Chapter 3

Methodology

Literature Review chapter (2) defines current approaches to solve the task of Commit Messages Generation**(CMG)**. It includes deep learning models, used for generating messages, datasets constructed for this purpose, and metrics typically used to evaluate the model performance. In this chapter, I'm aiming to describe the steps I took to achieve the goal from the Introduction chapter (1). Section 3.1 describes the overall structure of conducted experiments and precisely defines the goal of the work.

3.1 Experiment design

The main objective of this study is to develop a system for automatically generating commit messages that are competitive with the SOTA methods described in the literature. To this end, I first need to construct my dataset. The next step is to analyse the retrieved data as well as data from other works. This step is essential for a deeper understanding of the data structure and the extraction

of relevant statistics. This analysis also includes filtering the data to ensure the quality of the parsed data. The next step of my research is a detailed analysis of the metrics used in the Neural Machine Translation(**NMT**) tasks. To use these metrics to evaluate my models, I first need to understand the methods used to calculate them and the intuition behind them. The next step is to train different models. This will include a concrete description of each model architecture and the conceptual proof of why this particular approach will succeed in the CMG task.

3.2 Data retrieval

The first step I took in my work was to get the data from open sources. In the task of the automatic commit messages generation, the data to train or evaluate the models should be in the format of labelled pairs code changes and the corresponding commit message.

3.2.1 Retrieval process

To get the data I decided to parse the most starred GitHub[1] repositories which were mostly written in Python programming language. For this purpose, I first get the names of the repositories and links for them via GitHub API. One way of getting the commits content from the repository is directly using the GitHub API, but due to the API calls limit I decided to do this in another way. With the use of the GitPython[2], which is a python library used to interact with git repositories I firstly cloned all the repositories from the retrieved list into my local machine and

[1] GitHub: Cloud storage for code projects.
[2] Description of the library

then fetch the information about all the commits from the .git directory.

3.2.2 Retrieved components

The main components of the data sample are the code changes and corresponding commit messages. However, for further research, I included some additional fields in my dataset. At the end of the data parsing process, I came up with the following features:

1. Name of repository - Name of the project in which the current commit where added. It might be helpful for the association of the commit with its project.

2. Commit message - label for the prediction that will be used by the model in the training phase

3. Commit changes - input data for the model.

4. Number of changed files in the commit - will be used to get the statistics about the retrieved data

5. Length of code changes in chars - will be used to get the statistics about the retrieved data

6. Hash of the commit - Unique identifier for the current commit to rid of the possible duplicates

3.2.3 Structure of the data

Before passing data to the Deep Learning model we first need to preprocess it. For my task, I decided to add some special tokens to the code changes and

commit messages. I used the following special tokens:

- <file_name> for name of the file before and after the commit
- <code_del> for code lines deleted in the commit
- <code_add> for code lines added in the commit
- <commit_msg> for the commit message

These special tokens are used to separate the most important parts of the input. Tokens described above are used at the beginning of the line, and the opposite with a backslash is used to signify the end of this part. The final format of the model input data is the following:

<file_name> old file name </file_name>
<file_name> new file name </file_name>
<code_del> deleted code </code_del>
<code_add> added code </code_add>
<commit_msg> commit message </commit_msg>

3.3 Collected data analysis and filtering

After collecting the data from GitHub I have ~1.2 million commit samples from 311 different repositories. Fig 3.1 shows the distribution of the number of commits among the repositories I took from GitHub. It does not include some data outliers, where the number of commits is too big. From this histogram, we can conclude that the mean number of commits in each repository is ~3000, which is quite a lot. Given the fact that each repository has its topic and commit messages

style we can make a conclusion that the collected dataset is representative enough in terms of different styles of writing commit messages.

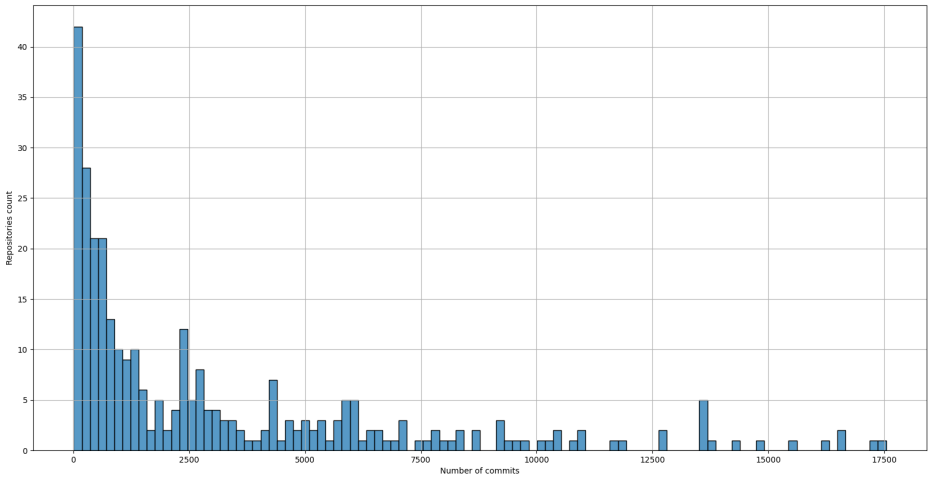

Fig. 3.1. Distribution of the number of commits in the repositories.

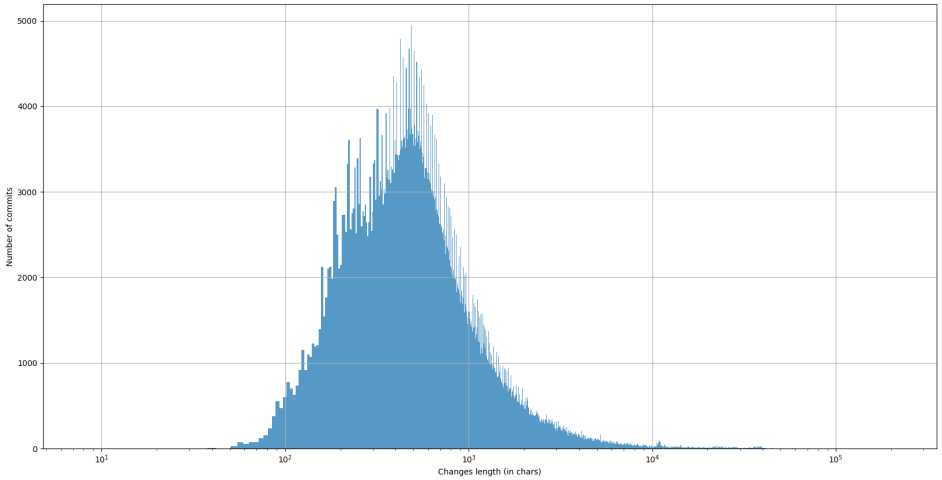

Fig. 3.2. Distribution length of the commits in the dataset.

Fig 3.2 represents the distribution of the code changes in commits. From

this distribution, we can see that the average length of the code changes is ~5000 characters. From this, we can conclude that the code changes on average is a very long sequence. Most of the modern Deep Learning models show bad performance on long sequences, so that might be a problem in future. Also in this step, I decided to trim my dataset, to have only samples with less than 6000 characters. With this truncation, we lose only 15% of the data but have much shorter sequences on average.