_report_3.pdf`.

Model evaluation: For the Commit Message Generation I highlighted the following metrics:

- **BLEU score** - it's the most common metric for the machine translation task. It's based on the n-gram precision between the generated and the original commit message. The main drawback of this metric is that it doesn't take into account the semantic meaning of the generated commit message. So, it can be high, but the generated commit message can be completely different from the original one. And the main reason that I used this metric is that it's used in almost all the papers, which are related to the commit message generation task. So, it's the most suitable metric for the model comparison.
- **BERTScore** - it's the metric, which is based on the BERT embeddings. It's also used for the machine translation task. The main advantage of this metric is that it takes into account the semantic meaning of the generated commit message.

Resulting metrics are the following:

As you can see, the model performance is quite good.

And comparing with the SOTA model, presented in [1] I got the following results:

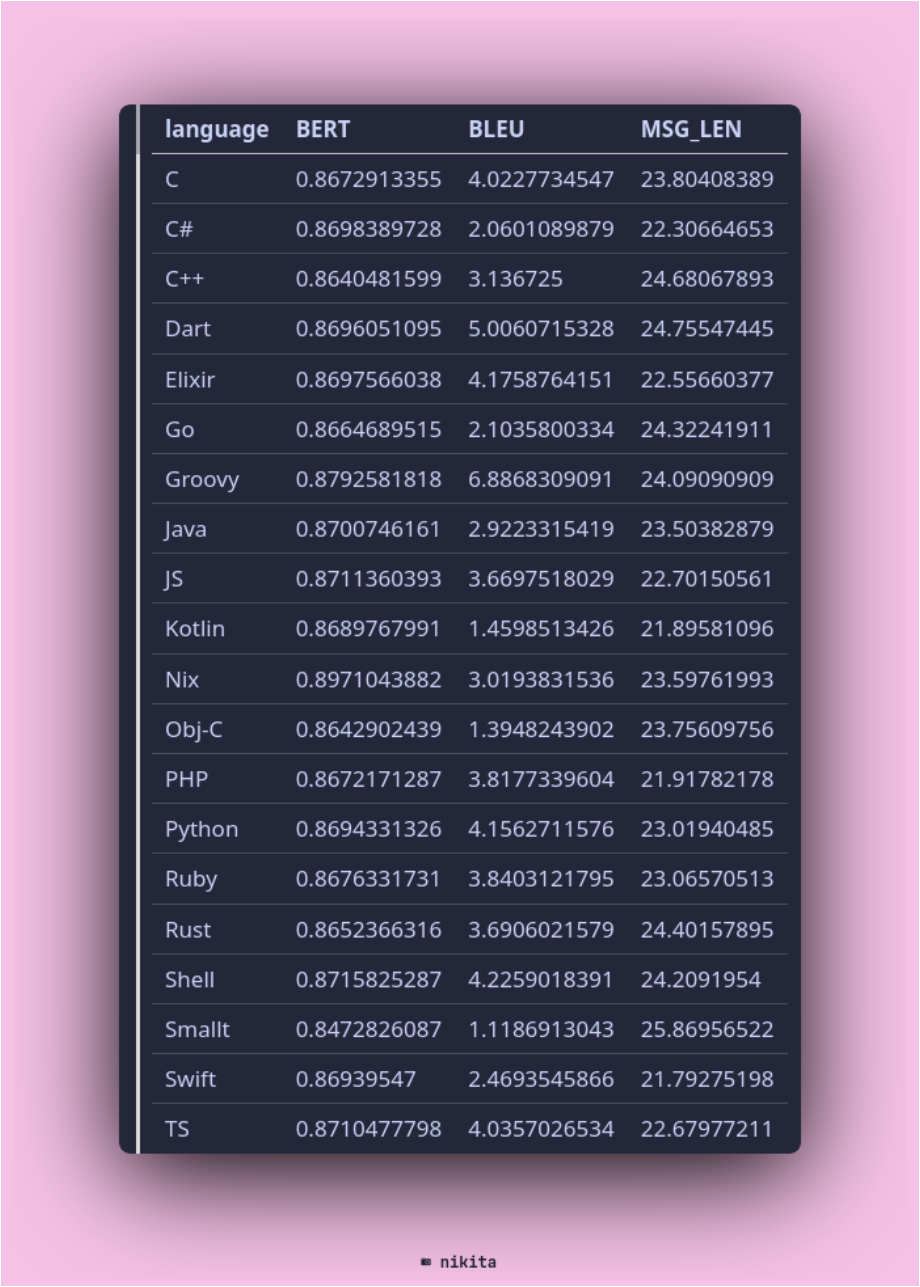
And the results of the evaluation in the custom dataset, which I parsed from github in the following.

Inference Model inference is implemented in the `Inference` folder. Or, another way to run it is to use the following command via the huggingface model card, which is publicly available. Link in the references list.

Conclusion Concluding all the work done, I can say that the model performance is quite good. And it can be used for the commit message generation task. However, there is a big room for improvement. Here is the list of the possible improvements: * Use the retrieval to leverage the model performance. As it's described in [5]. * The main drawback of the current solution is the limited size of the context window of the codeT5+ model, which is 512. So, some model architecture changes can be done to increase the context window size. And with the high probability it will increase the model performance. * Using some additional info from the repository apart from the code changes may give a model additional information to fit the style of the commit message of the certain repository.

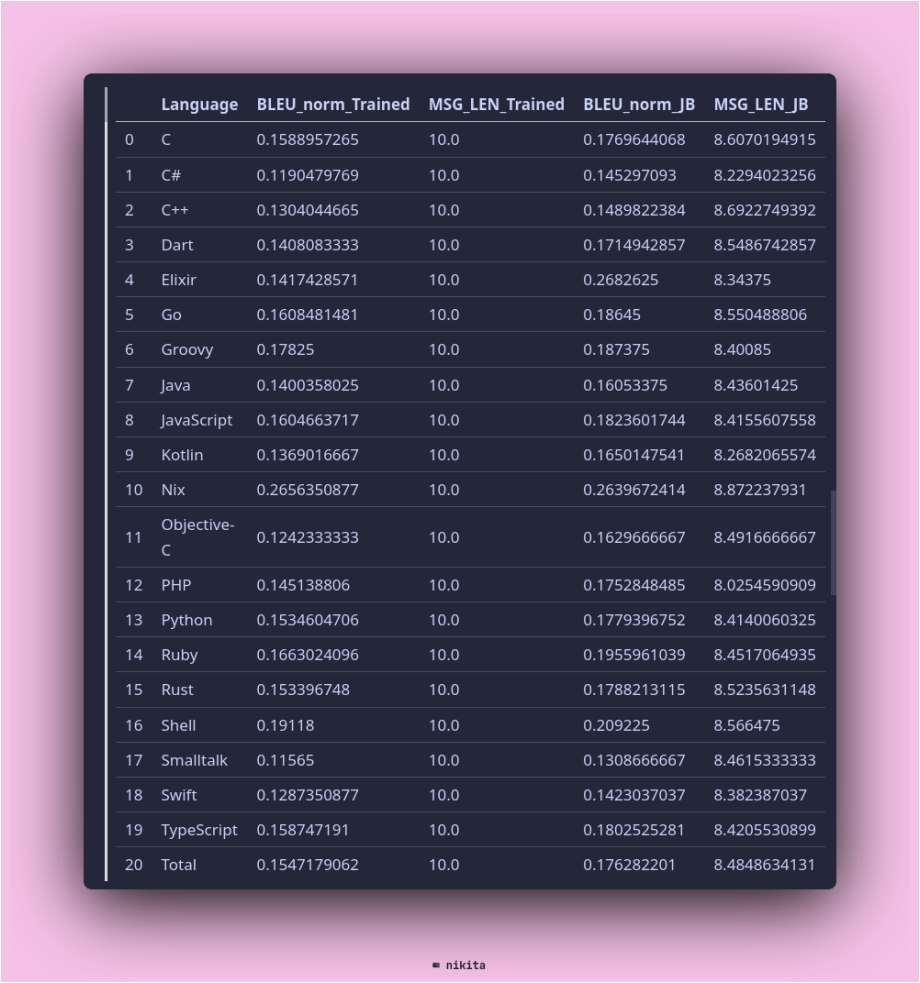
References

1) From commit message generation to history-aware commit message completion - link. In this article the authors proposed the CommitChronicle dataset, which is used in this project. They also proposed the model, which may be considered as the SOTA for the commit message generation task.



language	BERT	BLEU	MSG_LEN
C	0.8672913355	4.0227734547	23.80408389
C#	0.8698389728	2.0601089879	22.30664653
C++	0.8640481599	3.136725	24.68067893
Dart	0.8696051095	5.0060715328	24.75547445
Elixir	0.8697566038	4.1758764151	22.55660377
Go	0.8664689515	2.1035800334	24.32241911
Groovy	0.8792581818	6.8868309091	24.09090909
Java	0.8700746161	2.9223315419	23.50382879
JS	0.8711360393	3.6697518029	22.70150561
Kotlin	0.8689767991	1.4598513426	21.89581096
Nix	0.8971043882	3.0193831536	23.59761993
Obj-C	0.8642902439	1.3948243902	23.75609756
PHP	0.8672171287	3.8177339604	21.91782178
Python	0.8694331326	4.1562711576	23.01940485
Ruby	0.8676331731	3.8403121795	23.06570513
Rust	0.8652366316	3.6906021579	24.40157895
Shell	0.8715825287	4.2259018391	24.2091954
Smallt	0.8472826087	1.1186913043	25.86956522
Swift	0.86939547	2.4693545866	21.79275198
TS	0.8710477798	4.0357026534	22.67977211

Figure 1: Metrics results



	Language	BLEU_norm_Trained	MSG_LEN_Trained	BLEU_norm_JB	MSG_LEN_JB
0	C	0.1588957265	10.0	0.1769644068	8.6070194915
1	C#	0.1190479769	10.0	0.145297093	8.2294023256
2	C++	0.1304044665	10.0	0.1489822384	8.6922749392
3	Dart	0.1408083333	10.0	0.1714942857	8.5486742857
4	Elixir	0.1417428571	10.0	0.2682625	8.34375
5	Go	0.1608481481	10.0	0.18645	8.550488806
6	Groovy	0.17825	10.0	0.187375	8.40085
7	Java	0.1400358025	10.0	0.16053375	8.43601425
8	JavaScript	0.1604663717	10.0	0.1823601744	8.4155607558
9	Kotlin	0.1369016667	10.0	0.1650147541	8.2682065574
10	Nix	0.2656350877	10.0	0.2639672414	8.872237931
11	Objective-C	0.1242333333	10.0	0.1629666667	8.4916666667
12	PHP	0.145138806	10.0	0.1752848485	8.0254590909
13	Python	0.1534604706	10.0	0.1779396752	8.4140060325
14	Ruby	0.1663024096	10.0	0.1955961039	8.4517064935
15	Rust	0.153396748	10.0	0.1788213115	8.5235631148
16	Shell	0.19118	10.0	0.209225	8.566475
17	Smalltalk	0.11565	10.0	0.1308666667	8.4615333333
18	Swift	0.1287350877	10.0	0.1423037037	8.382387037
19	TypeScript	0.158747191	10.0	0.1802525281	8.4205530899
20	Total	0.1547179062	10.0	0.176282201	8.4848634131

Figure 2: Metrics comparison

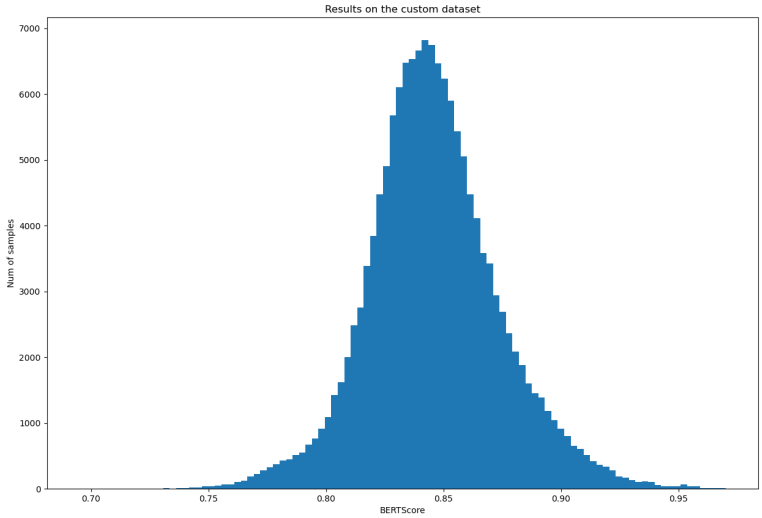


Figure 3: Score on custom data

2) Commitbert: Commit message generation using pre-trained programming language model - link - In this article the authors proposed the CommitBERT model, which is based on the CodeBERT model. They also proposed the dataset for the commit message generation task.
3) Commitbart: A large pre-trained model for github commits link - This is an improvement of the previous work. The authors proposed the CommitBART model, which is based on the BART model.
4) Fira: fine-grained graph-based code change representation for automated commit message generation link - authors of this work used the Graph Neural Networks for the commit message generation task. They consider AST of the code changes as a graph and use it as an input for the model.
5) Race: Retrieval-augmented commit message generation - link - In this article the authors proposed the model, which uses the retrieval to leverage the commit message generation. This technique may be used to improve the model performance.
6) CodeT5+ - Model which was used for the commit message generation task.
7) CommitChronicle - Dataset, which was used for the model training.
8) Fine-tuned model - My fine-tuned model, which can be used for the commit message generation task.
9) CodeT5+ for CMG by Jetbrains - Model, which was trained by the authors of the CommitChronicle dataset. In the final table I compared my model with this one.