Exploring Github Issues using Natural Language Processing.

- Divyateja Pasupuleti - 2021A7PS0075H
- Kumarasamy Chelliah - 2021A7PS0096H
- Bhaarath K - 2021A7PS1816H

Introduction:

A typical problem developers face nowadays is that there is no consistent way to keep track of their work on GitHub. We propose using Natural Language Processing to process multiple commits for a given time frame and convert the same into a changelog, which the developing team can use for the release notes. This will reduce the hassle of manually going through all the commits and make it easier for the developer to generate monthly reviews.

A long-term goal would be to develop a system capable of converting an issue into multiple minor issues or suggest files per issue. This would streamline the development process and make it easier for open-source developers to find and fix minor issues.

We see open-source events happening all around us. Let us take Hacktoberfest[1], an event where open-source developers worldwide try to develop and help solve issues in open-source projects from organizations. This event also tries to be a place for many students and other developers to start their journey into development. In the open-source community, one major thing that influences all this is the concept of first-good-issue.

Most developers tend to create issues with minor fixes that inculcate the habit of sending pull requests on GitHub. A pull request on GitHub is the concept of changing a part of the code and suggesting it be merged into the main code. The issue we are trying to solve is to figure out and break significant issues into minor issues for developers independent of each other to work on them without deep knowledge of the other issues, thereby creating multiple first-good issues.

Previous Work:

GitHub commits play a crucial role in software development as they record code changes with natural messages for descriptions. These descriptions give the developers a brief understanding of what has been altered and act as a refresher in case any of the developers forget something in the future.

CommitBART(Shangqing Liu et al., 2023) is pre-trained using three categories of objectives: denoising tasks, cross-modal generation, and contrastive learning, which encompass six pre-training tasks. These tasks help the model learn to commit fragment representations. The authors introduce a "commit intelligence" framework with one understanding task and three generation tasks for commits.

However, CommitBERT(predecessor of CommitBART, Jung, 2021) did not pre-train a commit-specific model on the commit data, limiting the performance on commit-related downstream tasks. The paper highlights the need for various pre-trained models for generating code; it was more focused on generating code for a specific commit message. The attention to modified code by software developers can be significantly influenced by the number and nature of their daily activities. This is compounded by the frequent interruptions that developers face during their work. These daily activities can contribute to developers overlooking or forgetting

the implementation details behind their code changes by the time they commit their code. Furthermore, identifying and recalling the precise set of changes made during a commit can be challenging and costly, particularly for substantial changes that span multiple code packages, classes, methods, configuration files, database schemas, and other related artifacts.

ChangeScribe(Fernando, et al.,) aimed at generating commit messages using the changes made in the code and the files altered. ComSum(Choschen, 2021) primarily aims at text summarizing while documenting commits, gathering those to work summarizing datasets.

Most previous research on GitHub commits was primarily based on generating meaningful descriptions or code based on descriptions. We plan to find similarities between files based on commits and use the similarity obtained to suggest files related to a specific issue based on their description. Another goal is to be able to summarize all the commits within a specific time frame to give the developer a rough understanding of the developments made during that time frame and also help in generating change logs.

Task Definition:

Using LLMs to create a beginner friendly github experience

Dataset:

We select one open-source GitHub repository for the pre-processing dataset and retrieve their lists of issues and commits. We check all the files modified in a particular commit and use a skip-gram model to relate two files of a single commit together, thereby reducing all files into a lower-dimension embedding space. We also use the same data to cluster files and find correlations between files.

In the second part, we split one issue into multiple minor issues. Our proposed idea is to take input from the user, all the files that could be involved in solving the problem. We then find the k-nearest neighbors using the embedding space vectors and split the one single issue into multiple parts, each dealing with a single file.

Sample Dataset - File Pairs:

```
github/__init__.pyi,github/MainClass.py
github/__init__.pyi,github/MainClass.py
github/__init__.pyi,github/__init__.py
```

Sample Dataset - Commits Per File:

```
{
        "fileName": ".github/workflows/label-conflicts.yml",
        "commits": [
            "CI: remove conflict label workflow (#2669)",
```

```
            "CI: Increase wait and retries of labels action (#2670)
Co-authored-by: Enrico Minack <github@enrico.minack.dev>",
            "CI: update labels action name and version (#2654)
Co-authored-by: Enrico Minack <github@enrico.minack.dev>",
            "CI: label PRs that have conflicts (#2622)"
        ]
    }
```

Sample Dataset - Issues:

```
{
        "title": "Add `GithubObject.last_modified_datetime` to have
`last_modified` as a `datetime`",
        "body": "As nearly all dates are returned as `datetime.datetime`,
it would be convenient to have `last_modified` as a `datetime.datetime` as
well.dAs discussed below, let's introduce `last_modified_datetime` new
argument to cover this need."
    }
```

Methodology, Models and Loss Functions

We split this section into multiple parts, each dealing with a different part of the task ahead. We start off with finding the file name embeddings.

Section 1 | Skip gram based File name Embeddings

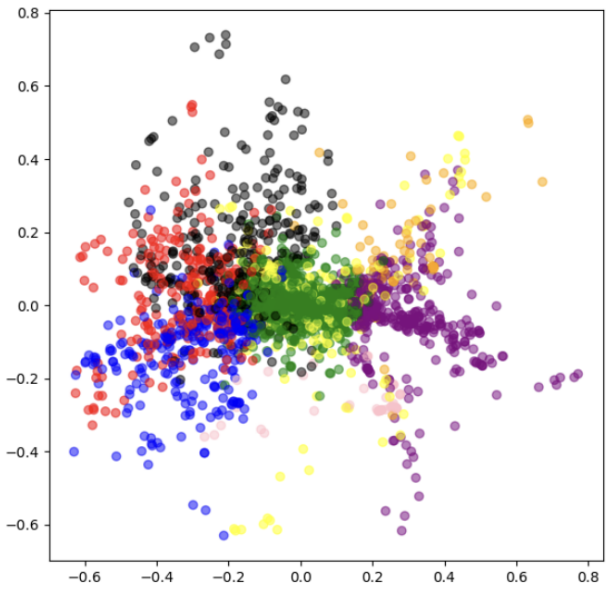
We start with a skip gram based implementation which gives us the embeddings for each file in the file correlation vector space. We generate the data from every commit to group similar files together. After getting over 8 lakh file pairs in total from over 2000 commits, we use this data to train the skip gram model. The goal of this model is to ensure we accurately determine which 2 files are related to each other.

For the training purposes, we use the **Adam Optimizer** and the **Cross Entropy Loss function**. We represent every file using a **128 dimension vector** and use a batch size of 512 over about **50 epochs** to train the model. These are the hyperparameters used. We save the least loss model and use this model to generate embedding vectors whenever needed in the further parts.

```
SkipgramModel(
  (embedding): Embedding(860, 128)
  (linear): Linear(in_features=128, out_features=384, bias=True)
  (linear2): Linear(in_features=384, out_features=860, bias=True)
)
```

We report the results of the skip gram model in terms of accuracy of the model to predict file pairs. We use a train test split of 80-20 and try to check the loss

Epoch	**Loss**
1	0.01069943634037449
25	0.01061809078890714
50	0.010604893165702286



Section 2 | Commit Summarization

We use the 'tuner007/pegasus_summarizer' model and its tokenizer to summarize commit messages. When working in large teams, it could become hard for the team leader to keep track of what all changes are happening within the code base. Summarizing commits into a short one line message could help the developer in **creating change logs** as well as keep track of what all is being modified. We generate **commit message based summaries for the last 15 commits** tokenizing, padding, and generating summaries using the Pegasus model with specified parameters such as the max length of input, max length of output, temperature etc.

We have used a pre-trained model for this part of the project. This model is a version of Google's PEGASUS model fine tuned for summarization. This model is **based on BERT** and the transformer models.

We also use the same model to create **developer reports on a per user basis** and summarize them for the team lead.

```
pypi-release workflow, allow for manual run (#2771), Merge
StatsContributor.pyi back to source (#2761), Merge Repository.pyi back to
source (#2749), Merge /Organization.pyi back to source (#2744) and
PullRequest.pyi back to source (#2743). Merge StatsContributor.pyi back to
source (#2761), Merge Repository.pyi back to source (#2761), Merge
Repository.pyi back to source
```

Section 3 | Sentence Similarity

The next step was to summarize code of every file into short paragraphs, but existing models and methods turned out to be very heavy due to the overhead of processing the code and then extracting information from it. We propose an alternative where we perform sentence similarity between commits, function names and issues and then try to link the files with the commit for this issue.

Commit-Issue Similarity

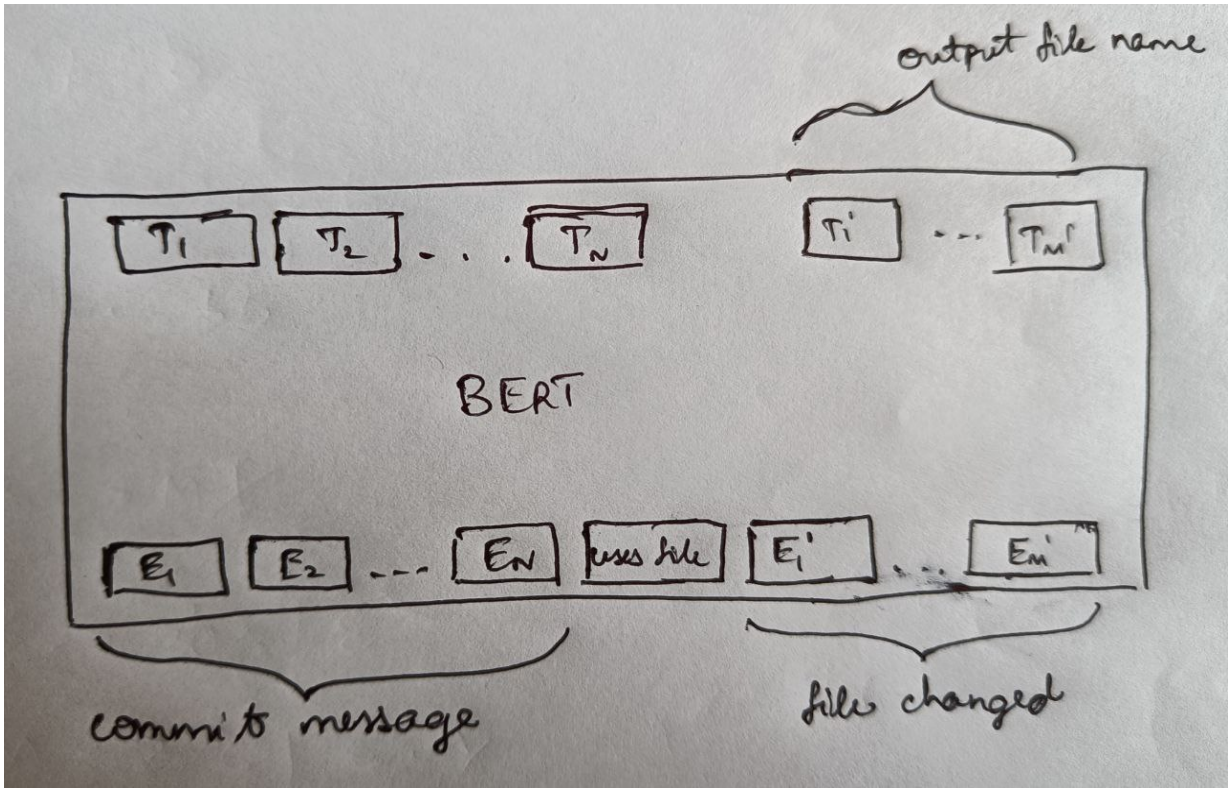
We used sentence transformers to encode all commit messages, now if given an issue by the user we can encode it and find cosine similarity to all the encoded commit messages and get the most related commits. From those commits we extracted the files that need to be changed.

File-Issue Similarity

We've also tried extracting all the functions in a file and concatenating them and then using sentence similarity to find out the file closest to the issue (a well written issue has keywords related to the feature changes required). This model predicted 50-60% of the correct files.

Section 4 | Bert - Extractive File Search

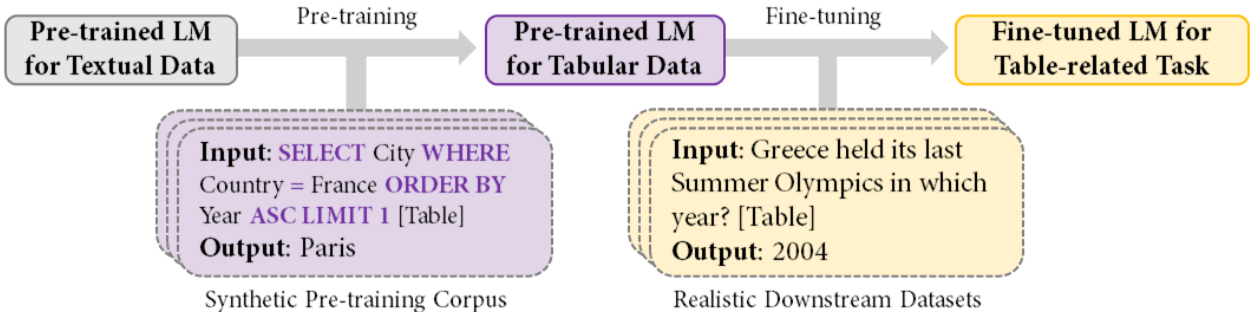
After some experimentation with the previous methods, we have come to the conclusion that it might not always be possible to figure out the files using sentence transformers. In order to increase the accuracy of the model, we shift towards a **BERT Question Answering** based system where we give the answer paragraph as a list of all files that are in the commit(s) corpus which now includes the commit name, files modified and the author of the commit.



Section 5 | Table Pre-training via Execution based Modeling

We explore a new concept called Table Pre-training via Execution where we put all the commits into a table like structure and train the model to answer questions based on this. For this we use Microsoft's **TAPEX Model.** This model has a limitation of 25 rows processing at a single moment, so we take them separately and rank the outputs and try to take the top 5 files.

This model uses Natural Language processing to convert our queries into SQL type structures and then give the result based on running that SQL query on the table given as input.



Future Goals:

We have displayed that the idea has scope for a lot of work and will continue working on this. We have listed out our methodologies and details regarding the flow which can be further enhanced and continued upon.

Bibliography:

1. (n.d.). Hacktoberfest 2023. Retrieved October 17, 2023, from https://hacktoberfest.com/
2. CommitBART: A Large Pre-trained Model for GitHub Commits https://arxiv.org/pdf/2208.08100.pdf
3. On Automatically Generating Commit Messages via Summarization of Source Code Changes https://www.cs.wm.edu/~denys/pubs/SCAM14-ChangeScribe-CR.pdf
4. ComSum: Commit Messages Summarization and Meaning Preservation https://arxiv.org/pdf/2108.10763.pdf
5. Evaluation of Commit Message Generation Models - https://arxiv.org/pdf/2107.05373.pdf
6. Table Pre-training via Learning a Neural SQL Executor - https://arxiv.org/abs/2107.07653
7. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding - https://arxiv.org/abs/1810.04805
8. MPNet: Masked and Permuted Pre-training for Language Understanding - https://arxiv.org/pdf/2004.09297.pdf