Model Deployment Group 2: Deep Comment Generation

Motivation:

Code comments are essential to software development as ~59% of developers' time is spent writing comments for program comprehension to maintain software functionality for future iterations. As such, developers infer functionality from the source code which can be time-consuming and inefficient. This is compounded by tight project schedules leading to mismatched, missing, or outdated comments. In general, good comments help developers understand a part of code (such as functions or methods) through natural language descriptions of the comments. Automatic generation of code comments can save developers' time in writing comments, efficiently assist in the comprehension of source code, and reduce additional time spent on reading and navigating source code.

Therefore, our work will seek to integrate and enhance the current work within the realm of machine translation conducted in the field of Natural Language Processing (NLP). Our group will utilize language models learning from a large-scale source code corpus and generate comments from learned features within an implementable, enhanced model. To accomplish this task, we focused on implementing a Transformer-based neural architecture to train the model. In phase 2, our group attempted the following methods:

1. Replace CodeBERT with other distilled models that are RoBERTa-based.
2. Distill CodeBERT ourselves in a manner similar to how BERT was distilled.

Our goal is to deploy an off-the-shelf model called Microsoft CodeBERT on a cloud instance using AWS and augment the model through distillation for increased efficiency. An example of such a model is DistilBERT - a BERT model which is ~50% smaller, but still retains its learning capabilities whilst utilizing less memory [3]. We hope the increased throughput and efficiency of language understanding capabilities can be compared with the original CodeBERT model to quantify memory usage, throughput, training time, and latency when conducting metrics comparison. Furthermore, if feasible and time permits, we want to qualify the discrepancy between increased layers between the original and improved model. Through these models, we want to acquire the most "efficient and practical" model for comment generation such that the model averages the highest score, lowest training time, and training error over a reasonable number of epochs. Through this, we also wish to obtain a subjective nature of which model is the most GUI friendly and produces results most applicable for software development.

Methodology:

We first researched various model architectures, specifically DistilBERT, RoBERTa, and CodeBERT. DistilBERT is smaller, faster, cheaper, and lighter than BERT, utilizing fewer parameters than the original BERT model through pre-training distillation. Therefore, after pre-training, we wish to fine-tune the parameters to determine good performance for automatic comment generation. CodeBERT is an extension of the BERT model but is a bimodal pre-trained model for natural programming language and one of its downstream tasks is converting code into natural language through code documentation generation [1]. RoBERTa is a robustly optimized model for BERT where the following hyperparameters were modified: i) longer training time, with larger batches, over more data; ii) removing next sentence prediction objective; iii) training longer sequences; iv) dynamically changing training data masking pattern [6].

After running the base model for our task, we modified the existing pipeline to finetune our DistilBERT and CodeBERT models. As aforementioned, DistilBERT's pre-trained model can reduce the size of a BERT model by ~40% while retaining 97% of its language understanding capabilities and being 60% faster [3] - this is explained more thoroughly through the visual description in Figure 1 below. For CodeBERT evaluation, we constructed and executed a pipeline that generates comments for the CodeXGLUE dataset for the Python programming language. We then fine-tuned this model for our target downstream task. Figure 2 explains how CodeBERT downstream tasks enable automatic code comment documentation.

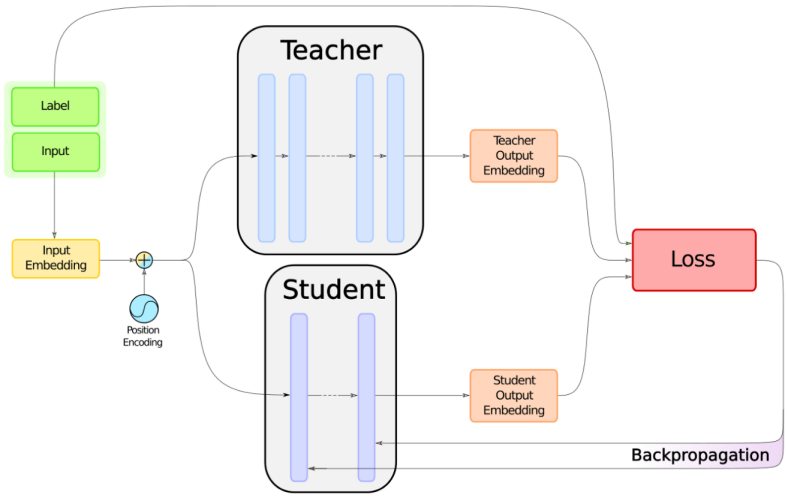


Figure 1: Training Process for DistilBERT

Our design choices for the three models are primarily based on the differences described above: each model has its own caveat and should perform differently when run under the same number of epochs. DistilBERT is modified during the pre-training phase whilst CodeBERT is based on RoBERTa thereby the hyper-parameters for this model are fine-tuned (post-training). Finally, a

comparative analysis of the results through various metrics such as training time, parameter size, loss, accuracy, etc. can help distinguish the performance of DistillBERT and CodeBERT models.

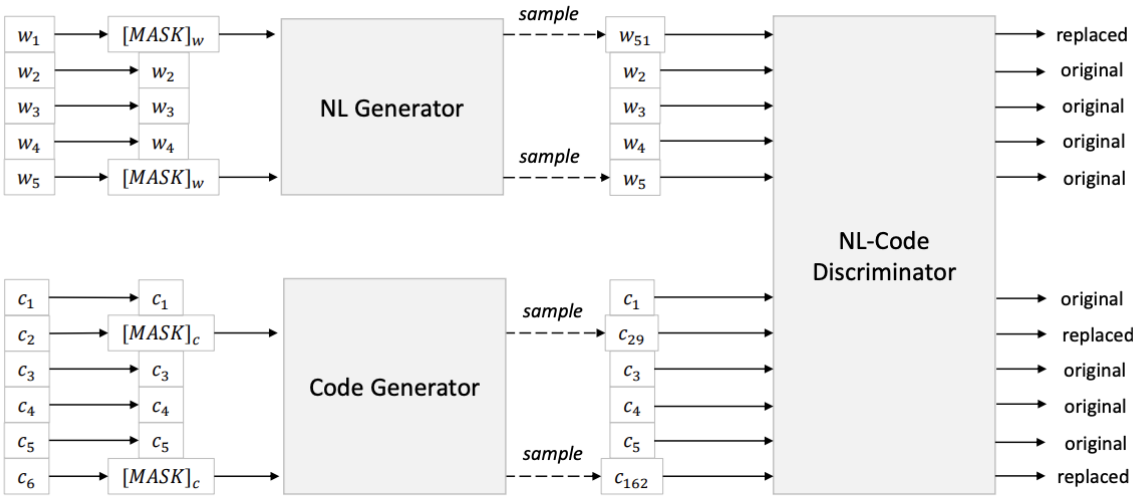


Figure 2: An illustration about the replaced token detection objective. Both NL and code generators are language models, which generate plausible tokens for masked positions based on surrounding contexts. NL-Code discriminator is the targeted pre-trained model, which is trained via detecting plausible alternatives tokens sampled from NL and PL generators. NL-Code discriminator is used for producing general-purpose representations in the fine-tuning step. Both NL and code generators are thrown out in the fine-tuning step.

	BERT	DistilBERT
Size (millions)	**Base**: 110 **Large**: 340	**Base:** 66
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 days*)	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT.
Performance	Outperforms state-of-the-art in Oct 2018	3% degradation from BERT
Data	16 GB BERT data (Books Corpus + Wikipedia). 3.3 Billion words.	16 GB BERT data. 3.3 Billion words.
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT Distillation

Figure 3: Analytical Comparison of DistilBERT vs BERT Model

Alongside the above metrics, the datasets we will utilize between Phase 1-2 and Phase 3 are different. Specifically, Phase 1 and 2 utilized CodeSearchNet whereas Phase 3 utilized CodeXGLUE. The discrepancy between them is visualized in Figure 4.

	Number of Functions	
	w/ documentation	All
Go	347 789	726 768
Java	542 991	1 569 889
JavaScript	157 988	1 857 835
PHP	717 313	977 821
Python	503 502	1 156 085
Ruby	57 393	164 048
All	2 326 976	6 452 446

Table 1: Dataset Size Statistics

Table 3: Data statistics about the filtered CodeSearchNet dataset for the code summarization task.

Language	Training	Dev	Testing
Go	167,288	7,325	8,122
Java	164,923	5,183	10,955
JavaScript	58,025	3,885	3,291
PHP	241,241	12,982	14,014
Python	251,820	13,914	14,918
Ruby	24,927	1,400	1,261

Figure 4: CodeSearchNet (Left) Dataset vs CodeXGLUE (Right) Dataset

Some focal features of this dataset which will further be examined in discussion and conclusion is that data preprocessing was already conducted by filtering out the following syntax to decrease overall parameters:

i) Code which could not be parsed into an abstract syntax tree
ii) Document tokens shorter than 3 or larger than 256.
iii) Document containing special tokens such as “http://"
iv) Empty document or not written in English

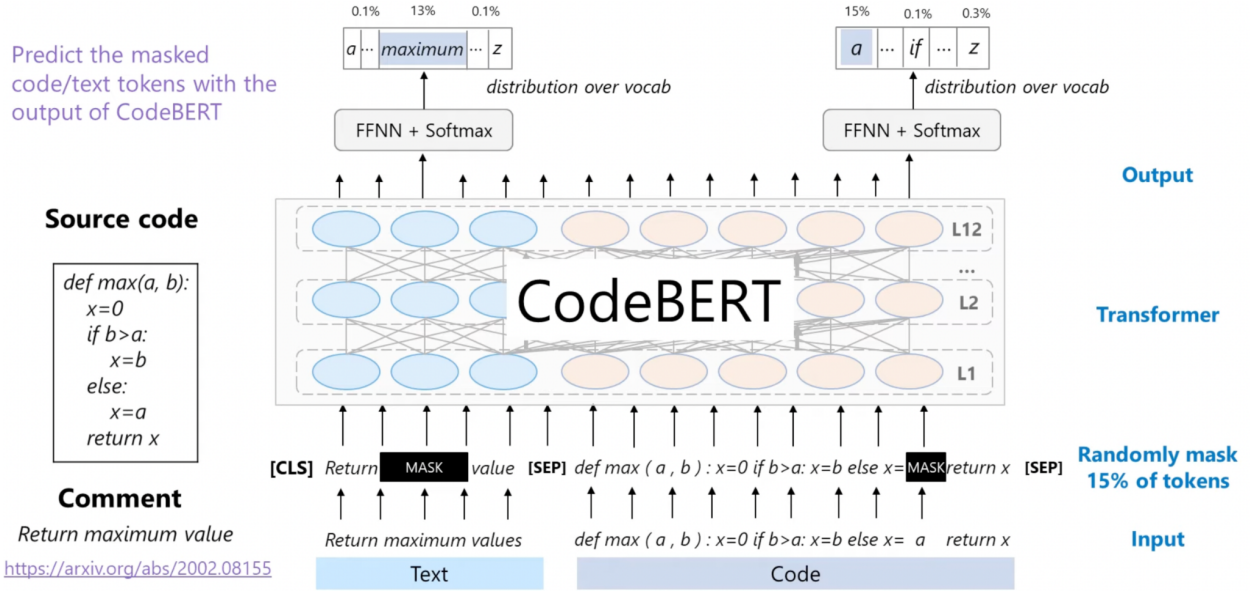


Figure 5: CodeBERT Model Architecture

Thereafter, we attempted to distill CodeBERT and compare the results of distilled-CodeBERT with CodeBERT; however, we ran into debugging issues and were not able to fully distill our CodeBERT model. Surprisingly, we found a curated model which incorporates a distilled version of CodeBERT with RoBERTa integrated as its base architecture called CodeBERTa. Therefore, our hope is that the fine-tuned hyper-parameters cultivated within the RoBERTa model can be

utilized in conjunction with distillation to decrease training time but preserve increased throughput and efficiency of language understanding capabilities. As such, we hope to compare 4 pre-trained models to each other with the hopes of increasing efficiency and performance: the original CodeBERT (codeBERT), a smaller CodeBERT (CodeBERTa-small), the base RoBERTa model (roberta) and a distilled RoBERTa model (DistilRoBERTa). We will use BLEU scores to quantify accuracy as it is a metric to evaluate machine-translated text; the closer the predicted sentence is to a human-generated target sentence, the better the BLEU score.

Results:
Plots and Figures:
Phase 1 & 2 results have been combined for brevity. All models were run on a AWS p2.xlarge instance with a NVIDIA K80 GPU. Furthermore, the model specifications, evaluation parameters and a summary table for the obtained raw data alongside analytics for training time averages is depicted below:

Model Specs:
CodeBERT: 125 M, 12 layers
CodeBERTa-small: 84 M, 6 layers

Evaluation Parameters:
Learning rate: 5e-5
Batch size: 16
Beam_size: 10
Source length: 256
Target length: 128
Epochs: 10

Dataset Split (samples)	**Validation**	**Testing**
	4020	**421**

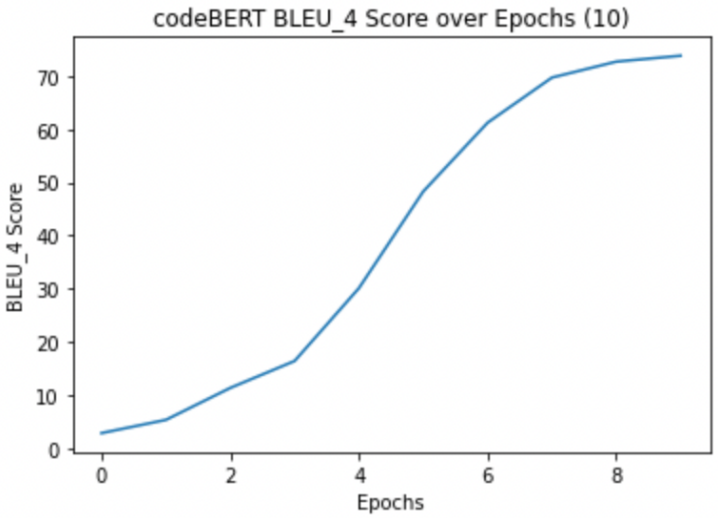
BLEU_4 Scores	**Validation**	**Testing**
codeBERT	**73.85**	**79.41**
codeBERTa	**72.9**	**74.4**

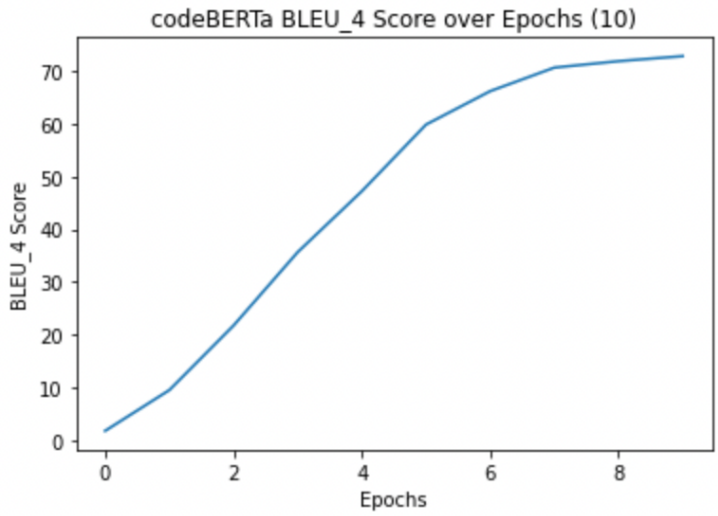
Final Validation Loss	
codeBERT	**0.7623**

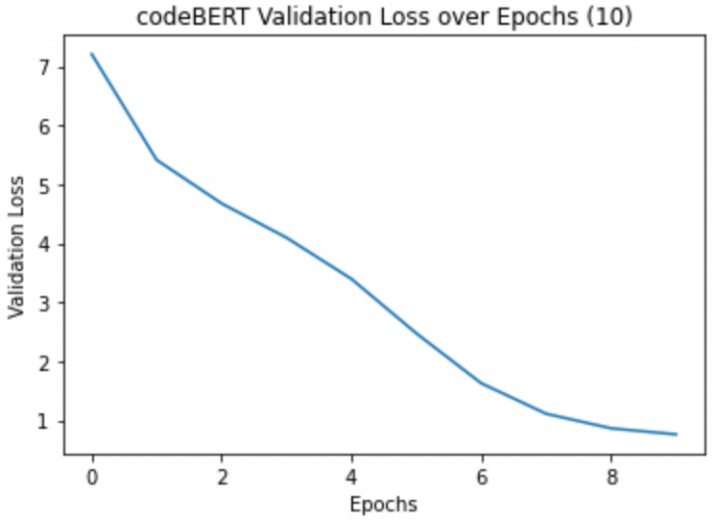
codeBERTa	**1.186**

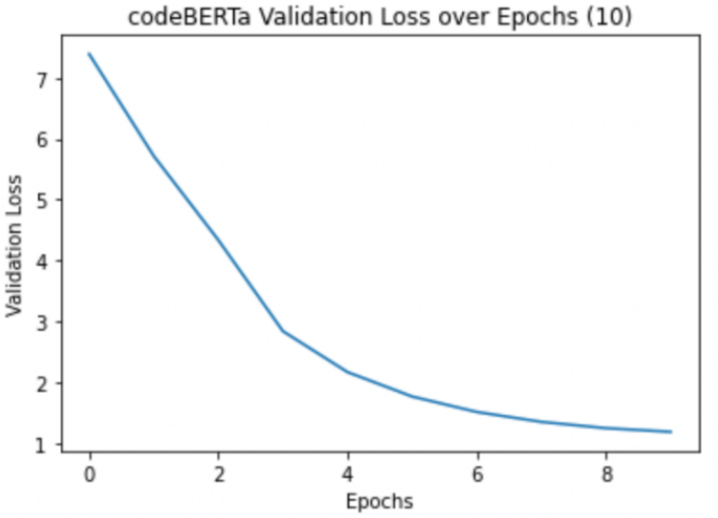
Average Evaluation Time (min:sec)	
codeBERT	**8:18**
codeBERTa	**5:55**

Top Left Plot: CodeBERT Validation BLEU Score over 10 Epochs,
Top Right Plot: CodeBERTa Validation BLEU Score over 10 Epochs,
Bottom Left Plot: CodeBERT Validation Loss over 10 Epochs,
Bottom Right Plot: CodeBERTa Validation Loss over 10 Epochs.









For Phase 3, we attempted to model the parameters under RoBERTa, DistilRoBERTa and compare against CodeBERTa-small. The goal was to deploy pre-trained, distilled versions of CodeBERT and RoBERTa and fine tune the model performance using the CodeXGLUE datasets. The data and results are presented in a similar fashion as depicted for Phase 1 and 2.

Model Specs:

CodeBERT:	125 M, 12 layers
RoBERTa:	125 M, 12 layers
DistilRoBERTa:	82 M, 6 layers
CodeBERTa-small:	84 M, 6 layers

Evaluation Parameters:

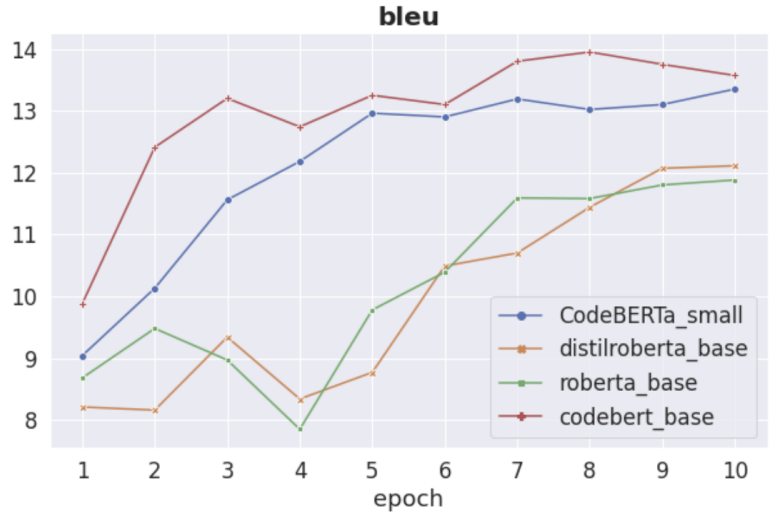
Learning rate:	5e-5
Batch size:	32
Beam_size:	10
Source length:	256
Target length:	128
Epochs:	10

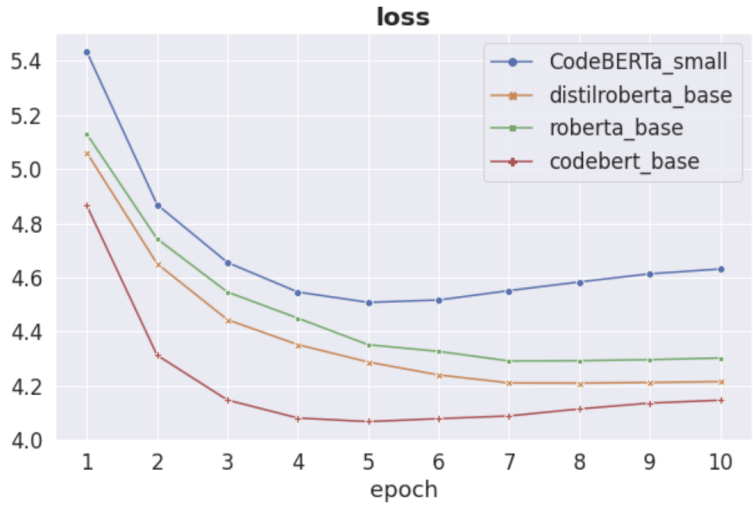
Plots and Figures:

All models were run on a AWS p2.xlarge instance with a NVIDIA K80 GPU. The results for Phase 3 have been visualized below in the following order, validated upon the CodeXGLUE dataset:

Left: Validation BLEU Scores of the 4 models over 10 Epochs

Right: Validation Loss of the 4 models over 10 Epochs





BLEU_4 Scores	Validation	Testing
codeBERT	**13.57**	**13.23**
codeBERTa	**13.35**	**12.91**
RoBERTa	**11.88**	**11.81**
DistilRoBERTa	**12.11**	**11.92**

Loss	Evaluation	Testing
codeBERT	**4.14**	**9.01**
codeBERTa	**4.63**	**9.18**
RoBERTa	**4.30**	**10.26**
DistilRoBERTa	**4.21**	**9.69**

The accuracy metric for our models will be assessed through the use of BLEU Scores. BLEU, or the Bilingual Evaluation Understudy, is a score for comparing a candidate's translation of the text to one or more reference translations. It's one of the most popular methods where the closer the predicted sentence is to the human-generated target sentence, the better the BLEU Scores are. Traditionally, BLEU Scores are between 0 and 1, but we acquired normalized scores between 0-100 through multiplying the raw scores to acquire their percentage equivalent scores. The value of the score indicates how similar the candidate text is to the reference texts, with values closer to 100 representing more similar texts. Few human translations would attain a score of 100 as this means that the candidate text is identical to one of the reference translations. Another way to think of this would be that even two humans would likely come up with different sentence variants for a problem, and would rarely achieve a perfect match. *For this reason, it is not required that our score is closer to 100 and is unrealistic in practice because it raises a flag that our model is overfitting.*

Discussion:

Our discussion will revolve around the results and implications associated with the specific values we acquired. Furthermore, we will discuss the intricacies of the methods utilized, the reasoning and logic for each model and the implications of our methodology. Initially, the CodeBERT model was to be distilled by ourselves for Phase II. Our experiment would incorporate hyperparameter tunings as discussed in class to associate each hyperparameter modulation with its

corresponding effect on the model and the results. Through this, we would be able to fully distinguish the model performance with respect to the modulation of hyperparameters. Through accomplishing this, we wanted to observe the discrepancy between two to three hyperparameter tunings to a fully optimized model with hyperparameters pre-ordained with set values and constraints (as was accomplished with our pre-trained BERT models). In essence, would scores have been lower, would training have taken more time, etc. In distilling our own model, we hoped to determine why RoBERT was robust and optimized. Some of the questions we hoped to resolve were the following:

- Why is it that the RoBERT model is robust and optimized?
- If we were to change one hyperparameter, what would its effect be?
- Can we determine exactly what the cause of one hyperparameter change would be?

With regards to the BLEU Scores, there are multiple implications to consider for the scores which we have received. In the current state of the field, DistillBERT is a knowledge based distillation of the CodeBERT model. To clarify, language used in the code is usually syntax-specific and logic-specific while natural language is more suitable for describing ambiguous concepts and ideas. Therefore, using knowledge distillation of multiple Teacher Models to train one Student Model corresponds to MPLCS (Multi-Programming Language Code Search), which has the ability to incorporate multiple languages into one model with low corpus requirements, as visualized above with the evaluation parameters. However, there are multiple limitations associated with the whole idea of code comment generation. The most glaring one is the crux of code comment generation which resulted in the low scores we observed in our results: code comments are poorly written and most developers do not know how to write good code comments. This in and of itself means that the training data on any corpus, whether a largely compiled list or python documentation itself, will contain poorly described comments. For the analysis below, our group consulted the following paper which details the shortcomings, implications and future considerations for our project.

In all the datasets the paper analyzed, BLEU was used to evaluate the quality of the comment generation. However, subtle differences in the way the BLEUs were calculated makes the results rather difficult to compare. BLEU (as do related measures) indicates the closeness of a candidate translation output to a “golden" reference result. BLEU per se measures the precision (as opposed to recall) of a candidate, relative to the reference, using constituent n-grams. BLEU typically uses unigrams through 4-grams to measure the precision of the system output. If we define :

$$p_n = \frac{number\ of\ n-grams\ in\ both\ reference\ and\ candidate}{number\ of\ n-grams\ in\ the\ candidate}$$

BLEU combines the precision of each n-gram using the geometric mean:

$$e^{\frac{1}{N}\times\sum_{n=1}^{N} log(p_n)}$$

Therefore, single word outputs or outputs that repeat common n-grams could potentially have high precision. Thus, a "brevity penalty" is used to scale the final score; furthermore each n-gram in the reference can be used in the calculation just once. From just a deep dive with the BLEU scores, we can see some obvious problems which will arise. The emphasis on the geometric mean with regards to the output generation can seem to indicate that precision is of the most importance. We believed to have rectified this by using a distilled corpus from CodeXGLUE, but there are multiple other factors to consider.

Corpus vs. Sentence BLEU: BLEU Scores are typically measured across all the samples in the held-out test set. Therefore, another factor is deciding the combination of the results between test set scores. The methods are either C-BLEU (corpus based) or S-BLEU (sentence based). The p_n in C-BLEU is calculated such that one example will always have a 4-gram overlap - therefore, the geometric mean will not be 0 as it calculates p_n across every example in the test corpus. S-BLEU score for the test set is calculated by simply taking the arithmetic mean the BLEU score calculated on each sentence within the set.

Tokenization Choices: A final consideration comes not from the metric calculation, but from the inputs itself. Because the precision counts are at a token level, it has been noted that BLEU is *highly sensitive* to tokenization. This means that when comparing prior work on a dataset, *one must be careful not only to use the same BLEU calculation*, but *also the same tokenization and filtering*. When calculating scores on the datasets, we use the *tokenization* provided with the dataset.

Tokenization can be very significant for the resulting score. As a toy example, suppose a reference contained the string "calls function foo()" and an hypothesis contained the string "uses function foo()". If one chooses to tokenize by spaces, one has tokens [calls, function, foo()] and [uses, function, foo()]. This tokenization yields only one bigram overlap and no trigram or 4-gram overlaps. However, if one instead chooses to tokenize this as [calls, function, foo, (,)] and [uses, function, foo, (,)] we suddenly have three overlapping bigrams, two overlapping trigrams, and one overlapping 4-gram. This results in a swing of more than 15 BLEU-M2 points or nearly 40 BLEU-DC points (different variations of BLEU scores).

These are crucial considerations which our group did not factor into when we calculated BLEU scores - we utilized a pre-trained model with the hopes of seeing some good results, but the reality is that BLEU scores in and of themselves can have multiple factors when considering how to reconcile the n-grams and geometric means associated with each calculation, and especially the issue of tokenization. From just tokenization, we can discern how poorly written comments can affect the arithmetic mean calculated. Furthermore, our models utilized the standard BLEU-4 score which does not consider any of the above factors or variations. Additionally, it was stated within the paper that even naive IR methods provide competitive performance in many CCT datasets,

they can be an important part for checking for issues in the new collection and new processing of CCT datasets.

This results in the following issues which the paper also noted was critical to the low BLEU Scores observed:

1. Code comments are far more repetitive than the English found in Natural Language Translation datasets
2. Frequent n-grams could wield a much stronger effect on the measured BLEU performance on code-comment translation tasks than on natural language translation
3. The natural language translation (WMT) shows a stronger input-output dependence than the CCT datasets in that similar inputs are more likely to produce similar outputs.

Therefore, BLEU Scores obtained from code comment generation will always incur some sort of penalty due to its repetitiveness alongside the instance that similar inputs are more likely to produce similar outputs - this was surprisingly observed in our phase 2 results where generated function comments were repeating the same words over and over again.

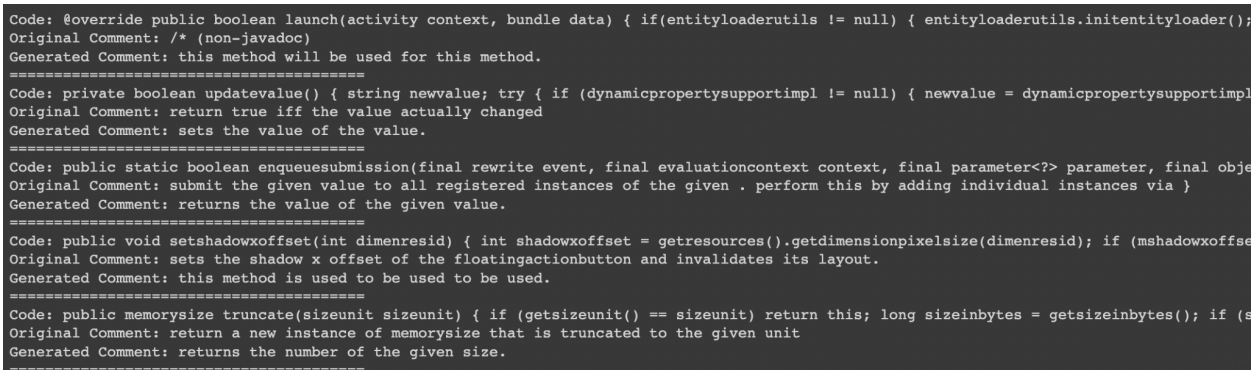


Figure 6: CodeBERTa Comment Generation (Epochs = 5; Distilled Model)

Clearly, “this method will be used for this method” is extremely repetitive and our assumption was to change the training corpus of the pre-trained CodeBERTa model from CodeSearchNet to CodeXGLUE (a distilled training corpus) to try and rectify the situation. However, this is a natural consequence of functions within the deep learning model exhibiting a smooth input-output dependency. An interesting caveat we could attempt was to try different variants of BLEU-Score. The paper found that variations such as BLEU-Moses or BLEU-ncs performed the best with scores ranging from ~30-40, but this was only due to the specific datasets they were using. It is even more crucial to note that BLEU scores cannot be compared from training set to training set - this is based on the prior considerations mentioned. This begs the question: would a variation in BLEU alleviated and rectified our scores? Comparison between even a distilled dataset and a normal dataset is not comparable. As such, the standard BLEU score may have been the most apt decision even after further consideration, but more scores in general may have allowed for more perceptive analyses for our models itself.

To summarize the paper's findings with our results, we consider the following issues which may have led to lower BLEU scores than expected.

Comment Repetitiveness

The paper found that comments in Code Comment Translation datasets are far more repetitive than the English found in the WMT dataset. Furthermore, this is a function of how words are combined in comments rather than greater vocabulary in distribution of comments. The highly-prevalent patterns in comment have a substantially greater impact on the measured BLEU performance of models with their datasets containing trigrams such as "creates a new, returns true if, constructor delegates to, factory method for" frequently. Getting these right (or wrong) has a huge influence on performance.

Implications:

These findings suggest that getting just a few common patterns of comments right might deceptively affect measured performance. The actual performance of comment generation might deviate more so than natural language translation. Comment repetition indicates that fill-in-the-blanks with more data-driven approaches are utilized; classify code first, to find the right template and then fill-in-the-blanks, perhaps using an attention or copy-mechanism.

Input/Output Dependence

When translating from one language to another, one would expect more similar inputs to produce more similar outputs, and that this dependence is relatively smooth and monotonic. This property is indeed true for general natural language outputs, but not as much for the comments.

Implications:

Functions exhibiting a smooth input-output dependency could be reasonably expected to be easier to model. BLEU is a measure of lexical (token sequence) similarity; the paper surmised that token-sequence models that work well for Natural language translation may be less accurate for code. As such, other models of code, such as tree or graph-based, could be further explored.

Baselining with IR

The paper emphasizes with their experience that a simple IR approach provides BLEU performance that is comparable to current state of the art.

Implications:

Our findings suggest that a simple, standard, basic IR approach would be a useful baseline for approaches to the CCT task. Especially considering the range of different BLEU and tokenization approaches, this would be a useful strawman baseline.

Interpreting BLEU Scores

BLEU, METEOR, ROGUE, etc. are measures that have been developed for different tasks in natural language processing, such as translation & summarization. BLEU is most commonly used in code-comment translation; therefore, we also decided to use BLEU score as our metric for accuracy. However, the results of the paper reported that BLEU scores are not that high.

Implications:
The best reported BLEU scores for the German-English translation tasks are currently in the low 40's. On some datasets, the paper suggests that the performance of models are comparable on average to retrieving the comment of a random method from the same class. This cannot be confirmed, but can provide insight into minimum expected numbers for a new CCT dataset.

Learning From NLP Datasets
Bottomline: the current landscape of CCT datasets is rather messy. There are often several different versions of the same dataset with different preprocessing, splits, and evaluation functions which all seem equivalent in name, but might not be comparable if not pre-processed properly. Due to the number of open source repositories, there are also a large repeated number of code and comment pairs. However, some tasks in NLP do not seem to observe such variance within a task. No human is required to label properties of text, therefore less effort might be taken on quality control. Since researchers are domain experts in the datasets, they might be also more willing to apply their own version of preprocessing. In addition, there are a wider array of tools to enforce consistency on various NLP tasks.

All in all, from the prior discussion, it is important to note for future considerations what ought to be done to assess accuracy of our models. Firstly, future considerations should involve the use of Information Retrieval methods to be utilized as a baseline performance metric for downstream comparative analysis against the scores obtained, whether it be BLEU, ROGUE, etc. Furthermore, it was not prudent to rely solely on BLEU scores for accuracy metrics and further research provided us more insight into how to vary the considerations of the BLEU metric such that it can be "tweaked" for the purposes of Code Comment Translation. In addition to using the BLEU score, contextualizing the multiple factors and considerations such as smoothing, S-BLEU vs. C-BLEU and tokenization would help assuage and bring better performance from the model itself. Repetitiveness of comments, input/output dependence and tokenization choice is all crucial in the overall performance of BLEU scores acquired from our models. Future considerations can hope to incorporate smoothing variations and/or S-BLEU variant scoring metric and compare against baseline BLEU scores.

Overall, the field is immense - the amount of work being done in NLP and in the tiny sector of comment generation is diverse and ever expanding. The applications and theories from CCT have scope to expand into different methods of continuous functional data approximation methods such as biology, chemistry and computer science itself. Our group learned a lot about the realm of code comment generation and realized the intricacies of model deployment over the past few weeks - we are excited with the results and hope future work done by us will rectify and resolve the shortcomings encountered with this model deployment.
Github Link:

https://github.com/kuya-kevin/Distilling-CodeBERT

Teamwork:

Wenting Shi:

Created a pipeline to run Phase 3 models

Rong Jin:

Deployed Phase 3 models on an AWS instance

Kevin Elaba:

Created a pipeline to run Phase 3 models, allocated AWS resources to teammates

Revised Phase 3 slides, revised Phase 3 report

Parnal Sinha:

Slides for Phase 3

Report for Phase 3 (excluding plot generation in results)

References:

1. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020). CodeBERT: A Pre-Trained Model for Programming and Natural Languages. *ArXiv:2002.08155 [Cs]*. http://arxiv.org/abs/2002.08155
2. Xing Hu1 , Ge Li1 , Xin Xia2 , David Lo3 , Zhi Jin1 . 2018. Deep Code Comment Generation. In Proceedings of IEEE/ACM International Conference on Program Comprehension, Gothenburg, Sweden, May 27 - May 28, 2018 (ICPC'18). ACM, New York, NY, USA, 11 pages. https://doi.org/10.475/123_4
3. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *ArXiv:1910.01108 [Cs]*. http://arxiv.org/abs/1910.01108
4. Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., Peters, M., Schmitz, M., & Zettlemoyer, L. AllenNLP: A Deep Semantic Natural Language Processing Platform (Version 2.9.2) [Computer software]. https://doi.org/10.18653/v1/W18-2501
5. Gros, D., Sezhiyan, H., Devanbu, P., & Yu, Z. (2020). Code to comment "translation": Data, metrics, baselining & evaluation. Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering, 746–757. https://doi.org/10.1145/3324884.3416546
6. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *ArXiv:1907.11692 [Cs]*. http://arxiv.org/abs/1907.11692

Related Works:

CodeBERT. (2022). [Python]. Microsoft.
CodeBERT: A Pre-Trained Model for Programming and Natural Languages. (2020)
DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. (2020)
Deep Code Comment Generation. (2018)
RoBERTa: A Robustly Optimized BERT Pretraining Approach. (2019)
Knowledge Distillation-Based Multilingual Code Retrieval. Algorithms (2022)
On the Opportunities and Risks of Foundation Models (2021)
Local Google Colab for Running CodeBERT for Comment Generation (2020)

Resources:

HuggingFace: DistillBERT Documentation
Noisy Self-Knowledge Distillation for Text Summarization (2021)
Visual Notebook to Using BERT
Fine Tuning DistilBERT for MultiLabel Text Classification
BLEU Score Explained

Extra Resources:

1. *NLP: DeepCom Google Colaboratory*. (n.d.). Retrieved March 29, 2022, from https://colab.research.google.com/github/autosoft-dev/ml-on-code/blob/main/notebooks/27_11_2020/Code_Summarization.ipynb#scrollTo=6bO4r-iI8fgm
2. *NaturalCC*. (2022). [Python]. CGCL-codes. https://github.com/CGCL-codes/naturalcc (Original work published 2020)
3. CodeBERT github: https://github.com/microsoft/CodeBERT/tree/master/CodeBERT