BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

HYDERABAD CAMPUS.

A Report on

Pragmatic Code Commenting Using Deep Learning Techniques.

Submitted by

Ashutosh Wagh (2022H1030052H)

Birla Institute of Technology and Science-Pilani

Hyderabad Campus.

Certificate

The Research Practice report entitled "**Pragmatic Code Commenting Using Deep learning**" submitted by Ashutosh Wagh (ID No. 2022H1030052H) and in fulfillment of the requirements of the course BITS G540 RESEARCH PRACTICE, embodies the work done by them under my supervision and guidance.

Date: 10-5-23

Prof. Dr. N.L Bhanu Murthy

HOD - Department of CSIS
BITS Pilani
Hyderabad Campus

Table of Contents

Abstract

Code comments are an important part of writing good code. It plays a paramount role in various software development tasks such as code reviews, maintenance, rewriting, reusability, and several purposes, including explaining the purpose of a section of code, providing documentation for functions and classes, and warning about potential issues or bugs, re-commenting mismatched or outdated comments. Our work focuses on the use of deep learning techniques for code commenting, with a focus on transformer-based models. We explore the use of BERT and CodeBERT for code commenting, providing an overview of their architectures and training methodologies.

Literature review

Transformers[4]:

The Transformer model was first introduced in 2017 by Vaswani et al. in the paper "Attention Is All You Need". The paper proposed a new architecture for machine translation that replaced the traditional recurrent neural network (RNN) with a self-attention mechanism. The self-attention mechanism allowed the model to attend to all words in the input sentence, rather than relying on a fixed-length context window.

BERT[2]:

In 2018, Devlin et al. introduced BERT (Bidirectional Encoder Representations from Transformers) in the paper "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". BERT is a language model pre-trained on a large corpus of text using a masked language modeling objective. It was trained to predict missing words in sentences by masking some of the input tokens and learning to predict the masked tokens based on the surrounding context. BERT achieved state-of-the-art results on a wide range of natural language processing tasks, including text classification, question-answering, and named entity recognition.

CodeBERT[1]:

In 2020, Feng et al. proposed CodeBERT in their paper "CodeBERT: A Pre-Trained Model for Programming and Natural Language Processing". CodeBERT is a pre-trained model for programming language processing that is based on the BERT architecture. It was trained on a large corpus of code and natural language data to learn representations of code snippets and their accompanying natural language descriptions. CodeBERT was evaluated on a variety of tasks, including code retrieval, code generation, and code comment generation, and achieved state-of-the-art results on many of them.

Introduction

Code commenting is an important task in software engineering, as it can help improve the maintainability, readability, and understandability of software code. Code comments provide valuable information about the purpose, functionality, and usage of code, enabling developers to easily comprehend and modify code in the future. However, manually generating comments can be a time-consuming and error-prone task, and developers may not always have the necessary domain knowledge or expertise to write accurate and informative comments.

To address these challenges, researchers have explored the use of deep learning techniques for automatic code commenting. In recent years, transformer-based models such as BERT and CodeBERT have emerged as a powerful and flexible framework for natural language processing tasks, including code commenting. These models can learn to automatically generate comments that describe the functionality of code, without requiring manual annotation or intervention from developers.

The application of transformer-based models to code commenting presents several advantages over traditional methods. Firstly, transformer models can capture the complex dependencies and long-range dependencies between the source code and its associated comments, improving the accuracy and informativeness of the generated comments. Secondly, these models can learn to generate comments that are consistent in style and tone, which can help improve the readability and maintainability of code. Finally, transformer-based models can reduce the workload of developers by automating the tedious and repetitive task of generating code comments.

Methodology

The pre-trained model is fine-tuned using CodeBERT[1]. This model uses the dataset which is fetched from Hugging Face website using APIs. It is used to teach a RoBERTa based model to represent code and natural language in a useful way. This practice of teaching these large language models to represent text in a useful way is common practice now since these representations have been shown to be helpful in fine tuning these models on other tasks. The CodeBERT paper showed these representations are helpful by finetuning them on the programming task of code search and comment generation. We have done more preprocessing, and our model generates inline comments of code snippets and not just method level comments.

It combines two different training objectives that have been shown to be useful for natural language. The Masked Language Modeling objective (MLM)[2], which is from the original BERT paper, and Replaced Token Detection (RTD) objective[3]. The MLM objective is where we randomly mask out parts of the text that we feed into the model and ask the model to predict those masked out pieces. The RTD objective is where random tokens in the text are replaced and the model has to determine which of these tokens are replaced. However, to make it harder for the model, these replaced tokens attempt to be plausible alternatives and not just random words. The CodeBERT model actually used a n-gram based model to generate these alternatives whereas the ELECTRA paper used a small BERT based model.

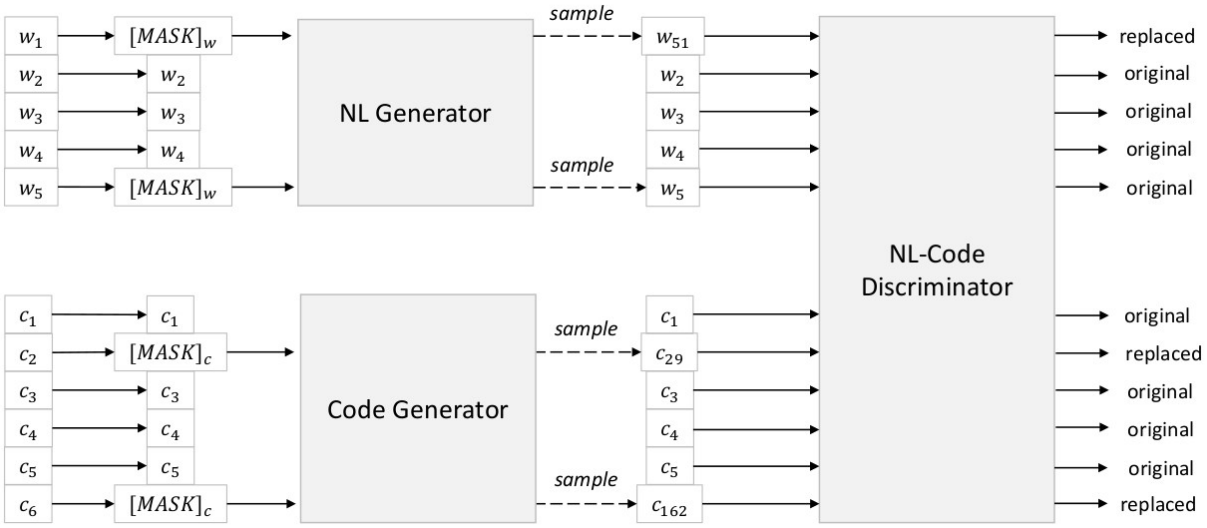


Figure 1: A diagram describing the new token detection objective.

Language models like NL and code generators produce plausible tokens for masked positions based on the contexts around them. The targeted pre-trained model is the NL-Code discriminator, and it is trained by identifying plausible alternatives in token samples drawn from the NL and PL generators. For creating all-purpose representations during the fine-tuning stage, an NL-Code discriminator is used. The fine-tuning phase eliminates both NL and code generators.

Dataset

The dataset used in our project is taken from the Hugging Face website. We have fetched it using APIs from the website. It is the same dataset which is used in the CodeBERT paper. The name of the dataset is called CodeSearchNet. It contains a piece of code and corresponding comments to it. It contains over 2 million lines of Java code collected from open-source repositories on GitHub, along with natural language queries that were extracted from the corresponding issues and pull requests. The dataset is divided into training, validation, and test sets, each with different natural language queries and code snippets. The training set contains about 1.6 million code snippets, the validation set has about 150,000 code snippets, and the test set has about 300,000 code snippets. We have used only 0.5% of the dataset, as it requires a long time to train the model.

Implementation

Step 1:
Installing the necessary packages and downloading the data.

```
[ ] !pip install transformers
    !pip install bleu

[ ] from google.colab import drive
    drive.mount('/content/drive')

[ ] import requests
    import pandas as pd
    import math
    import re

[ ] r = requests.get("https://datasets-server.huggingface.co/parquet?dataset=code_search_net&config=java&split=train")
    # r = requests.get("https://datasets-server.huggingface.co/parquet?dataset=blog_authorship_corpus")
    j = r.json()
    urls = [f['url'] for f in j['parquet_files'] if f['split'] == 'train']

[ ] urls

[ ] df = pd.concat([pd.read_parquet(url) for url in urls])

[ ] final_dataset = df[['whole_func_string', 'func_documentation_string']]
    final_dataset.to_csv('/content/drive/MyDrive/RP/dataset.csv')

[ ] dataset_10_percent_train = final_dataset[:math.ceil(len(final_dataset)*0.05)]
    dataset_10_percent_test = final_dataset[math.ceil(len(final_dataset)*0.05):math.ceil(len(final_dataset)*0.25)].sample(frac=0.1)
    dataset_10_percent_valid = final_dataset[math.ceil(len(final_dataset)*0.025):math.ceil(len(final_dataset)*0.03)].sample(frac=0.05)

[ ] dataset_10_percent_train = dataset_10_percent_train.rename(columns={'whole_func_string': 'mthd', 'func_documentation_string': 'cmt'})
    dataset_10_percent_test = dataset_10_percent_test.rename(columns={'whole_func_string': 'mthd', 'func_documentation_string': 'cmt'})
    dataset_10_percent_valid = dataset_10_percent_valid.rename(columns={'whole_func_string': 'mthd', 'func_documentation_string': 'cmt'})

    dataset_10_percent_train.to_csv('/content/drive/MyDrive/RP/dataset_10_percent_train.csv')
    dataset_10_percent_test.to_csv('/content/drive/MyDrive/RP/dataset_10_percent_test.csv')
    dataset_10_percent_valid.to_csv('/content/drive/MyDrive/RP/dataset_10_percent_valid.csv')
```

Fig.2 : Code describes installing the necessary packages and downloading the data.

```
[ ] dataset = pd.read_csv('/content/drive/MyDrive/RP/dataset.csv')
```

```
[ ] dataset_10_percent_train = pd.read_csv('/content/drive/MyDrive/RP/dataset_10_percent_train.csv')
    dataset_10_percent_test = pd.read_csv('/content/drive/MyDrive/RP/dataset_10_percent_test.csv')
    dataset_10_percent_valid = pd.read_csv('/content/drive/MyDrive/RP/dataset_10_percent_valid.csv')
```

```
[ ] dataset_10_percent_train.drop(['Unnamed: 0'], inplace=True, axis=1)
    dataset_10_percent_test.drop(['Unnamed: 0'], inplace=True, axis=1)
    dataset_10_percent_valid.drop(['Unnamed: 0'], inplace=True, axis=1)
```

```
dataset_10_percent_train
```

	mthd	cmt
0	public List<Point2D> apply(PointFeature featur...	calculates the convex hull of the specified ar...
1	public void publish(int qos, String payload) t...	@param qos Quality of Service (0, 1, or 2) use...
2	public void setFactor(int factor) {\r\n ...	Set Contrast adjusting factor, [-127, 127].\n\...
3	@Override\n public ImageSource apply(ImageS...	Expects a height mat as input\n\n@param input ...
4	public <L extends Listener> void pushEvent(Eve...	Pushes the event onto the event stack. This ac...
...	...	...
22718	@NullSafe\n public static String[] toAssociat...	Converts the given {@link Map} into an associa...
22719	public static <K, V> Map<K, V> transform(Map<K...	Transforms the values of the given Map with th...
22720	@Override\n public <E> List<E> sort(final Lis...	Uses the Shell (Insertion) Sort algorithm to s...
22721	public static Date toDateFromIso8601(String s)...	Parses a text string with a date in ISO 8601 f...
22722	public static String toIso8601(Date date) {\n ...	Returns a ISO8601 string of the given date.\n@...

22723 rows × 2 columns

```
[ ] dataset_10_percent_test
```

	mthd	cmt
0	@Nonnull\n public DescribeCommand tags(@Nulla...	<pre>--tags</pre>\n<p>\nInstead of using only ...

Fig.3 : Code describes displaying the training data.

```
dataset_10_percent_test
```

	mthd	cmt
0	@Nonnull\n public DescribeCommand tags(@Nulla...	<pre>--tags</pre>\n<p>\nInstead of using only ...
1	public void reset(String resourceGroupName, St...	Reset Live Event.\nResets an existing Live Eve...
2	protected DCTree inlineText() throws ParseExce...	Read plain text content of an inline tag.\nMat...
3	public HalFormsTemplate andContentType(MediaTy...	Returns a new {@link HalFormsTemplate} with th...
4	public void update(String resourceGroupName, S...	Configures the HTTP settings on the specified ...
...	...	...
9084	public List<ValidationError> validateInput() {...	Returns a list of validation errors of the for...
9085	private SearchNode getSearchNode()\n {\n ...	Find a SearchNode for use by the current threa...
9086	public static IMessageReceiver createMessageRe...	Creates a message receiver to the entity.\n@pa...
9087	protected void initiateCommsHandshakingImpl(fi...	Actually performs the handshake.\n\n@param ser...
9088	public JenkinsServer createJob(FolderJob folde...	Create a job on the server using the provided ...

9089 rows × 2 columns

```
[ ] dataset_10_percent_valid
```

	mthd	cmt
0	public static TableCellButtonRenderer newTable...	Factory method for creating the new {@link Tab...
1	@Pure\n\tpublic BusStop getBusStop(UUID id) {\...	Replies the bus stop with the specified id.\n\...
2	@Pure\n\tpublic static File getUserConfigurati...	Replies the user configuration directory for t...
3	@Pure\n\tpublic static int getPreferredColor()...	Replies the default color for map elements.\n\...
4	private String getWthInsiCodeOr(Map wthData) {...	Get the 4-bit institute code for weather file ...
...	...	...
109	@Pure\n\tpublic static String lowerEqualParame...	Parameter A must be lower than or equal to Par...
110	public static <K1, K2, V> ImmutableSetMultimap...	Creates a copy of the supplied multimap with i...
111	@Pure\n\tpublic static boolean intersectsOrien...	Compute intersection between an OBB and a caps...

Fig.4 : Code describes displaying the testing and valid data.

Step 2:
Then, we perform data cleaning. First, we remove any non-ascii characters to simplify the problem so that the model only has to think about generating English comments.

```
# collapse
# From https://stackoverflow.com/a/27084708/5768407
def is_ascii(s):
    '''
    Determines if the given string contains only ascii characters

    :param s: the string to check
    :returns: whether or not the given string contains only ascii characters
    '''
    try:
        s.encode(encoding='utf-8').decode('ascii')
    except UnicodeDecodeError:
        return False
    else:
        return True

df_trn = dataset_10_percent_train[dataset_10_percent_train['mthd'].apply(lambda x: is_ascii(x))]
df_val = dataset_10_percent_valid[dataset_10_percent_valid['mthd'].apply(lambda x: is_ascii(x))]
df_tst = dataset_10_percent_test[dataset_10_percent_test['mthd'].apply(lambda x: is_ascii(x))]

df_trn = dataset_10_percent_train[dataset_10_percent_train['cmt'].apply(lambda x: is_ascii(x))]
df_val = dataset_10_percent_valid[dataset_10_percent_valid['cmt'].apply(lambda x: is_ascii(x))]
df_tst = dataset_10_percent_test[dataset_10_percent_test['cmt'].apply(lambda x: is_ascii(x))]

len(df_trn), len(df_val), len(df_tst)
```

```
(22462, 113, 8965)
```

```
[ ] df_trn
```

	mthd	cmt
0	public List<Point2D> apply(PointFeature featur...	calculates the convex hull of the specified ar...
1	public void publish(int qos, String payload) t...	@param qos Quality of Service (0, 1, or 2) use...
2	public void setFactor(int factor) {\r\n ...	Set Contrast adjusting factor, [-127, 127].\n\...
3	@Override\n public ImageSource apply(ImageS...	Expects a height mat as input\n\n@param input ...
4	public <L extends Listener> void pushEvent(Eve...	Pushes the event onto the event stack. This ac...
...	...	...
22718	@NullSafe\n public static String[] toAssociat...	Converts the given {@link Map} into an associa...

Fig. 5 : Code describes data cleaning performed on the data, also display the modified training data.

Step 3:
Now we'll add in the additional pairs of code snippets/inline comments.

```
# collapse
from tqdm.auto import tqdm

def get_inline_pairs(mthd):
    '''
    Get all pairs of inline comments and corresponding code snippets

    :param mthd: the method to retrieve the pairs of comments and corresponding
    code snippets from
    :returns: all pairs of comments and corresponding code snippets
    '''
    pairs = [[]]

    comment = False
    bracket = False
    indent_lvl = -1
    lines = mthd.split("\n")
    for line in lines:
        if "//" in line and not bracket and not "://" in line:
            pairs[-1].append(line)
            if '\t' in line:
                indent_lvl = line.count('\t')
            else:
                indent_lvl = line.split("//")[0].count(' ')
            comment = True
            bracket = False
        elif comment:
            if '{' in line and not bracket:
                bracket = True
                pairs[-1].append(line)
            elif '}' in line:
                line_indent = -1
                if '\t' in line:
                    line_indent = line.count('\t')
                else:
                    line_indent = line.split("//")[0].count(' ')
                if indent_lvl == line_indent:
                    pairs[-1].append(line)
                if not bracket:
                    pairs.append([])
                    comment = False
                    bracket = False
            elif line.isspace() or line == '' and not bracket:
                pairs.append([])
```

Fig. 6 : Code describes the adding of additional pairs of code snippets/inline comments.

```
        elif comment:
            if '{' in line and not bracket:
                bracket = True
                pairs[-1].append(line)
            elif '}' in line:
                line_indent = -1
                if '\t' in line:
                    line_indent = line.count('\t')
                else:
                    line_indent = line.split("//")[0].count(' ')
                if indent_lvl == line_indent:
                    pairs[-1].append(line)
                if not bracket:
                    pairs.append([])
                    comment = False
                    bracket = False
            elif line.isspace() or line == '' and not bracket:
                pairs.append([])
                comment = False
            else:
                pairs[-1].append(line)

    # Convert pairs into proper format of (code snippet, inline comment) dataframe
    code_snippets   = []
    comments        = []
    for pair in pairs:
        if pair and len(pair) < 5:
            code    = []
            comment = []
            skip = False
            for line in pair:
                if "TODO" in line: break
                if "//" in line:
                    comment.append(line.replace('//', ''))
                else:
                    code.append(line)
            if len(code) > 1 and len(comment) > 0:
                        code_snippets.append('\n'.join(code))
                        comments.append('\n'.join(comment))

    pairs = pd.DataFrame(zip(code_snippets, comments), columns = ["mthd", "cmt"])
    return pairs
```

Fig. 7 : Code describes the adding of additional pairs of code snippets/inline comments.

Step 4:

We also remove pairs where the size of the code is smaller than the comment. This is because we found that in these cases the comments contain a bunch of extra information that the model won't have access to such as how the method is being used by other methods in the software system.

```
def add_inline(df: pd.DataFrame) -> pd.DataFrame:
    '''
    Helper function to go through all methods in a given dataframe and add all
    pairs of inline comments and corresponding code snippets

    :param df: the dataframe to retrieve and add all pairs of inline comments
    and corresponding code snippets to
    :returns: a new dataframe with the newly added pairs of inline comments and
    corresponding code snippets
    '''
    new_df = df[df['mthd'].str.contains("//")]
    all_pairs = []
    for mthd in tqdm(new_df.mthd.values):
        pairs = get_inline_pairs(mthd)
        all_pairs.append(pairs)

    df_pairs = pd.concat([pairs for pairs in all_pairs])
    return pd.concat([df, df_pairs])

df_trn = add_inline(df_trn)
df_val = add_inline(df_val)
df_tst = add_inline(df_tst)

len(df_trn), len(df_val), len(df_tst)
```

```
100% 3783/3783 [00:03<00:00, 725.06it/s]
100% 28/28 [00:00<00:00, 518.00it/s]
100% 2493/2493 [00:02<00:00, 1218.71it/s]
(23286, 118, 9448)
```

```
[ ] # collapse
    df_trn = df_trn[df_trn.apply(lambda row: len(row.mthd) > len(row.cmt), axis = 1)]
    df_val = df_val[df_val.apply(lambda row: len(row.mthd) > len(row.cmt), axis = 1)]
    df_tst = df_tst[df_tst.apply(lambda row: len(row.mthd) > len(row.cmt), axis = 1)]

    len(df_trn), len(df_val), len(df_tst)
```

```
(19570, 99, 7567)
```

FIg. 8 : Code describes the removal of pairs where the code is smaller than the comment.

Step 5:

Next, we remove any examples that have the special <code> tag since these also tend to contain extra information that the model doesn't have a good hope of generating. We remove the JavaDoc parts of the comments other than the description.. The other pieces of information can usually be auto generated or may require external knowledge to document them.

```
# collapse
def has_code(cmt: str) -> bool:
    '''
    Determinine if the given comment contains the HTML <code> tag

    :param cmt: the comment to check whether it contains the HTML <code> tag
    :returns: whether or not the given comment contains the HTML <code> tag
    '''
    if '<code>' in cmt: return True
    else: return False

df_trn = df_trn[~df_trn['cmt'].apply(lambda x: has_code(x))]
df_val = df_val[~df_val['cmt'].apply(lambda x: has_code(x))]
df_tst = df_tst[~df_tst['cmt'].apply(lambda x: has_code(x))]

len(df_trn), len(df_val), len(df_tst)
```

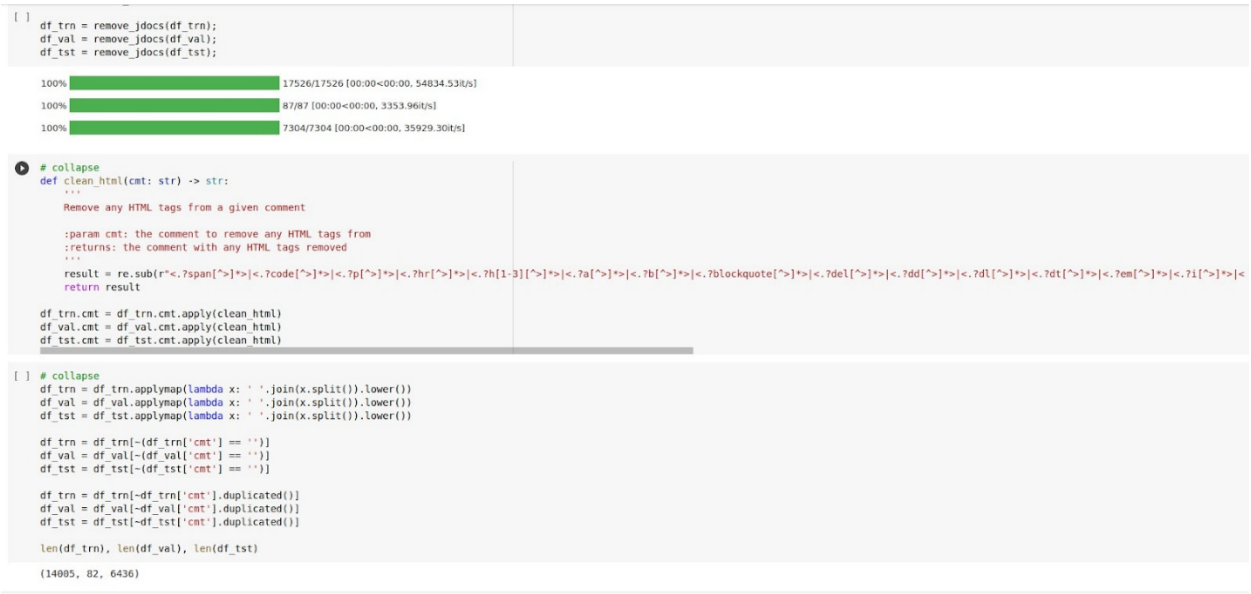
```
(17526, 87, 7304)
```

```
[ ] # collapse
def remove_jdocs(df: pd.DataFrame) -> pd.DataFrame:
    '''
    Remove the JavaDocs leaving only the description of the comment

    :param df: the pandas dataframe to remove the JavaDocs from
    :returns: a new pandas dataframe with the JavaDocs removed
    '''
    methods = []
    comments = []
    for i, row in tqdm(list(df.iterrows())):
        comment = row["cmt"]
        # Remove {} text in comments from https://stackoverflow.com/questions/14596884/remove-text-between-and-in-python/14598135
        comment = re.sub("([\{\[]).*?([\)\}])", '', comment)


        cleaned = []
        for line in comment.split('\n'):
            if "@" in line: break
            cleaned.append(line)
        comments.append('\n'.join(cleaned))
        methods.append(row["mthd"])
    new_df = pd.DataFrame(zip(methods, comments), columns = ["mthd", "cmt"])
```

Fig. 9 : Code describes the removal of <code> tag and JavaDoc part of the comments.



```
df_trn = remove_jdocs(df_trn);
df_val = remove_jdocs(df_val);
df_tst = remove_jdocs(df_tst);
```

```
100% 17526/17526 [00:00<00:00, 54834.53it/s]
100% 87/87 [00:00<00:00, 3353.96it/s]
100% 7304/7304 [00:00<00:00, 35929.30it/s]
```

```
# collapse
def clean_html(cmt: str) -> str:
    '''
    Remove any HTML tags from a given comment

    :param cmt: the comment to remove any HTML tags from
    :returns: the comment with any HTML tags removed
    '''
    result = re.sub(r"<.?span[^>]*>|<.?code[^>]*>|<.?p[^>]*>|<.?hr[^>]*>|<.?h[1-3][^>]*>|<.?a[^>]*>|<.?b[^>]*>|<.?blockquote[^>]*>|<.?del[^>]*>|<.?dd[^>]*>|<.?dl[^>]*>|<.?dt[^>]*>|<.?em[^>]*>|<.?i[^>]*>|<
    return result

df_trn.cmt = df_trn.cmt.apply(clean_html)
df_val.cmt = df_val.cmt.apply(clean_html)
df_tst.cmt = df_tst.cmt.apply(clean_html)
```

```
# collapse
df_trn = df_trn.applymap(lambda x: ' '.join(x.split()).lower())
df_val = df_val.applymap(lambda x: ' '.join(x.split()).lower())
df_tst = df_tst.applymap(lambda x: ' '.join(x.split()).lower())

df_trn = df_trn[~(df_trn['cmt'] == '')]
df_val = df_val[~(df_val['cmt'] == '')]
df_tst = df_tst[~(df_tst['cmt'] == '')]

df_trn = df_trn[~df_trn['cmt'].duplicated()]
df_val = df_val[~df_val['cmt'].duplicated()]
df_tst = df_tst[~df_tst['cmt'].duplicated()]

len(df_trn), len(df_val), len(df_tst)
```

```
(14005, 82, 6436)
```

Fig. 10 : Code describes removal of HTML tags from the comment.

```
df_trn
```

	mthd	cmt
1	public void setfactor(int factor) { this.facto...	set contrast adjusting factor, [-127, 127].
2	@override public imagesource apply(imagesource...	expects a height mat as input
3	public <l extends listener> void popevent(even...	pops the top event off the current event stack...
4	public apiresponse<exporthistoryresponse> gete...	get export history get the history of export r...
5	public apiresponse<string> getexportresultwith...	get export result retrieve result of the expor...
...	...	...
17521	path path = filesystems.getdefault().getpath(t...	set up
17522	adv.adddelayeddata(gt, c); a.add(adv);	remember data and chunk
17523	} catch (final exception e) { throw new mojoex...	checkstyle.off: illegalcatch checkstyle.on: il...
17524	throw new victimsexception("a custom jdbc dri...	custom drivers require custom urls
17525	map<string, string> searchmap = new hashmap<st...	return wrappedsubmit(new findbyparam("~name", ...

14005 rows × 2 columns

```
# collapse
import numpy as np

from collections import Counter
from statistics import mean, median, stdev
from transformers import AutoTokenizer

def get_counter(df: pd.DataFrame, tokenizer: AutoTokenizer, col: str) -> Counter:
    '''
    Get the counts for each token in a given pandas dataframe column

    :param df: the pandas dataframe to get the counts of tokens from
    :param tokenizer: the tokenizer to use for tokenizing the rows in the pandas
    dataframe
    :param col: the column to grab rows from when tokenizing
    :returns: the counts of each token in the given pandas dataframe
    column
    '''
```

Fig. 11 : Code describes the displaying of the modified training data.

Step 6:
To accomplish the training data, we use the CodeXGLUE repository. This repository is similar to the NLP equivalent GLUE benchmarks.

```
    '''
    toks = []
    for i, row in df.iterrows():
        toks.extend(tokenizer.tokenize(row[col]))

    cnt = Counter()
    for tok in toks:
        cnt[tok] += 1
    return cnt

tokenizer = AutoTokenizer.from_pretrained('microsoft/codebert-base')
mthd_cnt = get_counter(df_trn, tokenizer, 'mthd')
cmt_cnt = get_counter(df_trn, tokenizer, 'cmt')
mthd_lens = df_trn.mthd.apply(lambda x: len(tokenizer.tokenize(x))).values
cmt_lens = df_trn.cmt.apply(lambda x: len(tokenizer.tokenize(x))).values
max_mthd_len = int(np.quantile(mthd_lens, 0.95))
max_cmt_len = int(np.quantile(cmt_lens, 0.95))
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (2015 > 512). Running this sequence through the model will result in indexing errors
```

```
# collapse
import json

df_trn['code_tokens'] = df_trn.mthd.apply(lambda x: x.split())
df_trn['docstring_tokens'] = df_trn.cmt.apply(lambda x: x.split())
with open('/content/drive/MyDrive/RP/java/train.jsonl','w') as f:
    for _, row in df_trn.iterrows():
        f.write(json.dumps(row.to_dict()) + '\n')

df_val['code_tokens'] = df_val.mthd.apply(lambda x: x.split())
df_val['docstring_tokens'] = df_val.cmt.apply(lambda x: x.split())
with open('/content/drive/MyDrive/RP/java/valid.jsonl','w') as f:
    for _, row in df_val.iterrows():
        f.write(json.dumps(row.to_dict()) + '\n')

df_tst['code_tokens'] = df_tst.mthd.apply(lambda x: x.split())
df_tst['docstring_tokens'] = df_tst.cmt.apply(lambda x: x.split())
with open('/content/drive/MyDrive/RP/java/test.jsonl','w') as f:
    for _, row in df_tst.iterrows():
        f.write(json.dumps(row.to_dict()) + '\n')
```

```
lang = 'java' # programming language
lr = 5e-5
```

Fig. 12 : Code describes creating a json file for training, testing and valid data.

```
lang = 'java' # programming language
lr = 5e-5
batch_size = 8 # change depending on the GPU Colab gives you
beam_size = 10
source_length = 256
target_length = max_cmt_len
data_dir = '/content/drive/MyDrive/RP'
output_dir = f'/content/drive/MyDrive/RP/model/{lang}'
train_file = f'{data_dir}/{lang}/train.jsonl'
dev_file = f'{data_dir}/{lang}/valid.jsonl'
epochs = 5
pretrained_model = 'microsoft/codebert-base'
```

```
! python /content/drive/MyDrive/RP/CodeXGLUE-main/Code-Text/code-to-text/code/run.py \
    --do_train \
    --do_eval \
    --do_lower_case \
    --model_type roberta \
    --model_name_or_path {pretrained_model} \
    --train_filename {train_file} \
    --dev_filename {dev_file} \
    --output_dir {output_dir} \
    --max_source_length {source_length} \
    --max_target_length {target_length} \
    --beam_size {beam_size} \
    --train_batch_size {batch_size} \
    --eval_batch_size {batch_size} \
    --learning_rate {lr} \
    --num_train_epochs {epochs}
```

```
2023-04-26 13:26:45.685142: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
04/26/2023 13:26:47 - INFO - __main__ -   Namespace(model_type='roberta', model_name_or_path='microsoft/codebert-base', output_dir='/content/drive/MyDrive/RP/model/java', load_model_path=None, train_filenam
04/26/2023 13:26:47 - WARNING - __main__ -   Process rank: -1, device: cuda, n_gpu: 1, distributed training: False
Downloading pytorch_model.bin: 100% 499M/499M [00:01<00:00, 263MB/s]
04/26/2023 13:26:59 - INFO - __main__ -   *** Example ***
04/26/2023 13:26:59 - INFO - __main__ -   idx: 0
04/26/2023 13:26:59 - INFO - __main__ -   source_tokens: ['<s>', 'public', '_void', '_set', 'factor', '(', 'int', '_factor', ')', '_{', '_this', '.', 'factor', '_=', '_factor', '_=', '_math', '.', 'max', '
04/26/2023 13:26:59 - INFO - __main__ -   source_ids: 0 15110 13842 278 31192 1640 2544 3724 43 25522 42 4 31192 5457 3724 5457 10638 4 29459 48614 24174 6 10638 4 4691 1640 24174 6 3724 48749 114 36 31192
04/26/2023 13:26:59 - INFO - __main__ -   source_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
04/26/2023 13:26:59 - INFO - __main__ -   target_tokens: ['<s>', 'set', '_contrast', '_adjusting', '_factor', ',', '_[-', '127', ',', '_127', '].', '</s>']
04/26/2023 13:26:59 - INFO - __main__ -   target_ids: 0 8738 5709 18624 3724 6 48906 24174 6 20598 8174 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
04/26/2023 13:26:59 - INFO - __main__ -   target_mask: 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
04/26/2023 13:26:59 - INFO - __main__ -   *** Example ***
04/26/2023 13:26:59 - INFO - __main__ -   idx: 1
```

FIg. 13 : Training the data.

Step 7:
We trained our model in the last step, we can now see how well it turned out by evaluating it.

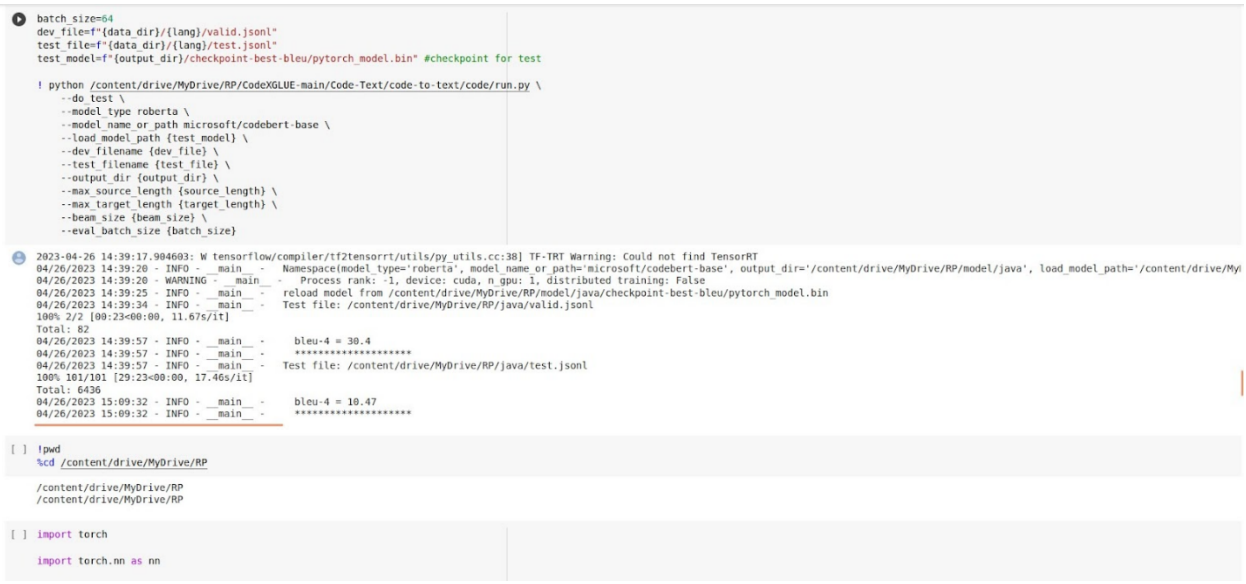


FIg. 14 : Code describes changing the path to access the required files.

```
import torch.nn as nn

from model import Seq2Seq
from transformers import RobertaConfig, RobertaModel

config = RobertaConfig.from_pretrained(pretrained_model)
encoder = RobertaModel.from_pretrained(pretrained_model, config = config)
decoder_layer = nn.TransformerDecoderLayer(d_model=config.hidden_size, nhead=config.num_attention_heads)
decoder = nn.TransformerDecoder(decoder_layer, num_layers=6)
model = Seq2Seq(encoder = encoder,decoder = decoder,config=config,
                beam_size=beam_size,max_length=target_length,
                sos_id=tokenizer.cls_token_id,eos_id=tokenizer.sep_token_id)
model.load_state_dict(torch.load(f"{output_dir}/checkpoint-last/pytorch_model.bin"))
model.to('cuda')
```

```
Seq2Seq(
  (encoder): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
```

Fig. 15 : Code describes that encoder used is CodeBERT model and decoder used is Transformers

Results

Checking the results in the test data along the validation data.

Fig. 16 : Comparing the Original and Generated comment by the model.

```python
def get_preds_losses(df: pd.DataFrame):
    ps = []
    losses = []
    for idx, row in tqdm(df.iterrows(), total=len(df)):
        examples = [
            Example(idx, source = row.mthd, target = row.cmt)
        ]
        eval_features = convert_examples_to_features(
            examples, tokenizer, args, stage='test'
        )
        source_ids = torch.tensor([f.source_ids for f in eval_features], dtype = torch.long).to('cuda')
        source_mask = torch.tensor([f.source_mask for f in eval_features], dtype = torch.long).to('cuda')
        target_ids = torch.tensor([f.target_ids for f in eval_features], dtype = torch.long).to('cuda')
        target_mask = torch.tensor([f.target_mask for f in eval_features], dtype = torch.long).to('cuda')

        with torch.no_grad():
            _, loss, _ = model(
                source_ids = source_ids, source_mask = source_mask,
                target_ids = target_ids, target_mask = target_mask
            )
            preds = model(source_ids = source_ids, source_mask = source_mask)
            for pred in preds:
                t = pred[0].cpu().numpy()
                t = list(t)
                if 0 in t:
                    t = t[:t.index(0)]
                text = tokenizer.decode(t,clean_up_tokenization_spaces=False)
                ps.append(text)
                losses.append(loss.item())

    return ps, losses
```

```python
df_head = df_val.copy()
ps, losses = get_preds_losses(df_head)
df_head['pred'] = ps
df_head['loss'] = losses
df_sorted_losses = df_head.sort_values('loss', ascending = False)

for _, row in df_sorted_losses.head(10).iterrows():
    print('Code:', row.mthd)
    print('Original Comment:', row.cmt)
    print('Generated Comment:', row.pred)
```

Fig. 17 : Code describes calculating the predicted loss for the generated comment wrt to original comment.

```
df_head = df_val.copy()
ps, losses = get_preds_losses(df_head)
df_head['pred'] = ps
df_head['loss'] = losses
df_sorted_losses = df_head.sort_values('loss', ascending = False)

for _, row in df_sorted_losses.head(10).iterrows():
    print('Code:', row.mthd)
    print('Original Comment:', row.cmt)
    print('Generated Comment:', row.pred)
    print(row.loss)
    print('='*40)
```

```
100% 82/82 [00:26<00:00, 4.72it/s]
Code: @pure public rectangle2d getcellbounds(int row, int column) { final double cellwidth = getcellwidth(); final double cellheight = getcellheight(); final double x = this.bounds.getminx() + cellwidth *
Original Comment: replies the bounds covered by a cell at the specified location.
Generated Comment: replies the bounds of the box.
27.512229919433594
========================================
Code: public static point3i convert(tuple3d<?> tuple) { if (tuple instanceof point3i) { return (point3i) tuple; } return new point3i(tuple.getx(), tuple.gety(), tuple.getz()); }
Original Comment: convert the given tuple to a real point2i. if the given tuple is already a point2i, it is replied.
Generated Comment: convert the given tuple to a real point3i.
27.273990631103516
========================================
Code: @pure public doubleproperty depthproperty() { if (this.depth == null) { this.depth = new readonlydoublewrapper(this, mathfxattributenames.depth); this.depth.bind(bindings.subtract(maxzproperty(), min
Original Comment: replies the property that is the depth of the box.
Generated Comment: replies the property.
27.073217391967773
========================================
Code: @pure @suppresswarnings("checkstyle:cyclomaticcomplexity") public pt getstartingpointfor(int index) { if ((index < 1) || (this.segmentlist.size() <= 1)) { if (this.startingpoint != null) { return thi
Original Comment: replies the starting point for the segment at the given index.
Generated Comment: replies the start point at the given segment.
26.920101165771484
========================================
Code: public static vector3ifx convert(tuple3d<?> tuple) { if (tuple instanceof vector3ifx) { return (vector3ifx) tuple; } return new vector3ifx(tuple.getx(), tuple.gety(), tuple.getz()); }
Original Comment: convert the given tuple to a real vector3ifx. if the given tuple is already a vector3ifx, it is replied.
Generated Comment: convert the given tuple to a real vector3ifx. if the given tuple is already a vector3ifx, it is replied.
26.859310150146484
========================================
Code: public static vector2d convert(tuple2d<?> tuple) { if (tuple instanceof vector2d) { return (vector2d) tuple; } return new vector2d(tuple.getx(), tuple.gety()); }
Original Comment: convert the given tuple to a real vector2d. if the given tuple is already a vector2d, it is replied.
Generated Comment: convert the given tuple to a real vector2d.
26.83241844177246
========================================
Code: @pure public static int getcohensutherlandcode(int px, int py, int rxmin, int rymin, int rxmax, int rymax) { assert rxmin <= rxmax : assertmessages.lowerequalparameters(2, rxmin, 4, rxmax); assert ry
```

Fig. 18 : Comparing the original and generated comment wrt to the loss.

```
27.512229919433594
========================================
Code: public static point3i convert(tuple3d<?> tuple) { if (tuple instanceof point3i) { return (point3i) tuple; } return new point3i(tuple.getx(), tuple.gety(), tuple.getz()); }
Original Comment: convert the given tuple to a real point2i. if the given tuple is already a point2i, it is replied.
Generated Comment: convert the given tuple to a real point3i.
27.273990631103516
========================================
Code: @pure public doubleproperty depthproperty() { if (this.depth == null) { this.depth = new readonlydoublewrapper(this, mathfxattributenames.depth); this.depth.bind(bindings.subtract(maxzproperty(), min
Original Comment: replies the property that is the depth of the box.
Generated Comment: replies the property.
27.073217391967773
========================================
Code: @pure @suppresswarnings("checkstyle:cyclomaticcomplexity") public pt getstartingpointfor(int index) { if ((index < 1) || (this.segmentlist.size() <= 1)) { if (this.startingpoint != null) { return thi
Original Comment: replies the starting point for the segment at the given index.
Generated Comment: replies the start point at the given segment.
26.920101165771484
========================================
Code: public static vector3ifx convert(tuple3d<?> tuple) { if (tuple instanceof vector3ifx) { return (vector3ifx) tuple; } return new vector3ifx(tuple.getx(), tuple.gety(), tuple.getz()); }
Original Comment: convert the given tuple to a real vector3ifx. if the given tuple is already a vector3ifx, it is replied.
Generated Comment: convert the given tuple to a real vector3ifx. if the given tuple is already a vector3ifx, it is replied.
26.859310150146484
========================================
Code: public static vector2d convert(tuple2d<?> tuple) { if (tuple instanceof vector2d) { return (vector2d) tuple; } return new vector2d(tuple.getx(), tuple.gety()); }
Original Comment: convert the given tuple to a real vector2d. if the given tuple is already a vector2d, it is replied.
Generated Comment: convert the given tuple to a real vector2d.
26.83241844177246
========================================
Code: @pure public static int getcohensutherlandcode(int px, int py, int rxmin, int rymin, int rxmax, int rymax) { assert rxmin <= rxmax : assertmessages.lowerequalparameters(2, rxmin, 4, rxmax); assert ry
Original Comment: compute the zone where the point is against the given rectangle according to the cohen-sutherland algorithm.
Generated Comment: compute the given cohenation.
26.055528604074707
========================================
Code: @pure public double distancesquaredline(point3d point) { return distancesquaredlinepoint( getx1(), gety1(), getz1(), getx2(), gety2(), getz2(), point.getx(), point.gety(), point.getz()); }
Original Comment: replies the squared distance between the line of this segment and the given point.
Generated Comment: replies the squared distance between the given point.
25.379169464111328
========================================
Code: public static void setpreferredroadtype(roadtype type) { final preferences prefs = preferences.usernodeforpackage(roadnetworkconstants.class); if (prefs != null) { if (type == null) { prefs.remove("r
Original Comment: set the preferred type of road segment.
Generated Comment: set the preferred type of road segment.
25.26555633544922
========================================
Code: @suppresswarnings("checkstyle:magicnumber") public static void copy(inputstream in, int insize, fileoutputstream out) throws ioexception { assert in != null; assert out != null; try (readablebytechan
Original Comment: copy the first file into the second file. the content of the second file will be lost. this copy function allows to do a copy between two different partitions.
Generated Comment: copy a file.
25.095462799072266
========================================
```

Fig. 19 : Comparing the original and generated comment wrt to the loss.

Future Work

Firstly, we have to train the model on the entire dataset. For now, we have used only 0.5% of the dataset. To enable code comments to be generated automatically, we have improved the RoBERT-based model that was trained on code data. Although we have primarily discussed Java, the same principles can be used to learn any other programming language that piques your interest. Additionally, we will use various models to generate code in various languages, including Python, JavaScript, Ruby, Go, and PHP. We'll also build a VSCode add-on that uses this code commenting project, enabling you to create comments for code snippets you highlight.

Conclusion

We studied deep learning concepts like RNN, LSTM, TRANSFORMERS,models of transformers like BERT and CodeBERT to understand their working and how we can leverage these concepts for process languages. Our aim was to understand the methodology and use it to do code commenting in programming languages. We implemented CodeBERT using code and docstrings rather than only natural language. This made it possible for the CodeBERT model to acquire a practical representation of code that it could apply to different tasks. We used used the CodeXGLUE repository to test our model. We computed the loss, compared the original comment to the model-generated comment, and used the results to test our model. As we have used only 0.5% of the original dataset, still we got good results. The loss for every comment ranged between 25-28%, which is decent.

References

1. Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D. and Zhou, M., 2020. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*.

0. Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

0. Clark, K., Luong, M.T., Le, Q.V. and Manning, C.D., 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

0. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, *30*.

5. Uszkoreit, J., 2019. Transformer: a novel neural network architecture for language understanding, August 2017.