3.6 CodeT5+ training

My first attempt at approaching the task of commit message generation is straightforward. I will consider this task as a simple NMT (Neural Machine Translation) problem. In this setting, the model aims to translate the input code into a natural language description of the code changes. The current SOTA solutions in the NMT task utilize the transformer architecture approach presented in [1]. According to the specific domain of the input data for my task, the best choice is the model pre-trained on the massive amount of code. The model used as a backbone in the current SOTA solution of the CMG task presented at [4] is codeT5. The model was pre-trained on a variety of coding tasks and demonstrated proficient performance in code comprehension. However, since that time, the authors of CodeT5 have introduced an updated version of the model, called CodeT5+ [16]. This new model outperforms CodeT5 in performance across the majority of code-related benchmarks. So, I decided to train CodeT5+ on CMG task as the first experiment in my work. In this setting, I will use code difference from the commit as plain text.

As mentioned in Section 2.3, the most common method for generating commit messages involves representing both the input and output as a sequence of numerical tokens. In Paragraph 3.2.3, I described the specific format of the input and output data for my task. Therefore, before training the model, it is necessary to add special tokens to the tokenizer vocabulary. The model will learn the embedded representation of these special tokens during training. This step will aid the model in better understanding the structure of code changes.

$$L(y, \hat{y}) = -\sum_{i=1}^{C} \log\left(\frac{e^{\hat{y}_i}}{\sum_{j=1}^{C} e^{\hat{y}_j}}\right) \times y_i \tag{3.1}$$

The loss function that should be optimized by the neural network is cross entropy loss represented at 3.1. y here represents the true probability distribution of the next token choice from the tokenizer vocabulary in the target sequence. This is a vector with 1 in one position, and 0 in all the others, as we exactly know what token should come next. And $\hat{y}$ is the predicted vector of probability distribution to choose the next token. This function tends to make the predicted vector as similar to the true vector as possible, *i.e.* predict the next token correctly. Predicting the next token in an autoregressive manner model will generate a commit message for the given code edits.

3.7 Experiments with larger CodeT5+

To assess the impact of scaling the model in terms of parameters on the quality of generated commits, I trained the CodeT5+ model with 770 million parameters. The differences between the base version and this model are presented in Table 3.4. From this table, it is clear that the distinctions between these models are primarily in the hidden representations and the number of blocks in both the encoder and decoder parts. A larger model version does not process longer sequences but has better internal representation, potentially enabling the generation of better commit messages. The training objective I utilized and the data format are identical to those used for training the base CodeT5+. This experiment demonstrated how scaling the depth of the neural network and improving representation can affect model performance. Additionally, this phase allowed me to determine whether 220 million parameters are sufficient for constructing commit messages that are relatively simple sequences. The final aspect requiring analysis in this section is how this scaling influences the model's inference time since efficiency in terms of time is as crucial as message quality when considering real-world applications.

TABLE 3.4
Comparison of CodeT5+ with 220M and 770M parameters

Feature	**220M model**	**770M model**
Context window tokens	512	512
Hidden state dimension	768	1024
Encoder transformer blocks	12	24
Decoder transformer blocks	12	24
Embedding dimension	32100	32100

3.8 CodeT5+ with retrieval components training

One of the problem with the traditional language modeling approach to the task of commit messages generation is the shared style of messages within the same repository. For example, repository of linux kernel source code have a strictly specified format of commit messages. Using only code modificaions it is impossible to adapt generated commit message to have the same spcified format.

3.8.1 Retrieval approaches

To mitigate issue with a message style, works like [9], [5], and [17] adopt additinal retrieval module to the standard endcoder-decoder architecture. These methods are mostly uses the database of commits and corresponding commit message. And then new commit is given as an input to the model, this method firstly use encoder to get a embedded vector representation of code modifications. This representation is further used to find the most similar code changes from the dataset, and get both modifications and commit message. This additional information is then passed to the model to get the result. Analysis of these method shows, that retrieval mechanism leverages the results of commit generation.

One more way to use the historical data in generation of commit messages is to retrieve previous commits from the same repository, or previous commits of the same person. This method way used in the [4], and showed improvement of model performance. If previous methods are mostly used to improve the understanding of code changes, then this historical data retrieval method is used only for message style adaptation.

3.8.2 My experiment with retrieval

For my experiment, I decided on the method combining retrieving the most similar code modifications and getting historical data from the previous commits. My experiment is close to the one provided in [4] with a combination of the RACE method from [9] and the history of messages added to the model input. The main difference between my approach and the RACE is that the RACE leverage the commit generation by passing the combination of the input code changes with the most similar one from the database. At the same time, my approach is utilizing only the message from the most similar commit to adapt to the style of the commit message of the same code theme. Additionally, I pass to the model input history samples of the messages from the same repository. In the end, my input to the