A-RACE: Adjusted Retrieval-Augmented Commit Message Generation

Peiyi Jiang
Faculty of Computer Science
Dalhousie University
Halifax, Canada
PJ@dal.ca

Haoyi Zhang
Faculty of Computer Science
Dalhousie University
Halifax, Canada
Haoyi.Zhang@dal.ca

Jiahao Chen
Faculty of Electrical & Computer Engineering
Dalhousie University
Halifax, Canada
ch243852@dal.ca

***Abstract*—This research paper conducts experiments on the newly implemented Retrieval-Augmented Commit Message Generation (RACE) framework, attempts to improve its performance and addresses some of its limitations. Specifically, this research paper attempts to test the RACE framework's performance when using the CodeT5 and GPT-2 models, improve the framework by adding AST as the encoder, and expand the dataset to also contain and test on Go programming language. The results show that the original model still performs better that the CodeT5 and GPT-2 embedded model on 20 percent of the Python and Java dataset. However, the generated commit message is more readable even with a lower BLEU score. The results obtained after combining the AST encoder with the original encoder show a slight improvement in the framework's performance. In addition, RACE's performance on the newly created Go dataset shows an outstanding result when compared with the Java dataset. Overall, this research paper has potential impacts for helping better understand how the model can be improved in the future and addressed some preprocessing errors that were not discovered.**

Keywords—Commit Message Generation, RACE, GPT-2, CodeT5, Go, AST, MCMD

I. INTRODUCTION

With the improvement of machine learning in recent years, large language models become a useful tool in software engineering fields. Commit message generation is one of the important tasks. Writing a good commit message that follows the best conventions and covers the code changes is not an easy task and is time-consuming. So, there is much research about training a machine learning model to help developers to generate high-quality commit messages without putting much effort. Retrieval-Augmented Commit Message Generation (RACE) is one of the most recent commit message generation model training frameworks that outperforms all the other previous models [1]. Although it achieved significant success in training commit message generation models, it has some insufficient. In this report, we conducted three methods to improve RACE. First, we combined CodeT5 and GPT2 to generate more meaningful commit messages. Second, we leveraged a pre-trained model, UniXcoder [2] that encodes source code as an abstract syntax tree (AST) to analyze code structure, to enhance RACE'S understanding of code structure. Finally, we built a Go language dataset and trained a CodeT5 model using the RACE framework to expand its generality.

II. APPROACH

A. Model

In the original RACE framework, the encoder and decoder are from the same pre-trained model. In our research, we proposed a novel method that combines an encoder and decoder from different pre-trained models. Specifically, we use CodeT5 as an encoder in both the retrieval and generation phases and GPT-2 as a decoder in the generation part.

The reason why we select CodeT5 is that it is a general code understanding model that supports 8.35M functions and 8 program languages and achieves state-of-the-art performance on software engineering tasks [3]. We believe it can capture the most important features during the encoding phases. On the other hand, GPT is a type of model that is trained on a significant amount of English corpus [4]. It is good at generating text that is coherent and fluent. It already demonstrated its power in ChatGPT. So, we select GPT-2 as the decoder model in the generation phase of the RACE framework and expect it can generate human-readable commit messages.

The architecture keeps the same except for the decoder. GPT-2 accepts an input code diff and a similar commit message as its input and generates a commit message for the input code diff.

B. Abstract Syntax Tree (AST)

AST can help us to represent the structure of the programming source code hierarchically. Because of that, AST is more commonly used to analyze compilers, programming language theory, formal methods etc. Although the main purpose of our project is to generate the commit message. However, we also cannot ignore the generation is based on the analysis of code differences. According to the paper "*RACE: Retrieval-Augmented Commit Message Generation*" [1], we found RACE model didn't use any AST model even in the retrieval module or generation module. After the internal discussion, we decided to add one type of AST model into the code embedding process of the generation module. Dr. Tushar

Language	Training	Validation	Test
Java	160,018	19,825	20,159
C#	149,907	18,688	18,702
C++	160,948	20,000	20,141
Python	206,777	25,912	25,837
JavaScript	197,529	24,899	24,773

Fig. 1. Statistics of the MCMD dataset as illustrated in [1, Table 1]

Sharmar and Mr. Bowen Xu give us a useful introduction to the pre-trained AST model, UniXcoder. It is built and maintained by the Microsoft team. UniXcoder is a hypothetical unified cross-modal pre-trained model for programming languages, and is designed to support both code-related understanding and generation tasks [5]. UniXcoder can be used for several tasks, including code search, code completion, function name prediction, API recommendation and code summarization. We decide to use the mode for the code search part (Encoder-only Mode) to improve the RACE model. The first thing we do is to revise the decoding part of UniXcoder. We make it to suit the dataset in the RACE model. Next step, we revised the ECMG model function add the UniXcoder encode in it, and then combined the output generated by the UniXcoder model with the original code difference as input and the output generated by the CodeT5 model with the same input. Because of the time consumption, we didn't have a chance to absorb the results from the rest of the modes in UniXcoder. We guess it can improve more for the RACE model if we can build the new pipeline for UniXcoder into the RACE model. Meanwhile, we can use the encoding output from UniXcoder to calculate the similarity of input code diff and retrieved code diff, and then ensemble the existing similarity to get a more accurate score which means it can guide the commit message generation better.

C. *Dataset*

The original RACE framework uses the MCMD [6] dataset which contains five different programming languages (Java, C#, C++, Python, and JavaScript). For each of the programming languages, the dataset obtains the commit message, code diff, date, sha, and repo full name from GitHub repositories. Then, the dataset is filtered and separated to train, test and validate sets for each programming language as shown in Figure 1. RACE actions to represent the code diff. There are four action types added, they are <keep>, <insert>, <delete>, and <replace> each represents an action in the code diff. RACE stores this data as contextual data and uses it for training its model. In order to implement scripts to create a Go programming language dataset, we have to understand how MCMD gets all the commit information and preprocesses it. As mentioned in the papers [6, 8, 9], the steps for creating the dataset involve with collecting data, preprocessing, and filtering. Firstly, the data is collected from the top 100 starred

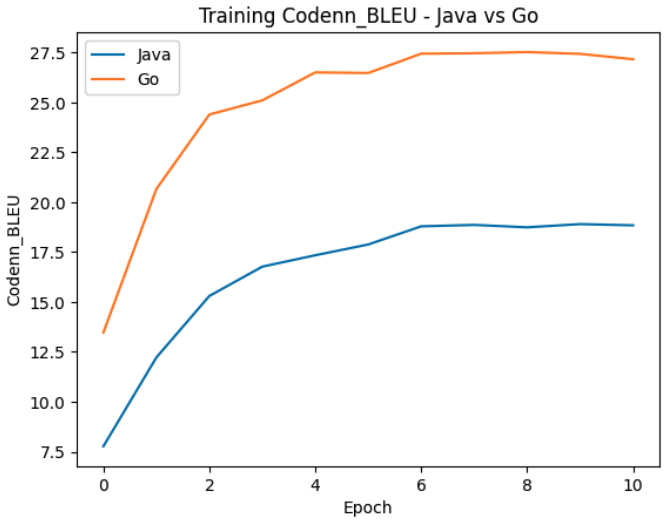

Fig. 2. Training Codenn_BLEU comparison between Java and Go

projects on GitHub before 2021. Then, the first sentence of each commit message is extracted and the further preprocess of this dataset by applying token-level change issue ids are moved to reduce the vocabulary size. Merge commits, rollback commits and large commits are also removed, and the commit message and code diff are split into tokens during the preprocessing. Finally, the data is filtered by length. Commits with code diff of more than 100 and commit message length of more than 30 are removed. In addition, the Verb-Directed Object (V-DO) [9] filter is used to identify the V-DO pattern. The commit messages that do not begin with a verb are also removed.

We implemented a script to create our Go dataset that adopts most of the steps MCMD used. The process of creating the Go dataset is described below. Firstly, a CSV file that has a list of reports are downloaded from GitHub Search by applying the filters: more than 10 contributors, more than 1000 stars, and a repo created after the beginning of 2021. Then, all the repos are downloaded, desired commit information is extracted, and all the commit data from all repos are combined and saved in a josnl file. Next, the first line of each commit message is extracted, the issue id is removed, and the message is tokenized for preprocessing the commit message. For preprocessing the code diff, the --- and +++ symbol are replaced by mmm and ppp so that the symbol will not be split into three tokens, and all the characters before the --- symbol are removed. All the tabs are removed and all words, punctuation, and symbols are separated by whitespace. After that, the newline characters are replaced by <nl> as this is the required step for converting the code diff to contextual data for RACE's preprocessing script. Also, all of the empty commit messages and code diff with longer than 512 tokens are all removed for filtering the data. Finally, the RACE preprocessing script is used to convert the data into contextual data.

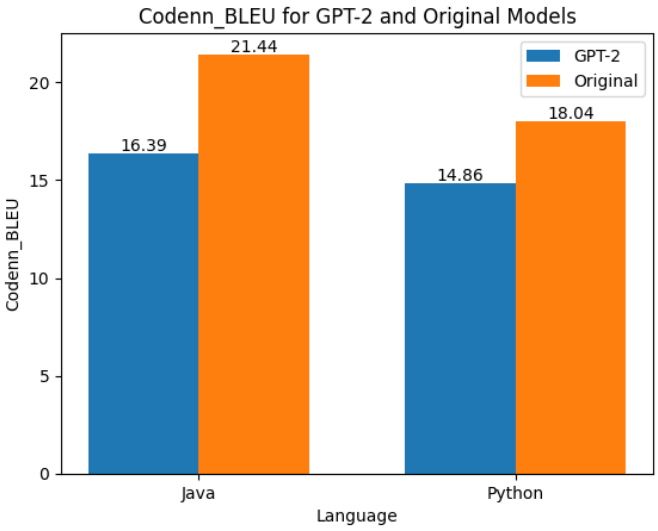

Fig. 3. Codenn_BLEU for GPT-2 and Original Models

III. EXPERIMENT RESULTS

Given the computing resources, we change the batch size from 32 to 4 and sampled 20% of the original RACE dataset. The rest hyperparameters are the same as RACE's setting.

A. *Model*

We trained the proposed CodeT5 + GPT-2 (C+G) model and the original model on Java and Python using one GTX 3070Ti. The training time for each programming language per model is around 12 hours (including both retrieval and generation phases). The generation part took the most time, around 8 hours. After comparing the BLEU scores in Fig. 3, we conclude that the C+G model performed around 20.5% worse on average than the original model on generating commit messages for Java and Python code difference. Through analyzing the training loss in Fig. 4, all training reached convergence around 100 evaluations. We can see the losses of C+G are higher than the original's, even though the difference is minor. Fig. 5 shows that BLEU scores

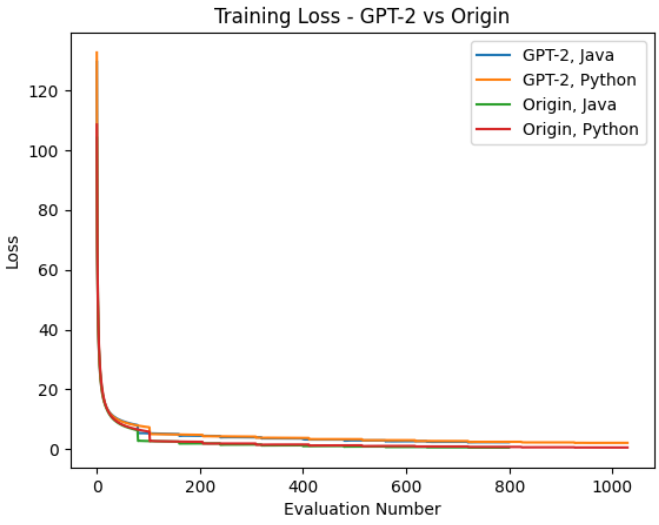

Fig. 4. Training Loss - GPT-2 vs Origin

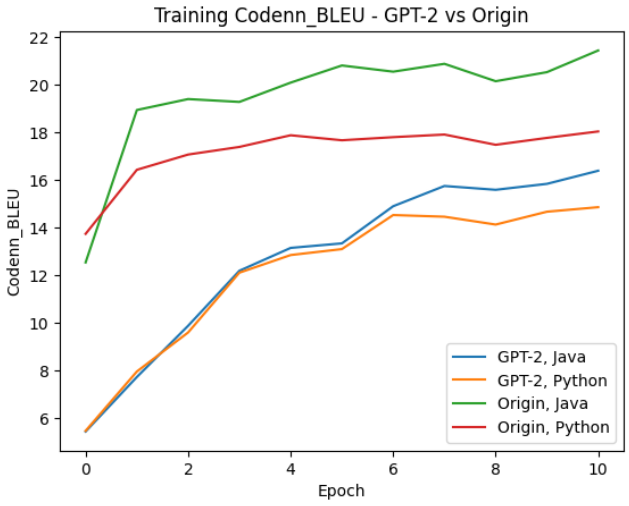

Fig. 5. Training Codenn_BLEU - GPT-2 vs Origin

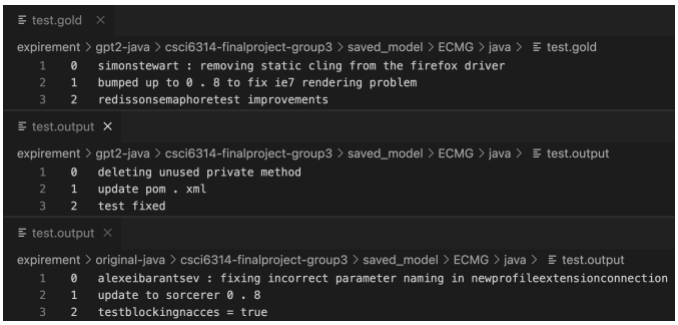

Fig. 6. Generated Commit Messges: Gold vs GPT-2 vs Original

reach converge around epoch 6, but we still see the trends of increases in the last few epochs. So, we expect higher and more accurate BLEU if we increase the epochs to 15. We can also find that both models performed well on Java, although the difference is smaller in the C+G model.

BLUE scores of the whole dataset using only 20% data. While it complies with the Pareto principle, we suspect if the RACE dataset has enough different data or if we need to adjust the attention mechanism in the RACE to fully utilize the rest 80% of the data. Another interesting finding is we achieved above 80% . By comparing the generated messages in Fig. 6, we found the messages generated by C+G are much simpler than the original and gold versions. We think it is because of the corpus used to train GPT-2. It can understand well the vocabulary that is used in human life but has some problems with the programming languages. But in the meantime, it improves the readability of the commit message. We found some generated commit messages have much better meaning even than the gold commit message. This can avoid the model from generating false responses (non-understandable commit messages) because of the bad training data.

B. *AST*

The hyperparameter and test environment are the same as the model test. The difference is we use the original RACE model and add an encoder of UniXcoder into the ECMG model

as a New RACE model. We called it C+U_en, referred to as CodeT5 + Encoder of UniXcoder. The hardware setup for the training model uses one Nvidia A10. The training time for Java language is shorter than Python language 3 hours. The training time for the Python language is around 25 hours (also including both the retrieval and generation phases). The generation part took the most time, around 21 hours. Compared with training time in the model part, we found the time has a large increase. The reason for that is due to the quality of the server that we rent. It performed not like the Nvidia A10 support provided for us.

Comparing the C+U_en model with the RACE model in BLEU score, Fig.7, we found new RACE model, after adding the encoder of UniXcoder, is slightly better than the original model. According to the training loss comparison in Fig.8, we can find the loss of the RACE model is higher than the C+U_en model. While the evaluations of the model reach a convergence is around 100 times. When we back to Fig.7 to analyze the trends of increase

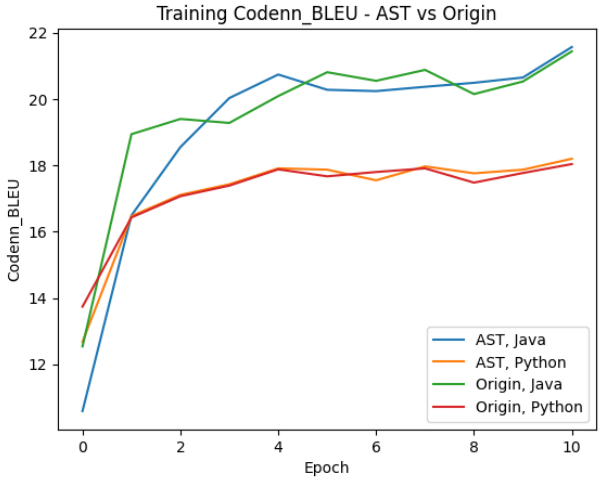

Fig. 7. Training Codenn_BLEU – Added AST Encoder vs Origin

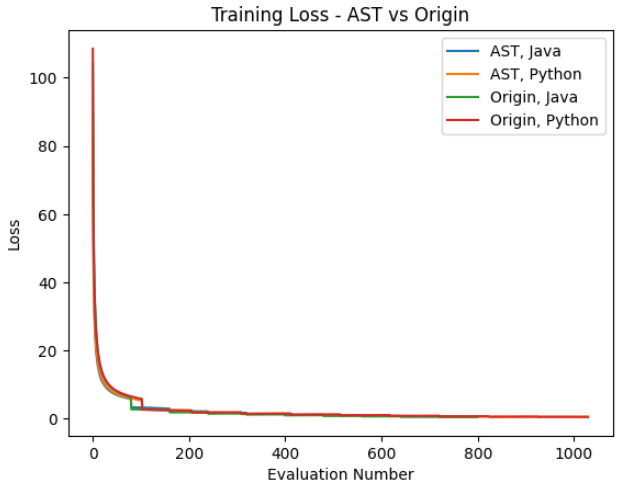

Fig. 8. Training Loss – Added AST Encoder vs Origin

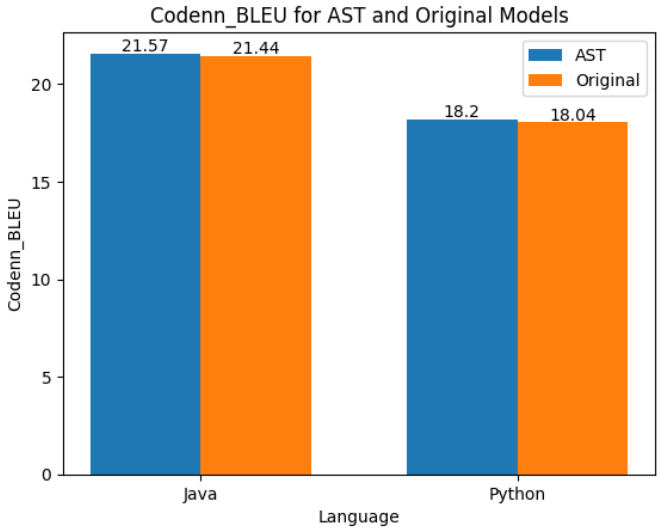

Fig. 9. Coden_BLEU for Added AST Encoder model and Original mode

based on current epoch numbers, we can find the trend become increasing more between epoch 8 to epoch 10. Based on this, we expect the BLEU score can become better when we increase the epoch number. The final Codenn_BLEU score in Fig.9, clearly represents added AST model is a valuable option.

C. *Dataset*

The Go dataset we created is extracted from 134 repos and contains 26920 commits in total. After splitting the dataset into training, testing, and validating sets, each set has 17342, 2167, and 2169 commits respectively. In order to have more reliable results, we collected 17342 commits from the Java dataset so that we have two balanced datasets. Then, the original RACE model is trained on the two datasets and the results are presented and analyzed below.

The model trained on the Go dataset outperformed the model trained on the Java dataset. The BLEU score that the Go model obtained on the validation set during the training is about 40% higher than the BLUE score on the Java model as shown in Figure 2. In Figure 10, the first 50 training losses are removed as the first 50 losses are generally larger. The larger loss will make the graph scale up and it hard to identify the small loss change later. As shown in this graph, the Go model's training loss is also significantly lower than the Java model's training loss. The BLEU scores evaluated on the testset are 27.16 for the model trained on Go, and 18.84 for the model trained on Java. This final BLEU score for Go is at the same level and even surpassed many of the original RACE's BLUE score train on the full dataset for other languages.

The potential cause for the huge performance difference might be the difference in preprocessing of the dataset. It was discovered that the preprocessed dataset RACE used does not account for a special case. In that special case, the dataset does not properly remove the commit index, therefore generating

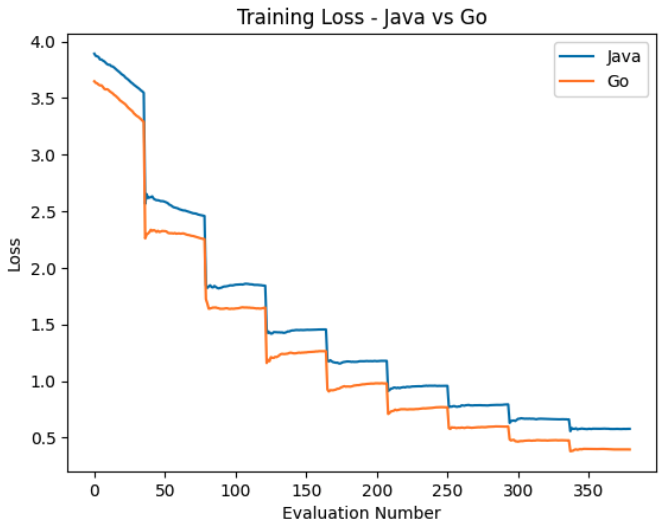

Fig. 10. The training loss comparison between Java and Go

noises in the dataset and increases the vocabulary. Our preprocessing script properly deals with that corner case and improved the quality of the Go dataset. In addition, Java's training data file is four times larger than Go's training data file with the same amount of commits. That means the MCMD's preprocessing script does not remove long commit changes which could reduce the BLEU score. Our preprocessing script removes the commit with more than 512 code diff tokens. It makes our Go dataset focus on small changes and therefore increases the BLEU score.

Overall, our Go dataset is well-processed. It improves the RACE model's performance and could generate more reasonable and reliable commit messages.

IV. RELATED WORK

Before we focus on the RACE model, we pick lots of journals that talk about this topic. In the following, we will mention a few models which also interested us and also related to the commit message generation. ATOM [7] is one of the hybrid models. In the journal, it proposes a new approach for generating commit messages by reading software code differences. The reason to start this project is the author feels writing commit messages is painful as a software engineer. Therefore, they want to develop an automated commit message generation tool. Meanwhile, the author told they will use a combination of AST and hybrid ranking to build the Generation module.

AST model approach begins by constructing an AST Encoder for the code differences, it captures the data structure and content of the code in a concise way. Using hybrid ranking to determine the most relevant message that stored in the commit message datasets. It will combine content-based and context-based features. The content-based feature takes care of the code differences and semantic meaning of the commit message. The context-based feature takes care of the original commit message and related messages.

They evaluated their approach based on a dataset that built themselves. They use GitHub to find high-quality projects and download all the content of those repos. Using these commit messages as their datasets. In terms of the accuracy and diversity of the message generated, the authors' approach outperformed the baseline.

Overall, the ATOM model is a highly precise and concise hybrid model to automate generating of the commit message in software development. It inspired us to implement AST in the RACE.

V. CHALLENGES

After reviewing this project, the common challenge for all the members is computing resources. Before the school provide the server to us, we tried to use a personal computer and google collab to run the model. The issues are our computer didn't have NVIDIA graphic card and Google Colab has limitations in user duration. Based on that, we had to postpone moving forward with the project until we got the server provided by the university.

A. GPT-2

The most challenging part of incorporating GPT-2 into the existing model is modifying the complex, flawed pipeline. Because RACE is a comparatively new repo, there are no public contributions maintaining it. There are many bugs and environmental issues in the existing pipeline. We took a long time to debug and fix the whole pipeline. Besides, training the model is also a challenging task. On average, it took 10 hours to train a model for one language using 20% data using NVIDIA GTX 3070Ti.

B. AST

From the ATOM model, we noticed that we can add one type of AST model into the RACE model to help it increase the precision of generating commit messages. The first problem is how to pick, which type of AST will better suit for RACE model. As the class begin, we touch on the code embedding topic with Mr. Xu Bowen. His research topic is similar to our project. And he mentioned a pre-trained model that he used before, which is UniXcoder.

C. Dataset

The challenge faced when preparing the dataset was mainly the lack of information and description. The RACE dataset only briefly mentioned how the MCMD dataset is created. MCMD paper [6] also does not focus on explaining preprocessing of their dataset. They mainly discuss the implication of their dataset on many existing models without a detailed explanation for how they obtained their dataset. We had to trace through many papers mentioned to have a better understanding of how the preprocessing was done and tried out many scripts and a machine learning model to find out the preprocessing script used for generating the MCMD dataset. We eventually made our own preprocessing script by collecting the information from different papers that are interlinked.

One of the limitations of our Go dataset is that it only extracts the commit information from 134 repos. These 134 reports were originally used for testing whether the reprocessing script could properly function. However, when we finally know how many

repos we need to create a Go dataset that is comparable to the 20% of the MCMD dataset, the GitHub Search Tool was down due to a download error. We were not able to get a bigger Go dataset, so we used the smaller dataset for training and evaluation. For future work, a bigger Go dataset can be obtained to test the RACE model's performance.

VI. CONCLUSION

Based on the result of the C+G model, we can say GPT-2 model is not a good choice to support the RACE model getting high accuracy and diversity of the generated commit message, but it increases the readability of the generated commit messages. Fig.2 and Fig.3 represent the BLEU score and training loss for the C+G model is lower than the original RACE model. Fig. 6 shows the commits messages generated by C+G are more reliable. The result of the C+U_en model gives us a better performance. The BLEU score is slightly higher than the original RACE model. In the dataset, we verified our new dataset for Go language outperformed the original dataset for Java language.

REFERENCES

[1] E. Shi, Y. Wang, W. Tao, L. Du, H. Zhang, S. Han, D. Zhang, and H. Sun, "Race: Retrieval-augmented commit message generation," *arXiv.org*, 22-Oct-2022. [Online]. Available: https://arxiv.org/abs/2203.02700. [Accessed: 02-Apr-2023].

[2] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified Cross-modal pre-training for code representation," *arXiv.org*, 08-Mar-2022. [Online]. Available: https://arxiv.org/abs/2203.03850. [Accessed: 02-Apr-2023].

[3] Salesforce, "Salesforce/Codet5: Code for Codet5: A new code-aware pre-trained encoder-decoder model.," *GitHub*. [Online]. Available: https://github.com/salesforce/CodeT5. [Accessed: 08-Apr-2023].

[4] "GPT2 · hugging face," *gpt2 · Hugging Face*. [Online]. Available: https://huggingface.co/gpt2. [Accessed: 08-Apr-2023].

[5] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "UniXcoder: Unified Cross-Modal Pre-training for Code Representation." Accessed: Apr. 09, 2023. [Online]. Available: https://arxiv.org/pdf/2203.03850.pdf

[6] W. Tao et al., "On the Evaluation of Commit Message Generation Models: An Experimental Study," *in proceedings of 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Luxembourg, 2021, pp. 126-136, doi: 10.1109/ICSME52107.2021.00018.

[7] S. Liu, C. Gao, S. Chen, L. Y. Nie, and Y. Liu, "ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking," *IEEE Transactions on Software Engineering*, vol. 48, no. 5, pp. 1800–1817, May 2022, doi: https://doi.org/10.1109/tse.2020.3038681.

[8] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing and X. Wang, "Neural-Machine-Translation-Based Commit Message Generation: How Far Are We?," *in proceedings of 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Montpellier, France, 2018, pp. 373-384, doi: 10.1145/3238147.3238190.

[9] S. Jiang and C. McMillan, "Towards Automatic Generation of Short Summaries of Commits," *in proceedings of 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, Buenos Aires, Argentina, 2017, pp. 320-323, doi: 10.1109/ICPC.2017.12.