Abstractive Question Answering on CodeQA dataset

Deval Srivastava
Department of Computer Science
Virginia-Tech
devalsrivastava@vt.edu

Chandrasekhar. P
Department of Computer Science
Virginia-Tech
chandrasekhardcs@vt.edu

Abstract

We propose to build a QA system that considers a code snippet as context and then tries to answer a question on the code in a coherent natural language. We propose to train our model on the CodeQA [11] dataset that contains code snippets along with corresponding question-answer pairs. We then experiment with various large language models designed for code and language. We evaluate the models and report performance of these models on both java and python datasets.

CCS Concepts: • **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

Keywords: Deep learning ,Code based Question answering, CodeQA, CodeBERT, CodeT5, RoBERTa, CodeXGlue, PLBART

ACM Reference Format:
Deval Srivastava and Chandrasekhar. P. 2018. Abstractive Question Answering on CodeQA dataset. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Deep Learning Project Report 'XX).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

In recent years with the advancement of deep learning and the availability of large-scale datasets. QA has received increasing attention from researchers. Alongside this growth there has been tremendous research in the space of deep learning for code. In this project, we motivate from both these fields and focus on task of Code Question Answering system which has the ability to read the code snippet and answer the questions about it. It essentially requires an understanding of both source code and natural language. QA-based systems have more specific guidance for models on what to generate compared to the code summarization task. Also, it is easier to be evaluated, since the output is more succinct, constrained, and targeted. Moreover, it can provide diverse information that can be leveraged to help perform a wide range of software engineering tasks, such as bug detection, specification inference, testing, and code synthesis.

A lot of work has been done on question-answering and machine reading comprehension, a few prior works have drawn attention to code question answering-based systems. QA-based code comprehension has direct use in education to facilitate a better learning experience for beginner programmers. Moreover, it can provide diverse information that can be leveraged to help perform a wide range of software engineering tasks, such as bug detection, specification inference, testing, and code synthesis.

2 Related Work

Our work's can be broadly considered to belong to a number of domains in machine learning and NLP, including Question answering, Transformer models, Code Based NLP and general NLP. In this related work section we perform a literature survey of related works and address the gap in literature. While doing that we also motivate our project and approach.

Question Answering (QA) has been one of the most important research tasks NLP. We can find use cases for question answering in fields ranging from information retrieval, chat bots and to now even in software engineering. Although QA task can be modelled in many ways, one of the styles of presenting a Question answering task is the Reading comprehension task, where the system is provided with a question and a context to help answer the question. There have been some popular datasets in this space that answer the question by directly picking out words from the context such as SQUAD [13] where Rajpurkar et al, provide answers in terms of selected spans from the input passage . Furthermore, there has been multiple-choice question answering datasets such as RACE[9] by Lai et al, where the model has to pick an answer from multiple-choice options. But these datasets enforce learning of superficial answers by selecting relevant text. In our work, we plan to focus on a subfield of questions popularly known as abstractive question answering where the answers are in natural language.

Deep Learning Project Report 'XX, May 09–, 2018,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

In this task, the answer is not directly perceivable to the model as spans or multiple choices. The model has to generate text based on the question, and context, this task forces the model to have a deep understanding of the context. This is also more similar to how a human being answers a question because we first understand the question and the answer then present it in our own language. There have been some datasets released for this task as well. MS-MARCO dataset[2] from Bajaj et al pose this problem where there are multiple contexts and a question and the objective is to generate a natural language answer. NarrativeQA dataset[8] by Kočiský et al, contains summaries and full texts of movies and raises questions that require the model to understand the relationships between people and the events that took place in the story. More recently, there has been the release of a dataset CodeQA [11] from Liu and Wan which includes code segments in python, java and the model has to answer logical questions on the code with natural language answers.

On the modelling side numerous approaches have been designed to solve QA tasks. Generally models that use recurrent units of some type to process data sequentially are used to solve these tasks. Attention based models like BiDAF[14] have also gained really high performance on these tasks. QAnet[17] is also one of the high performing approaches. It uses sepearable depthwise convolutions and self attention to encode the question and the context. then it uses context-query attention to find the correct spans. Recently transformers[15] have really changed the NLP landscape completely, from Vaswani et al.'s work to BERT[4] by Devlin et al. there has been tremendous growth. The question answering domain also has inherited exponential growth due to transformers. Works like SpanBERT[7] where Joshi et al. pushed the boundaries for QnA. For abstractive question answering domain models like BART[10] which is an encoder-decoder model have been designed by Lewis et al. and Raffel et al 's T5[12] a text to text-transform has been employed, these decoder style models such as GPT[3] Brown et al. can also be employed for this task. The success of transformers for abstractive QA motivates us to analyze their performance on codeQA.

There is a growing interest in automating software engineering (SE) using deep learning in the last few years. Vast sources of code in open source repositories and forums make deep learning feasible for SE tasks. In the past few years, code-based research has also grown tremendously both in the availability of datasets and the models that can solve the tasks. CodeBERT[5] is a pre-trained bimodal transformer similar to BERT that learns the semantic connection between natural language and programming language, our work will also use it as a foundation. Also since this is a generative task we can look at encoder-decoder style models. CodeT5[16] is a pre-trained encoder-decoder style model trained with a T5 style denoising loss. PLbart[1] is a unified pre trained encoder decoder transformer that was trained using an autoencoder style denoising loss.The task on which we are focusing is in a way similar to abstractive code summarization. A popular dataset for this task is the codeSearchNet[6] originally designed for code retrieval based on queries.

Table 1. Python vs Java data distribution

	Python	Java
Train-size	56,085	95,778
Dev-size	7,000	12,000
Test-size	7,000	12,000

From our limited literature survey, we observed that there has been relatively less research in the field of code based question answering tasks. With that knowledge we focus on the task of CodeQA which is an abstractive QA dataset. Motivated by the success of transformers on language based tasks we plan to employ them on code tasks and evaluate thier performance.

3 Approach

We propose the following plan, the dataset we will be focusing on in our research will be the codeQA dataset on both its python and java sections. For the model design, we will start off with baseline RoBERTa and CodeBERT and then we experiment with improved models. We will try known encoder decoder style models like CodeT5 and PLBart for this task and evalute performance.We plan to use metrics similar to generation tasks.The following sections below elaborate more on our plan.

3.1 Dataset Description

CodeQA dataset is the first of its kind diverse free-form question-answering dataset and at the time of its release, it was the first of its kind code comprehension dataset. It utilizes code comments ("docstrings") that are suitable to generate QA pairs using syntactic rules and semantic analysis. This also ensures that QA pairs generated from the code comments are corresponding to the code snippets. It targets various types of QA pairs such as Wh-questions which include what, when, where, and so on, Yes/No questions. More specifically, comments are transformed into dependency trees and converted to fit question templates that are invoked by semantic role labels.

Table 2. Distribution of questions after partitioning

Question Type	Percentage
What	67.24%
How	8.93%
Where	5.85%
When	6.89%
Why	1.02%
For what purpose	5.08%
Yes/No	2.86%
Other	2.13%

Table 3. Dataset Statistics (EDA Output)

Metric	Python	Java
Mean of No of tokens per code	49.8	119.5
S.D of No of tokens per code	46.5	165.5
Mean of No of tokens per question	8.1	9.5
Mean of No of tokens per answer	4.1	4.7
No. of unique tokens in codes	108,571	27,504

CodeQA contains 119,778 QA pairs for 56,545 Java codes and 70,085 QA pairs for 44,830 Python codes. The generated QA pairs are classified into four categories: functionality, purpose, property, and workflow. The table below summarizes the statistics of the question types in the dataset.

3.2 Data biases and Limitations

The training dataset used for pre-training includes user-written comments. It is possible that these comments would include information that may be subjective to that user regarding that code. Since code can be interpreted slightly differently, this can be considered to be bias within the data. Moreover, its possible that the comments may also have social biases present. Which a model may learn as it trains on it. Furthermore there may be a case of over reliance on the model once its been trained. Developers and software practitioners should bear in mind that generated output should be only taken as reference that require further correctness and security checking.

Since this is the first of its kind abstractive code question answering datasets there are some visible limitations. Firstly, it doesnt offer a large variety in the terms of questions a user may want to know about code. It only seems to offer boilerplate style of questions all of which follow a template. Secondly, the code snippets in the dataset are rather small, in software engineering we generally do observe that code can be fairly long and users may also ask long form questions which reuquire long explainations.

3.3 Pre-processing

The input code is pre-processed before feeding into the model. The following operations are done

- Lowercase code tokens
- Replace string and numbers in code tokens with generic symbols Ex: <STRING>, <NUM>.
- Filter punctuation in answers
- Spliting identifiers according to camelCase and snake-case

3.4 Model Description

Almost all the state-of-the-art question answering systems are built on top of end-to-end training and pre-trained language models. For our project, we plan to use the Seq2seq architecture as it was originally designed for text-to-text generation tasks, and we plan to experiment with different pre-trained language models as encoders and decoders.

Sequence-to-Sequence-Models are based on an encoder-decoder model architecture where both are separate models combined into a model. The encoder network generates an representation of the input in a smaller dimension, while the decoder takes the output of encoder and generates the output representation.

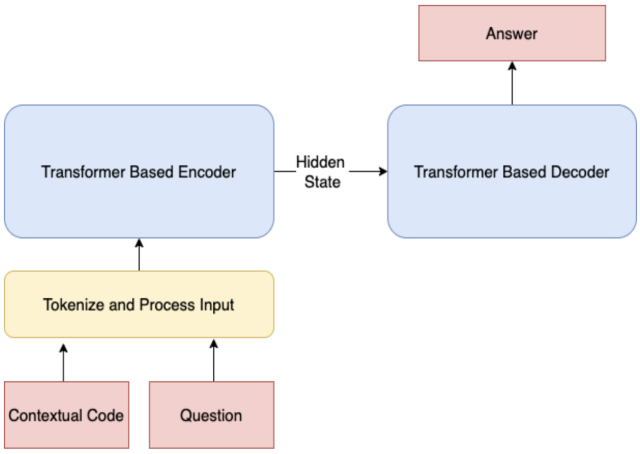

Figure 1. Preliminary architecture for the abstractive QA task

RoBERTa is a large language model that has been trained on a text corpus. RoBERTa was trained using dynamic masking, Full-sentences without NSP Loss, Large mini-batches and a larger byte-level BPE. Compared the other architectures we utilize going forward, this one hasnt been trained on "code" and is simply a language model. For this reason, this model will act as a good baseline since ideally code based models should outperform this model.

CodeBERT is a Transformer based encoder-decoder model where the encoder is initialized with codeBERT. codeBERT is large language model encoder similar to RoBERTa in architecture that has been trained on natural- language and Python, Java programming languages. Since this is an encoder only model, we added a a transformer based decoder with 6 layers and 12 attention heads on top of codeBERT to generate language. We fine-tune this architecture on the CodeQA task and analyze results.

CodeBERT shows good results for all the tasks in the code domain. Specially, it shows a higher score than other pre-trained models in the code to natural language tasks. In addition CodeBERT uses Byte Pair Encoding (BPE) tokenizer used in RoBERTa and doesn't generate unk tokens in code domain input.

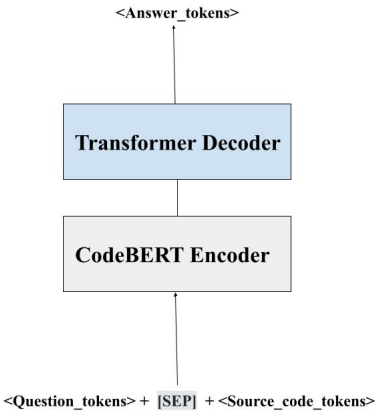

Figure 2. CodeBERT Generation for QA Task

CodeT5 model builds on the similar T5 architecture but incorporates code-specific knowledge to endow the model with better code understanding for various languages and utilizes an encoder-decoder-based model that was trained using the denoising objective. Compared to codeBERT it has the decoder included in the pretraining as well. Also, their training dataset included developer-assigned identifiers which provide extra information. It's Bimodel Dual Generation(dual-gen) objective primarily boosts NL-PL tasks such as text-to-code and code-summarization.

CodeT5 yields new state-of-the-art results on fourteen sub-tasks in CodeXGLUE benchmark.It supports both understanding and generation tasks and also allows for multi-task learning

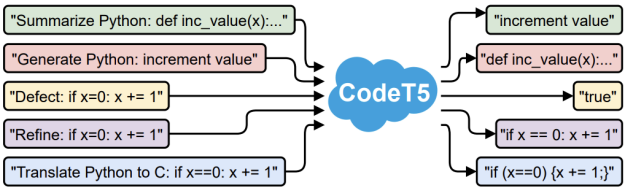

Figure 3. CodeT5 code understanding and generation

In figure 1 we illustrate an architectural design for our models. The input to the model will be the contextual code and the question. Contextual code will be tokenized through a pre-trained tokenizer for code and text will be tokenized through a text tokenizer. The inputs will be combined in some way and processed. We will use transformer-based models and furthermore we are going to be experimenting with these models mentioned above and finetune them on our task. We also plan to introduce some modifications to the model to achieve better performance on the metrics.

PLBART is a sequence-to-sequence model pre-trained on a large collection Java and Python functions and natural language descriptions collected from Github and StackOverflow, respectively.

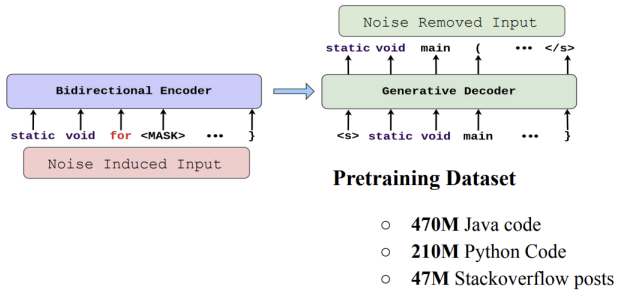

Figure 4. PLBART architecture

PLBART is pre-trained via denoising autoencoding (DAE) and uses three noising strategies: token masking, token deletion, and token infilling. PLBART is effective in program understanding and learns program syntax rules that are crucial to program semantics. It performs well on various code to code, text to code and code to text tasks.

3.3 Implementation Details We have implemented our models and training pipelines in python3.9. We use the huggingface library to download and use pretrained models. For

Table 4. Performance of various models on Python dataset

Model	BLEU	ROUGE	METEOR	EM	F1-Score
RoBERTa	29.26	20.47	6.28	2.30	21.24
CodeBERT	33.78	30.35	12.63	4.77	31.51
CodeT5	35.77	34.66	14.25	6.64	36.19
PLBart	34.90	33.02	11.20	6.02	30.64
RoBERTa with Beamsearch	28.55	22.42	6.85	2.96	23.65
CodeBERT with Beamsearch	35.90	31.90	13.31	5.61	33.15
CodeT5 with Beamsearch	35.27	34.23	14.52	6.86	35.66
PLBart with Beamsearch	32.88	31.60	13.51	5.59	32.95

Table 5. Performance of various models on Java dataset

Model	BLEU	ROUGE	METEOR	EM	F1-Score
RoBERTa	30.28	25.92	8.08	3.90	26.88
CodeBERT	32.10	27.89	9.93	6.05	28.86
CodeT5	33.35	31.85	11.87	31.41	33.12
PLBart	32.64	30.06	11.53	7.00	31.13
RoBERTa with Beamsearch	28.04	22.60	6.41	4.01	23.71
CodeBERT with Beamsearch	33.18	29.60	10.69	6.41	6.64
CodeT5 with Beamsearch	31.58	29.64	10.79	5.46	30.78
PLBart with Beamsearch	30.20	29.61	11.20	6.02	30.64

the tokenization we use pretrained tokenizers and vocabularies for the code available within hugggingface. For training we design scripts to train the model on CUDA. We also borrowed some of the code from the CodeXGlue repositories. All of our training has been done on an nvidia T4 GPU with 16 GiB memory where it takes roughly 10 hours to complete 10 epochs. To train our models we set the following **hyper parameters**.

- Batchsize= 16
- Epochs= 10
- Python-source-length= 100
- Python-target-length= 30
- Java-souce-length= 300
- Java-target-length= 30
- Learning-rate= $5e-5$
- Optimizer : Adam
- Beam-size :10

4 Evaluation and Results

We plan to provide qualitative examples for both python and java samples. For quantitative analysis of our model, we will be using the following metrics. As this is primarily a generation task many of the metrics are similar to other summarization and translation tasks.

4.1 Qualitative Evaluation

To better understand where the model makes mistakes, we took 5-examples (Shown in figure 5) from the test set for further analysis. In all the cases the model outputs a plausible answers. This does show that our model has understanding of code to some extent.

In examples 1-3 we observe that the answers are very close to the actual answers and also make sense as legible answers. Since this a generative model we cant expect the model to reach the exact required performance, but even then the answers seem to be accurate. For example 4 we see that the model is slightly confused in the order of events, but that is to be expected since this question requires understanding of the code and also of the temporal events in the code.

In example-5, the model output is $< sql-log-message >$, where as the ground truth label is $< a\ log\ event >$, prediction is intelligent and sensible here in this example as the code contains $< InsertInto >$ string it is adding SQL in to the model output.

4.2 Quantitative Evaluation & Metrics

BLEU

BLEU is a metric that is used to compare a model-generated sentence to a number of reference sentences, Bleu score is

Code	Prediction	Gold	Question
def str2css sourcestring colors None title '' markup 'css' header None footer None linenumbers 0 form None if markup lower not in ['css' 'xhtml'] markup 'css'stringIO StringIO StringIO parse Parser sourcestring colors colors title title out stringIO markup markup header header footer footer linenumbers linenumbers parse format form stringIO seek 0 if form None return parse _sendCSSStyle external 1 stringIO read else return None stringIO read	to a css style string	to colorized css / html	What does a code string convert ?
def _local_server_get url session request HttpRequest request method 'GET' request session session view args kwargs resolve url response view request *args **kwargs return response content	a get request	request for an in - process api	What do a server - server get ?
def _saferepr obj repr py io saferepr obj if py builtin _istext repr t py builtin textelse t py builtin bytes return repr replace t '\n' t '\\n'	a safe repr of an object	a safe repr of an object for assertion error messages	What does the code get ?
def mongo_db registry xml_parent data mongodb XML SubElement xml_parent 'org jenkinsci plugins mongodb MongoBuildWrapper' mongodb set 'plugin' 'mongodb' mapping ['name' 'mongodbName' None 'port' 'port' '' 'data-directory' 'dbpath' '' 'startup-params' 'parameters' '' 'start-timeout' 'startTimeout' 0]convert_mapping_to_xml mongodb data mapping fail_required True	before running a mongodb	while running the build	When does mongo - db build wrapper initialize a mongodb database ?
def dblog msg module return insert 'INSERTINTOlogVALUES ? ? ' module _timestamp msg rstrip	sql_log_message	a log event	What does the code insert ?

Figure 5. Error Examples - Processed-Code, Prediction, Gold, Question

calculated from the common presence of 1-4 N-grams between the model outputs and targets.

METEOR
METEOR (Metric for Evaluation of Translation with Explicit Ordering) is a metric for the evaluation of machine translation output. The metric is based on the harmonic mean of unigram precision and recall, with recall weighted higher than precision. METEOR score was developed in response to some pitfalls in the BLEU metric as a high BLEU score wasn't well correlated with a good generated sentence. The METEOR metric scores generated sentences by aligning them against references based on exact, stem, synonym, and paraphrase matches between words and phrases.

Exact Match (EM)
This is a binary measure indicating whether the output matches the ground truth answer exactly.

F1 Score
To Calculate the F1 Score for question answering we consider the shared words between the generated response and reference response to be the true positive. So from there, We can calculate the precision as the common words upon the total words in prediction, and similarly recall is calculated as common words upon total words in reference output.

Rouge score
This measure the number of contiguous words (referred to as n-grams) occured in model predicted answer compared to the reference answer.

$$ROUGE-2 = \frac{\sum_{s \in \{RefSummaries\}} \sum_{bigrams\ i \in S} \min(count(i,X), count(i,S))}{\sum_{s \in \{RefSummaries\}} \sum_{bigrams\ i \in S} count(i,S)}$$

Figure 6. Evaluation-Metric Rouge-2 score

5 Analysis

In the results tables 4 and 5 we have documented the results that we observed from the models on the test splits. Starting off with the baseline performance, we see that a language only model such as roberta isnt a good fit for this task. All our other models that have been pretrained on code-like data have been able to outperform the baseline. Other than this we observe a few trends in the results, which we elaborate below.

Overall Performance on Python dataset is higher compared to Java dataset. This practically makes sense as python syntax is much more readable compared to java.

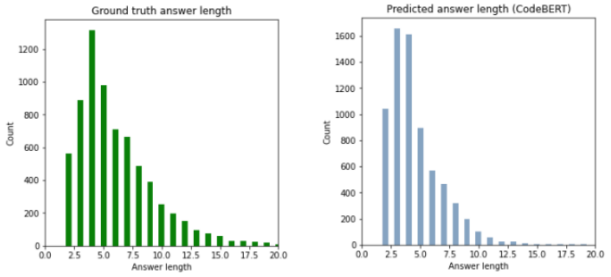

Figure 7. Ground truth, Predicted answer length distribution

The above figure 7. The distribution of the predicted answer length and the ground truth answer length have some similarity and we observed that for the question types "Yes/No" and "T/F" both predicted and ground truth answer lengths are same

Beam search always didn't help to improve the model accuracy all the time. In case of CodeBERT beam-search helped in performing better, where as in CodeT5 and PLBart the models were able to perform better with simple greedy decoding.

Performance breakdown Error analysis Table 6 shows the performance breakdown for different type of questions. To better understand when the model makes mistakes, we took 5 examples from the development set for further analysis. See Table 7 for the concrete examples

CodeT5 achieved the highest performance on Java set and **CodeBERT with beamsearch received the highest performance for python** set on the CodeQA dataset. We speculate that CodeT5 performs better on the java set java is a more complex language and CodeT5 is a model with a lot of parameters.

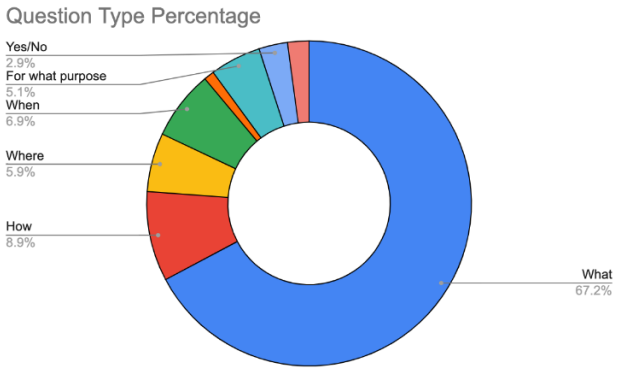

Figure 8. Training data Question type distribution

From Table 6 we can observe that the performance breakdown of different type of questions. From this we can better understand when the model makes mistakes. The top performing **Wh** question type is **Where**.

Another interesting note would be the higher performance of Others question type, which is due to the presence of higher number of **Yes or No** question types. We also observe that questions of the type **for what purpose** have less performance compared to some other questions, we believe this may be due to the fact that these type of questions are quite complex and the answers require deep knowledge of the code in context.

6 Lessons Learned

While working on this project we learnt how to approach a state of-the-art Deep learning problem. We first needed to get familiar with the data, requirements and environment, quickly build a working baseline model, and improve the performance with more complex modifications. An effective method to improve an existing model can be tuning the hyper-parameters that can make a big difference on the performance of the model.

Analysis of the model output is also a crucial step in improving the model performance. We can exactly understand which data class points need to be added in-order to improve the overall accuracy (In our case 6 we can add more questions types starting with How, since there is a higher

Table 6. Performance Analysis of various question types on test dataset

Measure	For what purpose	What	How	Where	When	Others
BLEU	26.67	35.23	32.45	40.17	31.13	80.46
EM	0.98	3.65	2.01	4.96	3.32	60.67
ROUGE	25.70	26.45	15.64	33.53	19.05	67.28
F1-Score	41.72	48.62	38.55	51.53	40.05	79.97

probability that this change will increase the overall accuracy of the model).

We also understood the importance of logging mechanism for debugging the model. Logging systematically will allow us to methodologically analyze how performance changes with different hyper-parameters and also it can be used as a reference for documentation.

7 Broader Impact

In this work we tackle the problem of trying to answer questions based on code snippets. This problem finds its motivations from all fields that utilize code and specifically software engineering. In major organizations there are massive codebases expanding more than millions of lines of code, due to the scale it becomes extremely tedious to understand which purpose a function may serve. This problem becomes even more challenging for new or less experienced members of a software engineering team when they try to work with a large unseen codebase. In this scenarios a more experienced engineer may pitch in to help resulting in wasted time for both the engineers. Our work tries to offer a solution for this challenge. Where one may ask questions for code in natural language to a system which may be running our models internally. We believe that our models and pipelines eventually could be wrapped in popular editors where users may quickly ask questions on a selected piece of code to get a gist of purpose.

8 Conclusion

In this work we solve the task of answering questions with natural language responses on python and java code. Firstly, we performed exploratory data analysis to understand the dataset. Then we trained and evaluated 4 models for both the programming languages. Alongside this we conduct experiments evaluating the performance of beam search when used during generation. For all our experiments we report results and then perform a qualitative analysis of a few selected results from our model, from that we find that the model actually has learnt to understand nuances of the code and is able to generate answers similar to a human.

From our results we observe that even though the models are effective are solving this task, there is overfitting on the data. In future work we will try to address this problem, one of the ways can be to add more regularization methods such as drop out. Furthermore this problem can be addressed by incorporating more code related data from github with more questions. Also when we do that we can address some other types of problems in the data like the limited number of questions and the short length of code snippets. To conclude we demonstrate a few methods to solve this problem of code question answering but still a number of improvements and modifications can be further performed.

In future work we can add more relevant data to the low performing question types observed in table 6. Which will eventually boost the overall accuracy of the QA Model.

9 Work Distribution

In the inital phases of the project both the team members worked on the idea conception and literature survey. Later on we decided an approach to solve this problem and listed down the tasks and models that will need to be developed. Both the team members contributed to development of utility and processing scripts but specifically each team member worked on the following models.

- **Deval**: Worked on the development and evaluation of RoBERTa and CodeT5 for this project on Java and python datasets.
- **Chandrasekhar**: Worked on the development and evaluation of CodeBERT and PLBart for this project on Java and python datasets.

Once the development phase was completed both team members worked on the report and presentation.

10 Proposal Feedback

We received valid and constructive feedback for our proposal. Some of the main points highlighted are listed below.

1. Qualitative evaluation is missing from the report
2. Authors are only experimenting with two models.

We analyzed the feasibility of the feedback and were able to address them to the best of our ability in this final report. Firstly, we have added qualitative analysis of few selected

results of the model. We analyze the same results. Secondly, In our inital report we had reported experimentation with two models CodeBERT and CodeT5. Since then we have been able to experiment with more models, specifically we are now using RoBERTa as a baseline and PLBart as another model.

References

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. Unified pre-training for program understanding and generation. *ArXiv*, abs/2103.06333, 2021.

[2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[6] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*, 2019.

[7] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.

[8] Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

[9] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[10] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.

[11] Chenxiao Liu and Xiaojun Wan. Codeqa: A question answering dataset for source code comprehension. *arXiv preprint arXiv:2109.08365*, 2021.

[12] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[13] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[14] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *ArXiv*, abs/1611.01603, 2017.

[15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[16] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859*, 2021.

[17] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *ArXiv*, abs/1804.09541, 2018.