Project_report_3

Team members: Sergeev Nikita BS20-AI n.sergeev@innopolis.university Github link

---

Missing part from the first report

Motivation: The task of automatically generating commit messages is the perspective research topic. Writing concise commit messages is a crucial task in the software development process. They serve as a valuable tool for developers to track changes in the project and to collaborate effectively. Despite this, a significant proportion of commit messages in open source are underinformative. State-Of-The-Art approaches to commit message generation (CMG) show promising results. However, most current research proposes methods that involve training encoder-decoder transformer-based models for this task. And one of the limitations of current CMG methods is the inability to handle the full context of code changes for the big commits, due to the limited context window of the model.

Project description: This project consist of the following parts:

1. Data collection. Parsing git diffs from github with the corresponding commit message. Filter the cellected data and transform it to the appropriate format for the Deep Lerning models
2. Find the appropriate model for the Commit Message Generation task and Fine-Tune it on the collected data
3. Find the appropriate metrics for commit message generation, taking into account semantic and syntactic similarity to the original commit message.
4. Evaluate the trained model and compare it with the existing solution for the Commit Message Generation(CMG) task.

Project progress:

Dataset selection

After collecting the dataset of Commits (from the previous report) I found out the CommitChronnicle - dataset collected from JetBrains research from this paper

TABLE II
NUMBER OF COMMITS IN THE OBTAINED COMMITCHRONICLE DATASET FOR EACH PROGRAMMING LANGUAGE. LANGUAGES ARE SORTED BY THE TOTAL NUMBER OF CORRESPONDING COMMITS.

Language	Train	Validation	Test	Total
Python	1,330,155	212,563	247,421	1,790,139
JavaScript	1,076,877	169,502	229,720	1,476,099
Java	952,162	200,035	204,862	1,357,059
TypeScript	936,697	177,258	198,272	1,312,227
C++	830,683	201,716	123,725	1,156,124
Go	672,045	133,954	134,699	940,698
C#	425,642	84,708	65,528	575,878
C	309,153	57,970	38,340	405,463
Rust	240,037	60,788	45,167	345,992
Ruby	181,916	39,912	33,433	255,261
PHP	178,556	32,293	36,618	247,467
Kotlin	154,276	29,781	28,021	212,078
Shell	117,927	13,902	27,019	158,848
Swift	101,274	28,227	8,938	138,439
Nix	2,526	86,022	8,108	96,656
Groovy	23,262	1,745	38,799	63,806
Dart	42,061	17,527	2,895	62,483
Elixir	41,562	3,380	5,874	50,816
Objective-C	32,517	1,294	7,708	41,519
Smalltalk	10,130	1,465	1,120	12,715
Total	7,659,458	1,554,042	1,486,267	10,699,767

As you can see from the dataset description table above, this dataset is much larger than mine, which was manually parsed from GitHub. Also, it doesn't include just Python commits, but commits with almost all of the most popular programming languages. So I decided to use this dataset to train my model. As with it, the model will be much more general in generating descriptive commits. And my dataset can be further used for evaluation to measure the model performance on the out of the dataset samples. With this we can check, how metrics differs in the val data from the CommitChronicle

Model Selection:

- **Efficiency**: This model boasts a relatively lightweight structure with 220 million parameters. Consequently, its training and inference processes are considerably faster when compared to Language Models (LLMs) with several billion parameters.
- **Pretraining on Extensive Code Data**: The codeT5+ model has been pre-trained on an extensive dataset of code, equipping it with a wealth of information about various programming languages (PLs). Given that our task essentially involves translating code differences into natural language text, the model's existing knowledge of code is a significant advantage.
- **State-of-the-Art Transformer Architecture**: Transformer-based models have established themselves as state-of-the-art performers in a wide range of NLP tasks. By leveraging this advanced architecture, we can harness the model's cutting-edge capabilities to excel in the CMG task.
- **Encoder-Decoder Model**: The decision to utilize an Encoder-Decoder model, rather than a Decoder-only architecture, is rooted in the task's similarity to Neural Machine Translation (NMT). In the CMG task, we effectively translate code differences into human-readable

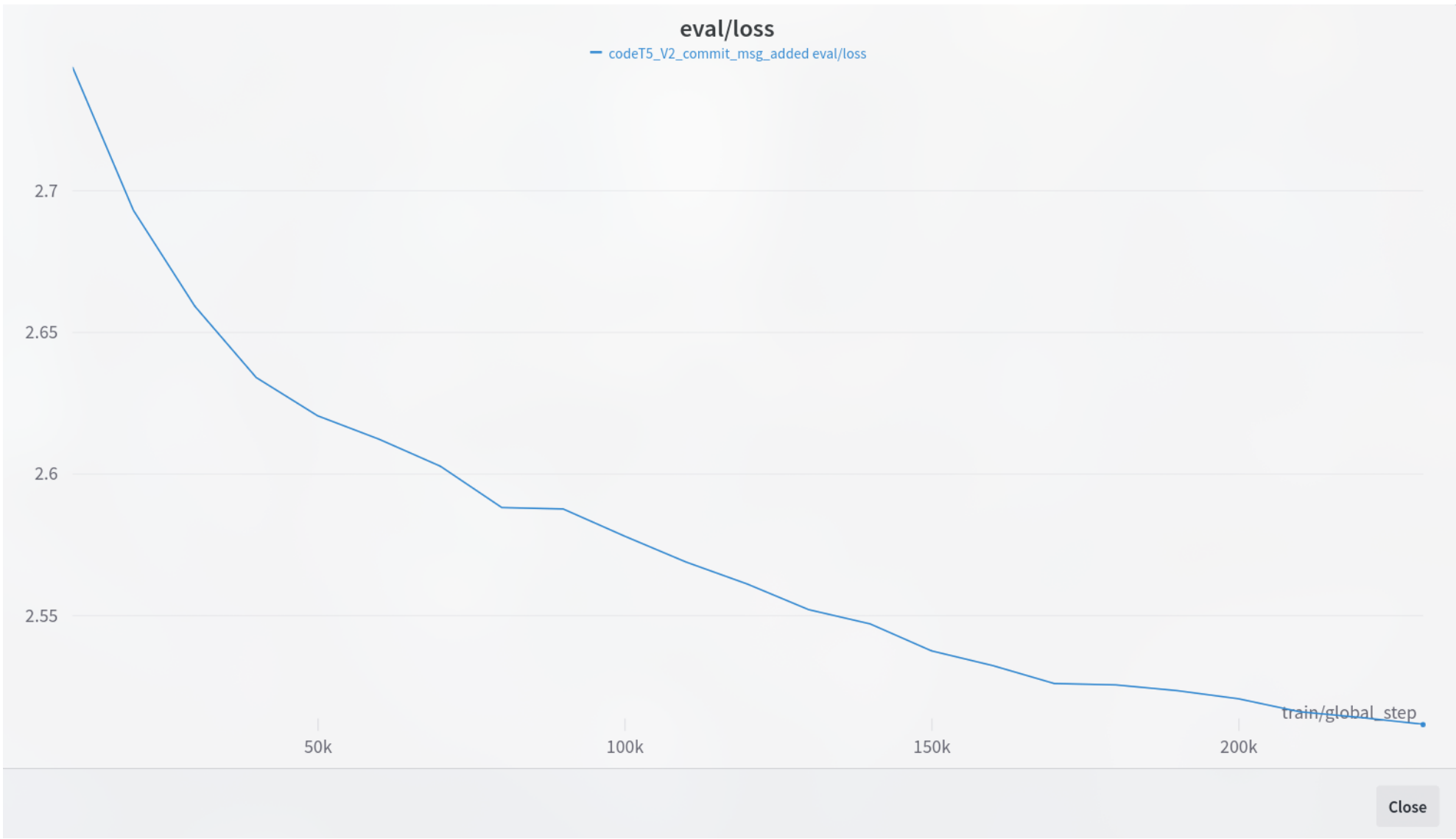
commit messages, making it pertinent to utilize both Self-attention and Cross-attention mechanisms. This approach enhances the model's ability to understand and generate contextually appropriate commit messages.

Model training:

Training of the codeT5+-220M performed with the following hyperparams:

```
train_args = transformers.Seq2SeqTrainingArguments(
    f"checkpoints_v3",
    evaluation_strategy = "steps",
    logging_strategy='steps',
    logging_steps=10000,
    learning_rate=2e-5,
    weight_decay=0.01,
    save_steps=75000,
    save_total_limit=4,
    num_train_epochs=1,
    predict_with_generate=True,
    report_to='wandb',
    bf16=True,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    optim="adamw_torch",
    warmup_steps=100,
    run_name='codeT5_V2_commit_msg_added',
    generation_config=gen_config,
)
```

Loss of the validation set during the training was the following (screen from wandb)



Model and tokenizer are available via link in Huggingface - https://huggingface.co/narySt/codeT5p_CommitMessageGen

Model inference:

Inference of the model performed with the following hyperparams:

```
sample_outputs = model.generate(
        **input,
        max_new_tokens=25,
        top_k=50,
        num_return_sequences=seqs,
        num_beams=5,
        no_repeat_ngram_size=2,
        do_sample=True,
        early_stopping=True,
        top_p=0.95,
)
```

To make Inference easy to use I implemented the script which does the following:

1. Takes from user Link to the commit From GitHub
2. Via GitHub API transforms gets info about this commit by link and transoforms it to format, which will be appropriate for the model
3. Generate model output for the given commit

Example of usage:

```
(commit-generation) ➜ get_github_data git:(master) ✗ python commit_info_from_url.py https://github.com/django/django/commit/4e790271e3e65c9ad037b347a34fa95e11982228
https://api.github.com/repos/django/django/commits/4e790271e3e65c9ad037b347a34fa95e11982228
Special tokens have been added in the vocabulary, make sure the associated word embeddings are fine-tuned or trained.
GENERATED COMMIT MESSAGES:
0: Add Denial-of-service possibility in django.utils.text.Truncator
--------------------------------------------------------------------------------
1: Add Denial of Service possibility in django.utils.text.Truncator
--------------------------------------------------------------------------------
2: Add Denial of Service possibility in django.utils.text.Truncator.
--------------------------------------------------------------------------------
3: Add Denial-of-service possibility in Django's security process.
--------------------------------------------------------------------------------
4: Add Denial-of-service possibility in Django's security process
--------------------------------------------------------------------------------
ORIGINAL MESSAGE:
Added CVE-2023-43665 to security archive.
```

Further steps:

1. Find an appropriate metric for measuring the performance of the trained model.
2. Comapre the resulting model performance with the Existing solutions
3. Probably, try another approach to generate Commit Messages (other model architecture) And compare it with the existing solution
4. Try to improve model performance with usage of some advanced techniques (Additional info the model input, Retrieval based improvements ...)