- Samples without Python code changes.

The filtered dataset I got in the end is made of ~300 thousand commits from 170 repositories.

3.4 Analysis of datasets from other works

In the literature review chapter (2) I described most of the existing datasets for the commit messages generation task. But in this section, I would like to focus only on the CommitChronicle dataset, presented in [9]. It consists of 10.7M commits in 20 programming languages from 11.9k GitHub repositories. It also includes not only code changes with the corresponding commit message, but lots of metadata about the commit, including the hash of the commit, commit date and time, and language which was used in the.

Regarding the filtration stage, CommitChronicle utilizes the following strategy: Authors of this dataset dropped examples out of the $[5\%, 95\%]$ percentile range of the length of code difference and samples with more than 16 files changed. They also get rid of the commits with non-ASCII symbols in the message, merge and revert commits, and samples with trivial messages [20], which does not provide any useful information about code updates.

For this dataset, I performed the same analysis to compare it with my data. Figure 3.4 displays the distribution of files changed within a single commit. Comparing CommitChronicle with my dataset in this regard we can observe, that the dataset presented in [9] has more balanced data. The number of samples decreases uniformly with an increasing number of files, at the same time my dataset has some outliers. Fig 3.5 represents the distribution of code changes length in characters. From this side, the statistics are similar, so we can say that

datasets are the same from this perspective. The distribution of the number of commits among the repositories from the CommitChronicle is shown in Fig 3.6. In this figure, I displayed only information for the repositories with less than 5000 commits to make the plot more representable. From this side, CommitChronicle has more balanced data, as the distribution is smoother, but it is mostly connected with the much fewer repositories in my dataset. The last thing I would like to mention about the statistics of the CommitChronicle is the most popular first words of commit messages. They are presented in the Table 3.3. As was mentioned above, merge commits were filtered out by the authors, and in all other respects, the results are the same as for my dataset. So datasets are identical from this side.

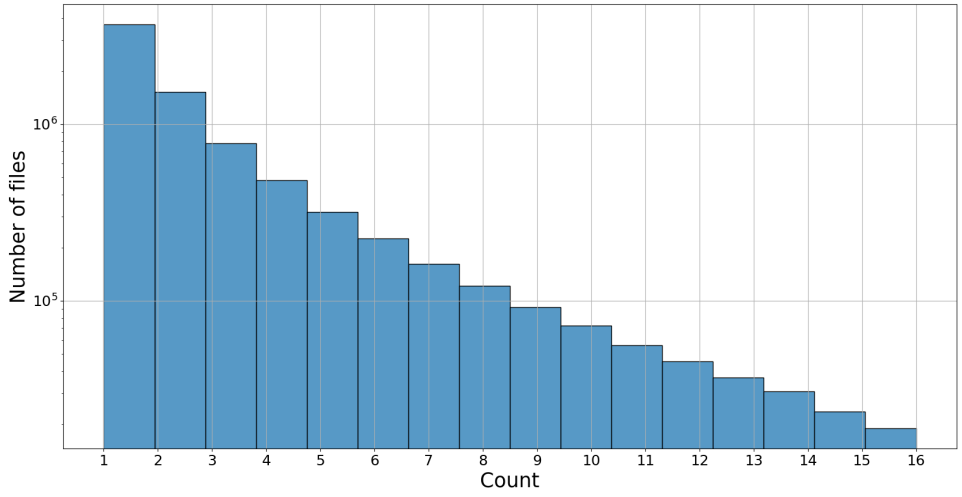

Fig. 3.4. Distribution of the number of files in commits in CommitChronicle.

TABLE 3.3
Distribution of commit messages

Commit Message	Count
Add	541205
Fix	427753
Update	251594
add	198032
fix	177001
Added	158065
Remove	157336
fix:	118704
Use	93908
Fixed	89186

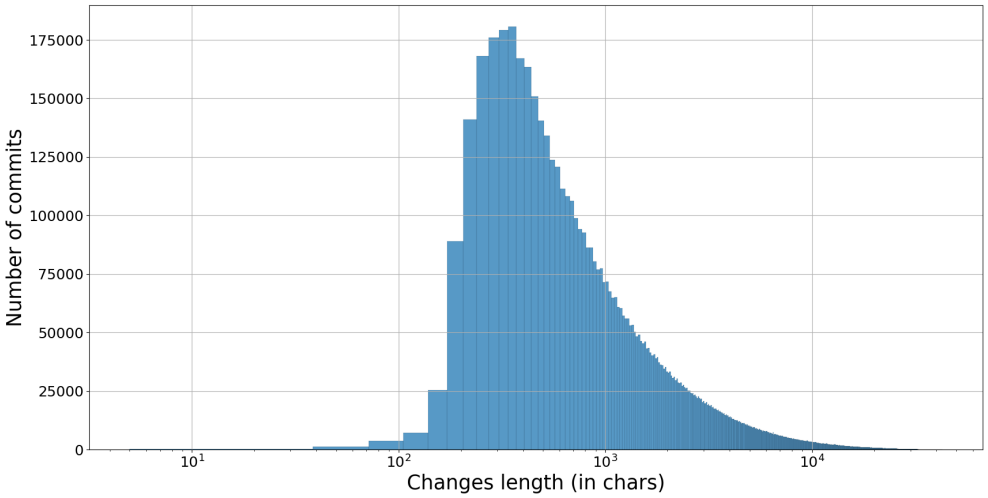

Fig. 3.5. Distribution of length of commits in CommitChronicle.

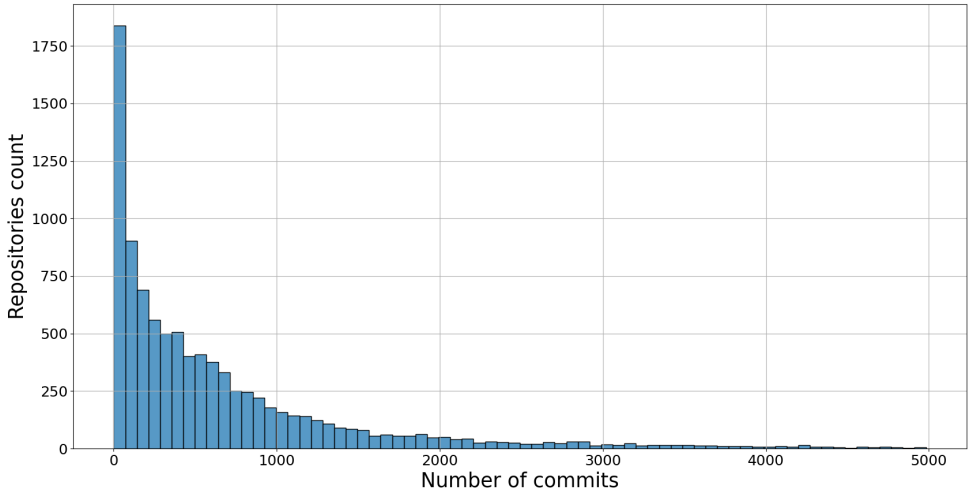

Fig. 3.6. Number of commits in the repositories in CommitChronicle.

3.5 Conclusion about the data

In this section, I would like to make the choice of the data used for the model training, evaluation and testing. From the data analysis, I got that CommitChronicle has a better representation of the data. It also has much more samples. So, it will be logical to use CommitChronicle as a train set. As a validation set, I will use validation split of the CommitChronicle to be sure that I have no repetitive samples with the train data. For the test of my models, I will utilize both my custom dataset and the test split of CommitChronicle. The results of the models in the custom dataset may be represented as the results of out-of-domain data, as the process of getting the data differs, and some samples filtered out from CommitChronicle may be included. So, this testing will examine the fair performance of the model in the ‘real world data‘.