Deep Learning Final Project Report

Problem Statement:

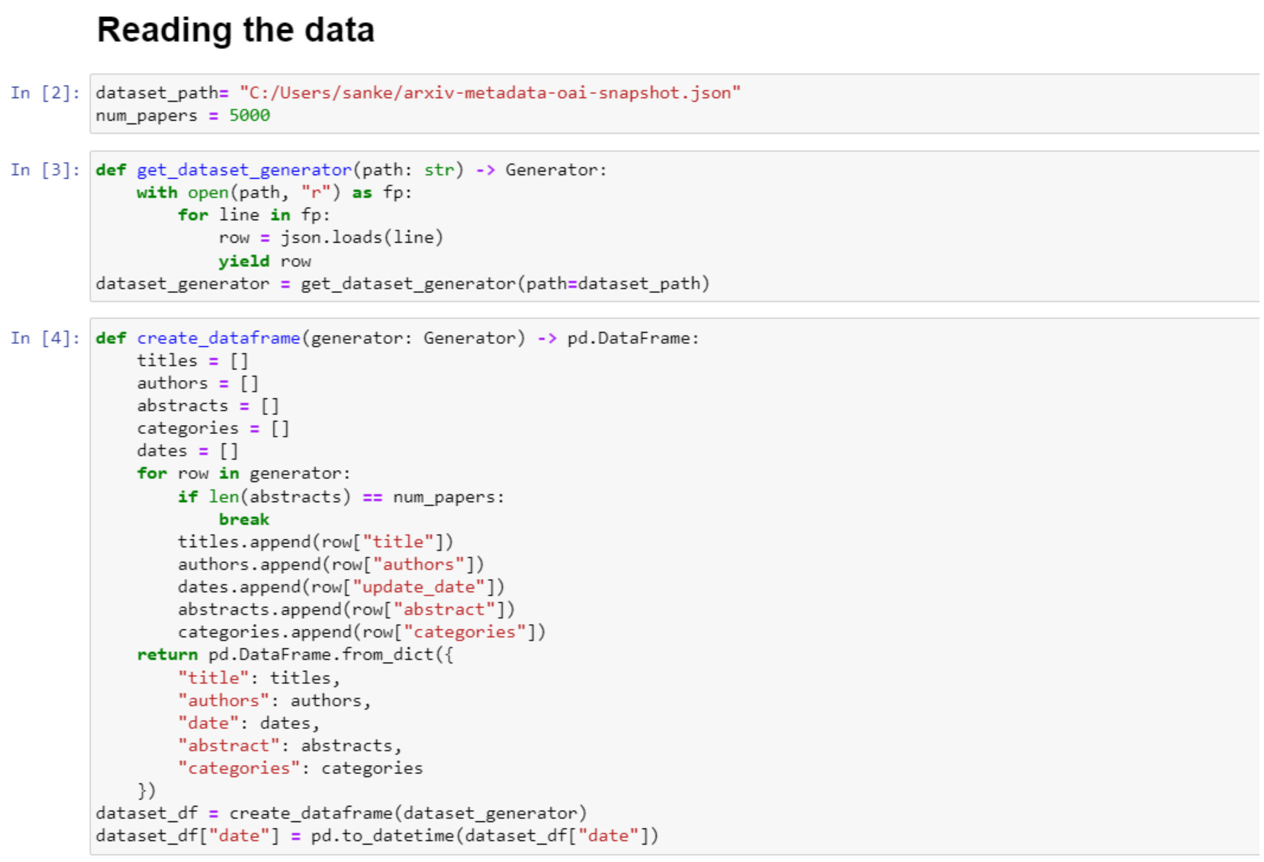
Generating research paper titles based on past titles and research progress. Creating a model to generate research paper titles by embedding various deep learning techniques, natural language processing, and different language models.

Dataset Description:

The dataset is from ArXiv, which is an open access platform that collects data about scientific research papers related to physics, computer science, and other subfields of science.The dataset used in this project is of JSON type and consists of data related to 1.7 million articles .

Data Preprocessing:

To keep the process simple, the dataset which is in JSON formatis transformed into pandas data frame. And only 5000 records are used for the initial model building.

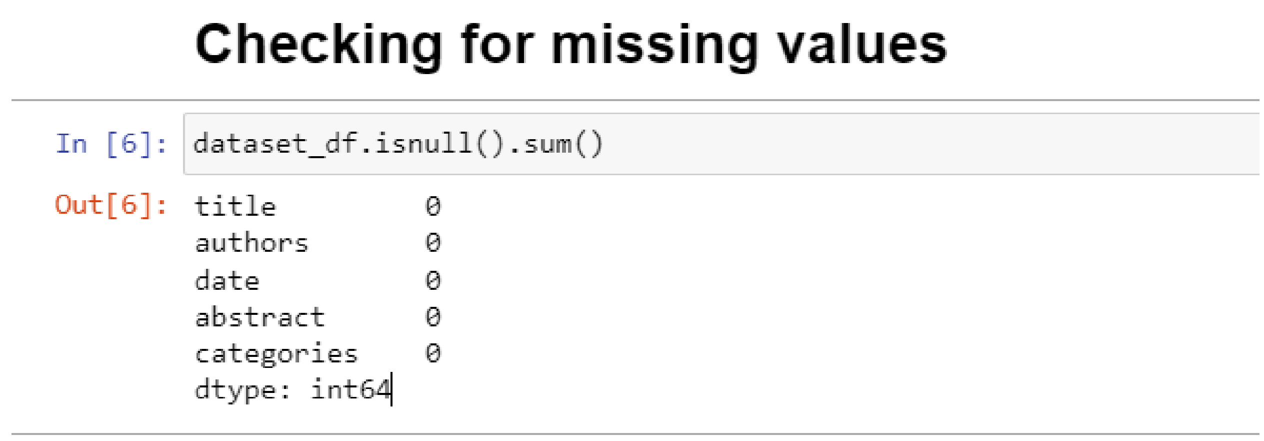


Reading the data

```
In [2]: dataset_path= "C:/Users/sanke/arxiv-metadata-oai-snapshot.json"
        num_papers = 5000
```

```
In [3]: def get_dataset_generator(path: str) -> Generator:
            with open(path, "r") as fp:
                for line in fp:
                    row = json.loads(line)
                    yield row
        dataset_generator = get_dataset_generator(path=dataset_path)
```

```
In [4]: def create_dataframe(generator: Generator) -> pd.DataFrame:
            titles = []
            authors = []
            abstracts = []
            categories = []
            dates = []
            for row in generator:
                if len(abstracts) == num_papers:
                    break
                titles.append(row["title"])
                authors.append(row["authors"])
                dates.append(row["update_date"])
                abstracts.append(row["abstract"])
                categories.append(row["categories"])
            return pd.DataFrame.from_dict({
                "title": titles,
                "authors": authors,
                "date": dates,
                "abstract": abstracts,
                "categories": categories
            })
        dataset_df = create_dataframe(dataset_generator)
        dataset_df["date"] = pd.to_datetime(dataset_df["date"])
```



Checking for missing values

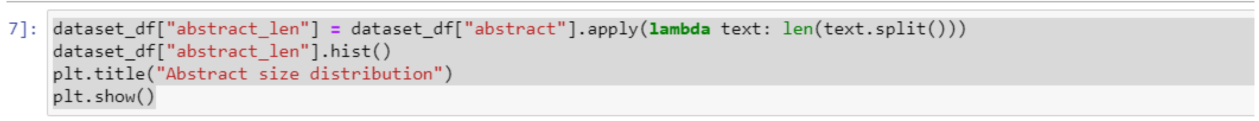
```
In [6]: dataset_df.isnull().sum()

Out[6]: title         0
        authors       0
        date          0
        abstract      0
        categories    0
        dtype: int64
```

No missing values

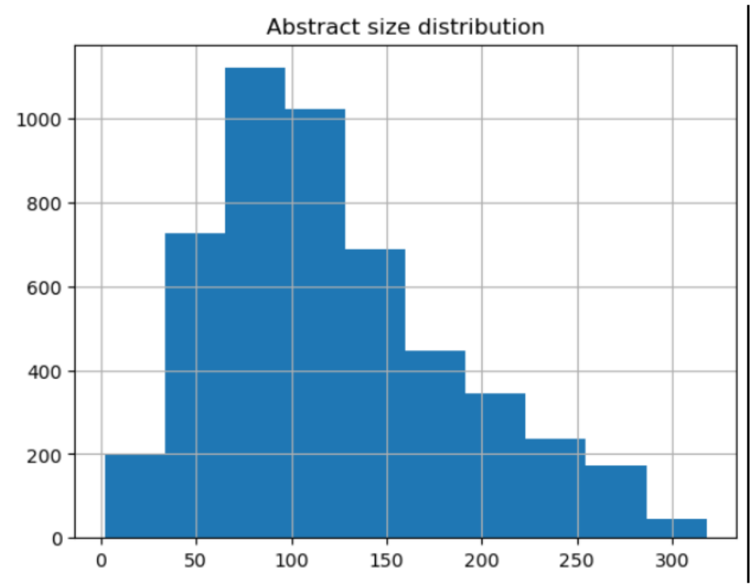
Null values were not found in the dataset, which removes the process of replacing or filling the null values.

Visualization:

Abstract size



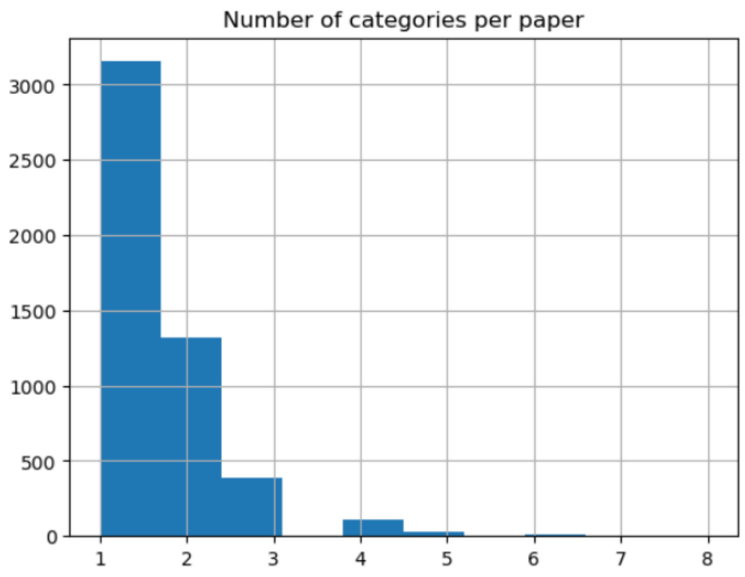
```
7]: dataset_df["abstract_len"] = dataset_df["abstract"].apply(lambda text: len(text.split()))
    dataset_df["abstract_len"].hist()
    plt.title("Abstract size distribution")
    plt.show()
```



The above plot depicts the distribution of abstract size in the research article. We can infer that there are more than 1000 words in the abstract in many articles. We can also infer that the variance in this histogram is wider, by which we can say that there is great diversity in abstract length.

Number of categories per paper

```
In [8]: dataset_df["categories"] = dataset_df["categories"].apply(lambda text: tuple(text.split()))
        dataset_df["num_categories"] = dataset_df["categories"].apply(lambda cats: len(cats))
        dataset_df["num_categories"].hist()
        plt.title("Number of categories per paper")
        plt.show()
```



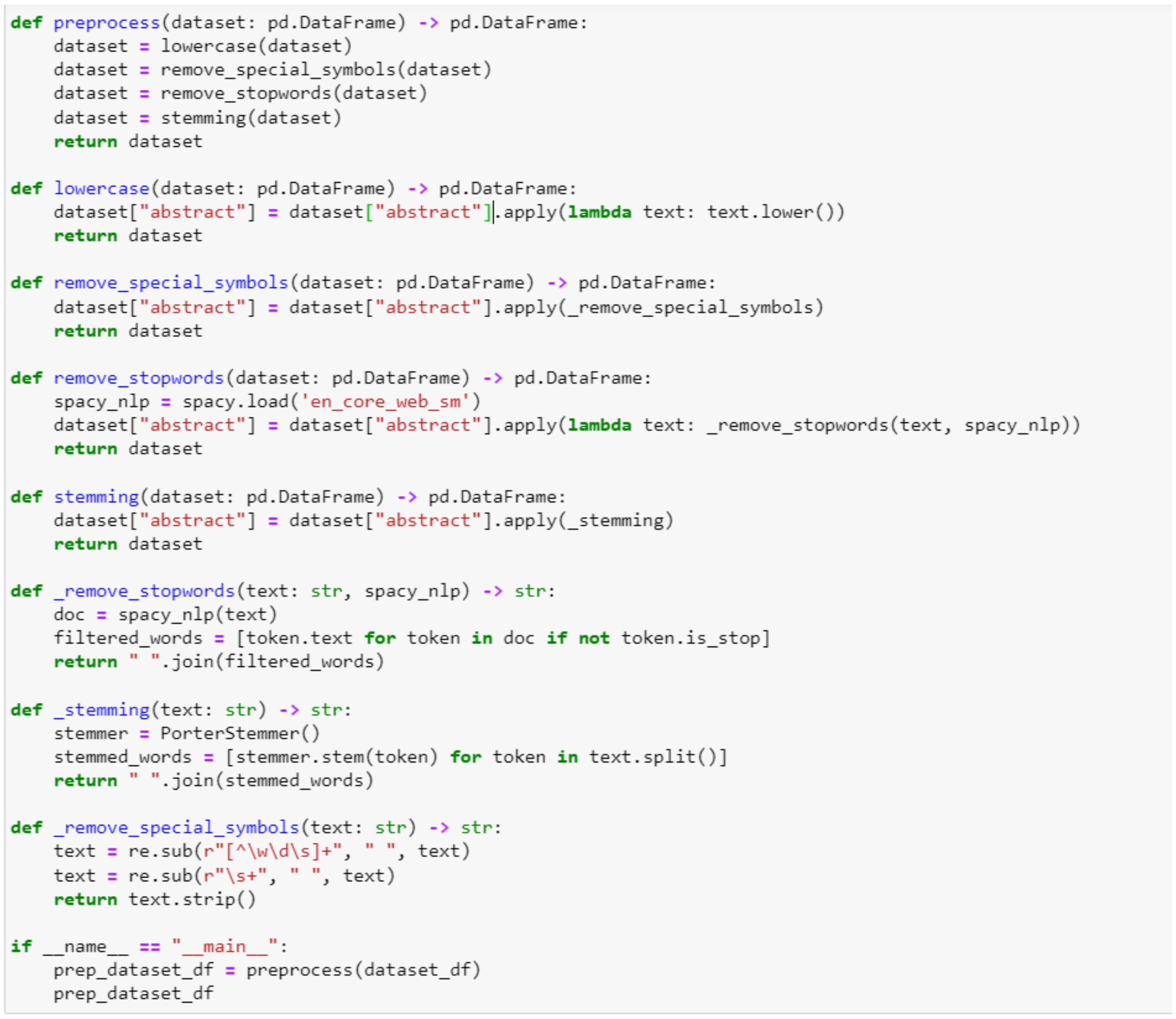
There are very few articles with more than one category.

Number of unique papers

```
[ ]: print(f"Num. papers: {len(dataset_df)}")
     dataset_df = dataset_df[~dataset_df[["title", "authors"]].duplicated()]
     print(f"Num. unique papers: {len(dataset_df)}")
```

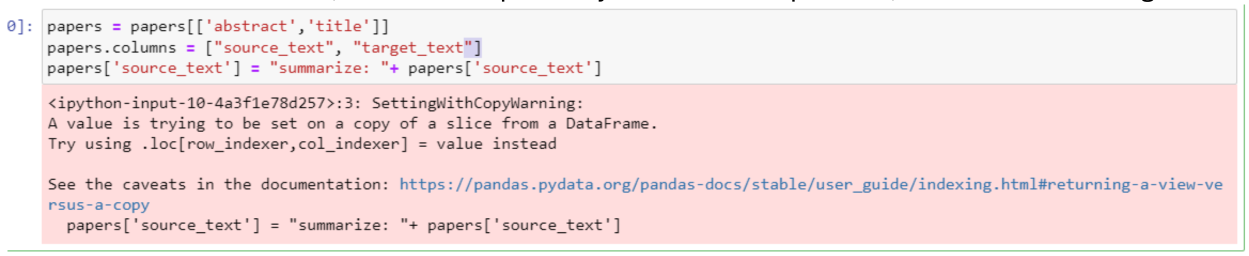
```
Num. papers: 5000
Num. unique papers: 4998
```

Two papers do not have a unique title and author name.



```python
def preprocess(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset = lowercase(dataset)
    dataset = remove_special_symbols(dataset)
    dataset = remove_stopwords(dataset)
    dataset = stemming(dataset)
    return dataset

def lowercase(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset["abstract"] = dataset["abstract"].apply(lambda text: text.lower())
    return dataset

def remove_special_symbols(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset["abstract"] = dataset["abstract"].apply(_remove_special_symbols)
    return dataset

def remove_stopwords(dataset: pd.DataFrame) -> pd.DataFrame:
    spacy_nlp = spacy.load('en_core_web_sm')
    dataset["abstract"] = dataset["abstract"].apply(lambda text: _remove_stopwords(text, spacy_nlp))
    return dataset

def stemming(dataset: pd.DataFrame) -> pd.DataFrame:
    dataset["abstract"] = dataset["abstract"].apply(_stemming)
    return dataset

def _remove_stopwords(text: str, spacy_nlp) -> str:
    doc = spacy_nlp(text)
    filtered_words = [token.text for token in doc if not token.is_stop]
    return " ".join(filtered_words)

def _stemming(text: str) -> str:
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(token) for token in text.split()]
    return " ".join(stemmed_words)

def _remove_special_symbols(text: str) -> str:
    text = re.sub(r"[^\w\d\s]+", " ", text)
    text = re.sub(r"\s+", " ", text)
    return text.strip()

if __name__ == "__main__":
    prep_dataset_df = preprocess(dataset_df)
    prep_dataset_df
```

The above code acts as a pipeline in cleaning and preparing the text data for training the model. It applies lowercase conversion, removal of special symbols and stop words, and word stemming.

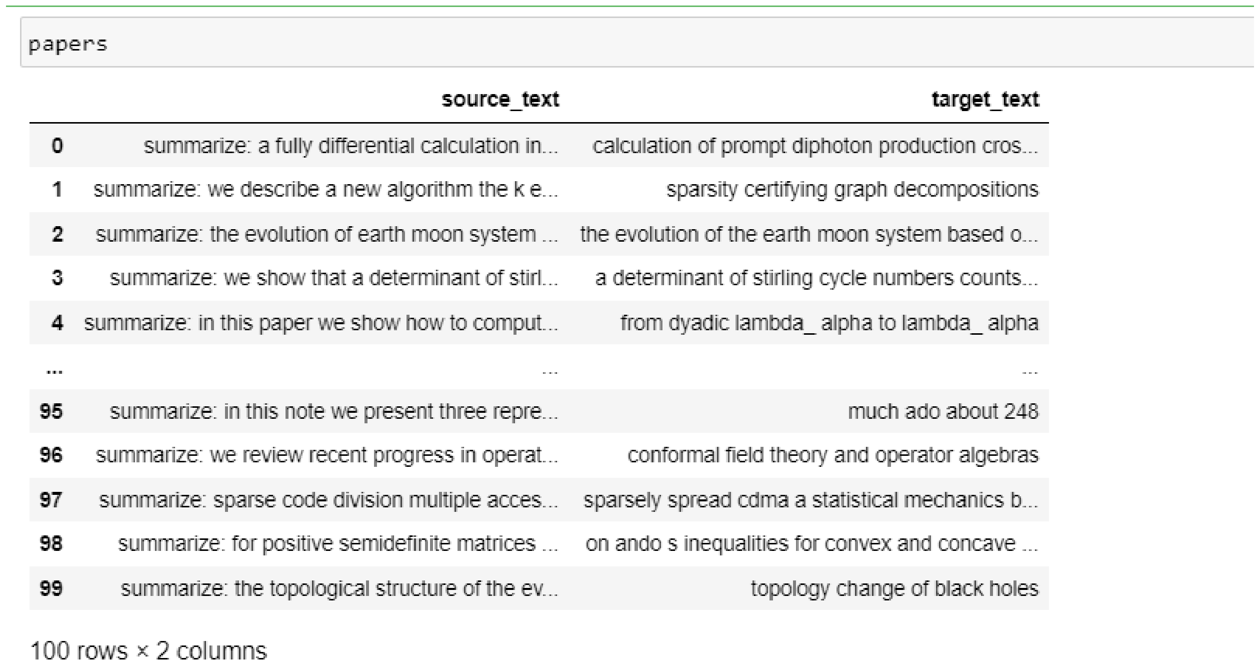


```
0]: papers = papers[['abstract','title']]
    papers.columns = ["source_text", "target_text"]
    papers['source_text'] = "summarize: "+ papers['source_text']
```

```
<ipython-input-10-4a3f1e78d257>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  papers['source_text'] = "summarize: "+ papers['source_text']
```

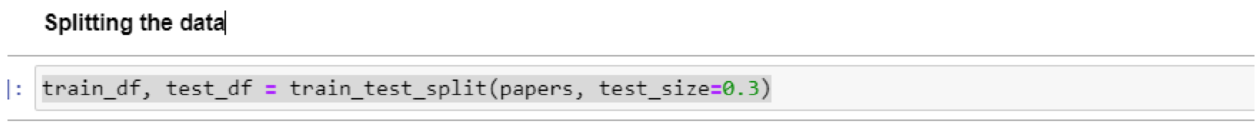
In the above snippet, we are selecting only the abstract and title column from the dataset. And it is renamed as source_text and target_text respectively.

The word “Summarize” is also added to the source_text.


```
papers
```

	source_text	target_text
0	summarize: a fully differential calculation in...	calculation of prompt diphoton production cros...
1	summarize: we describe a new algorithm the k e...	sparsity certifying graph decompositions
2	summarize: the evolution of earth moon system ...	the evolution of the earth moon system based o...
3	summarize: we show that a determinant of stirl...	a determinant of stirling cycle numbers counts...
4	summarize: in this paper we show how to comput...	from dyadic lambda_ alpha to lambda_ alpha
...	...	...
95	summarize: in this note we present three repre...	much ado about 248
96	summarize: we review recent progress in operat...	conformal field theory and operator algebras
97	summarize: sparse code division multiple acces...	sparsely spread cdma a statistical mechanics b...
98	summarize: for positive semidefinite matrices ...	on ando s inequalities for convex and concave ...
99	summarize: the topological structure of the ev...	topology change of black holes

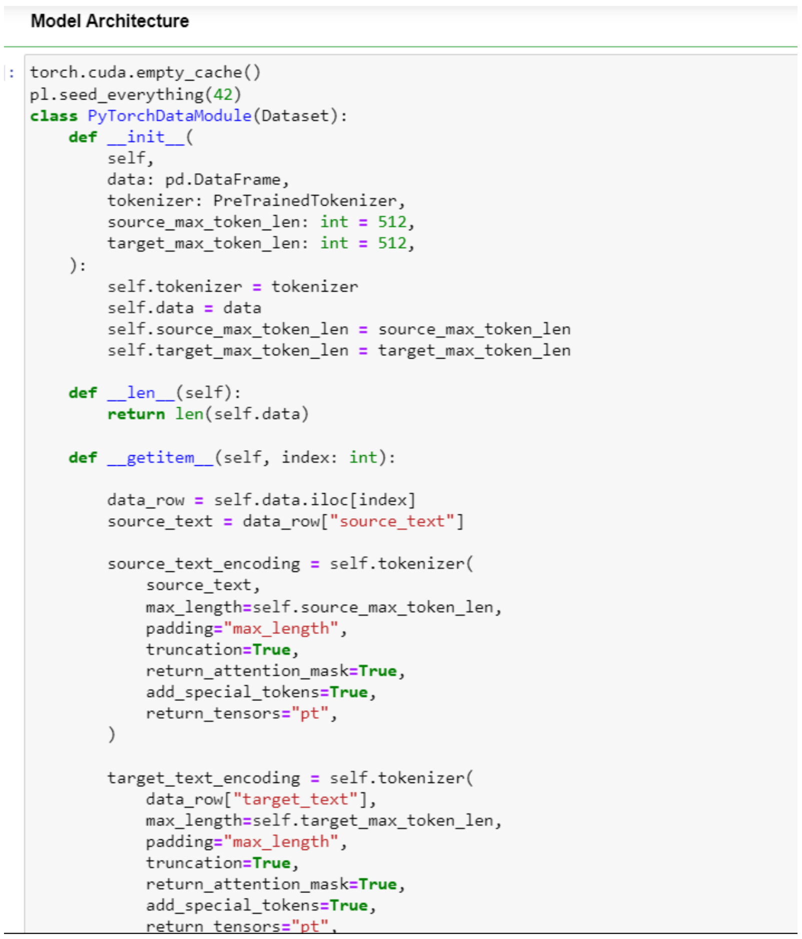
100 rows × 2 columns


Splitting the data

```
train_df, test_df = train_test_split(papers, test_size=0.3)
```

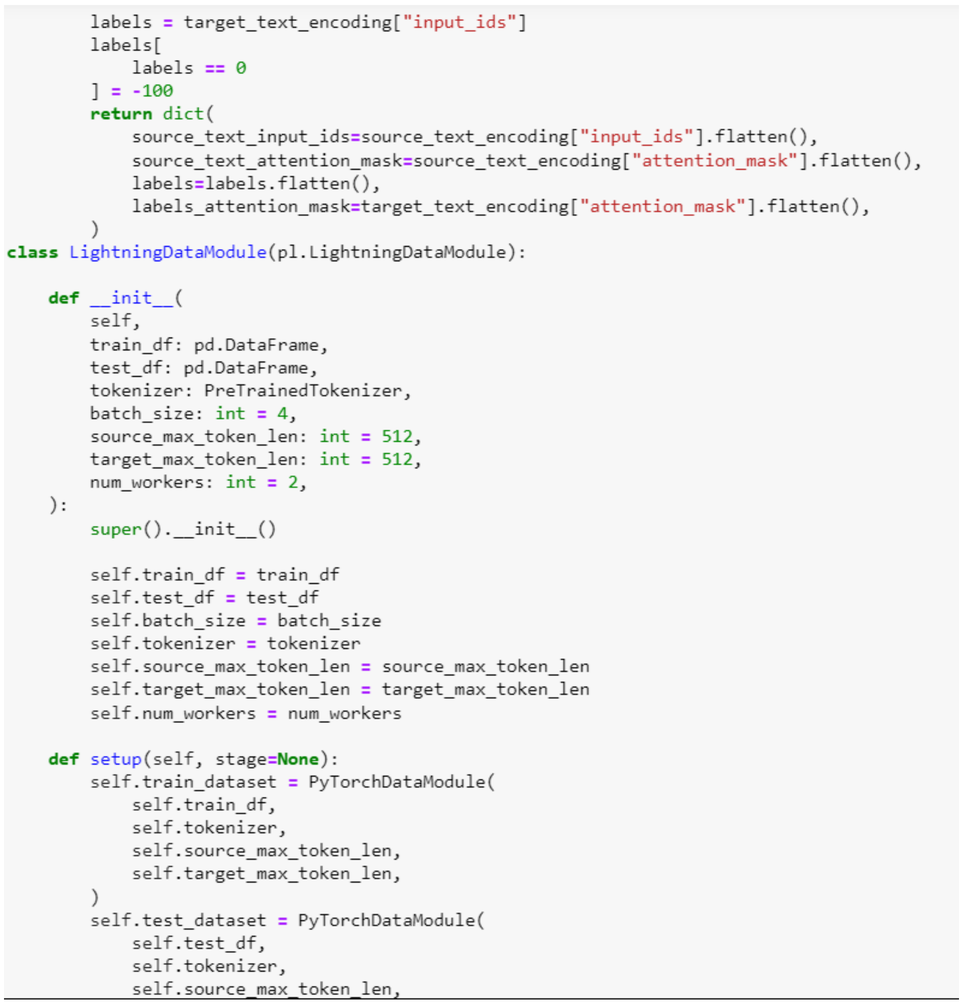
The dataset is split for proceeding with training and testing.

Model Building:

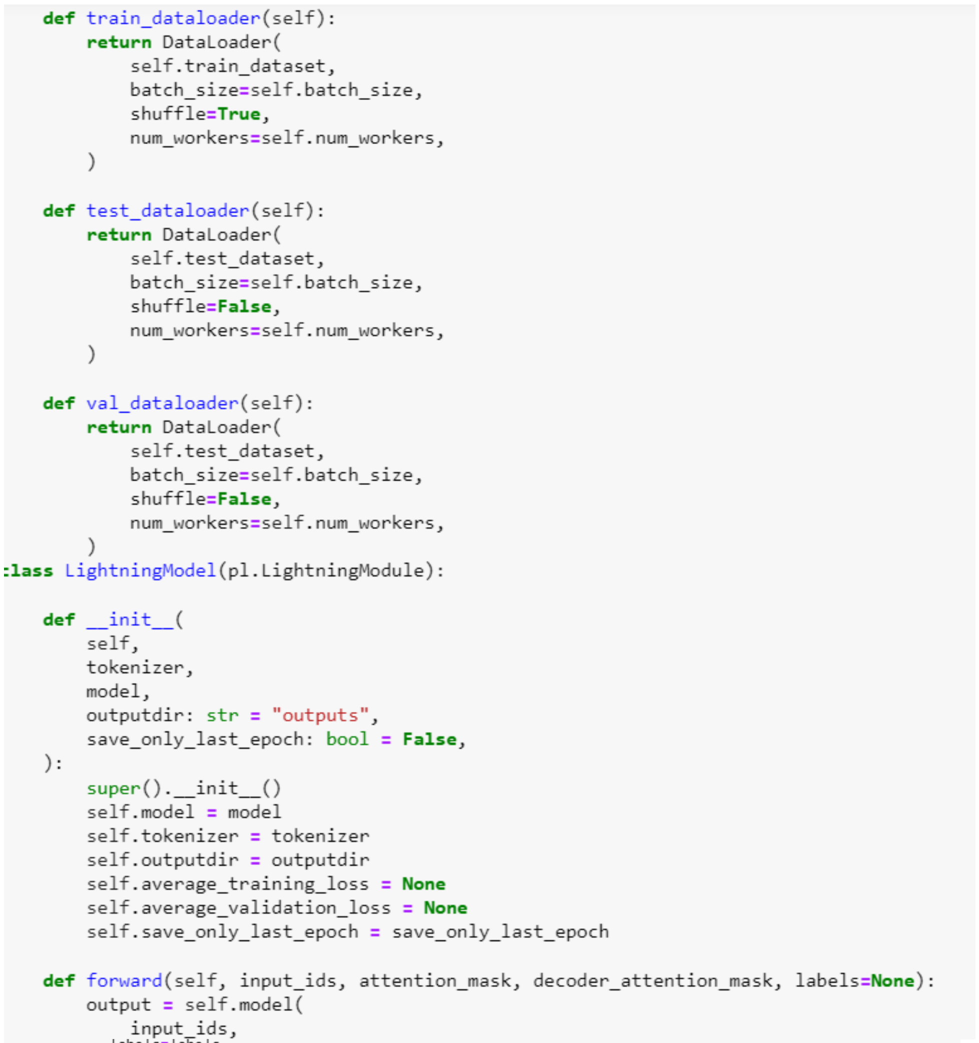


Model Architecture

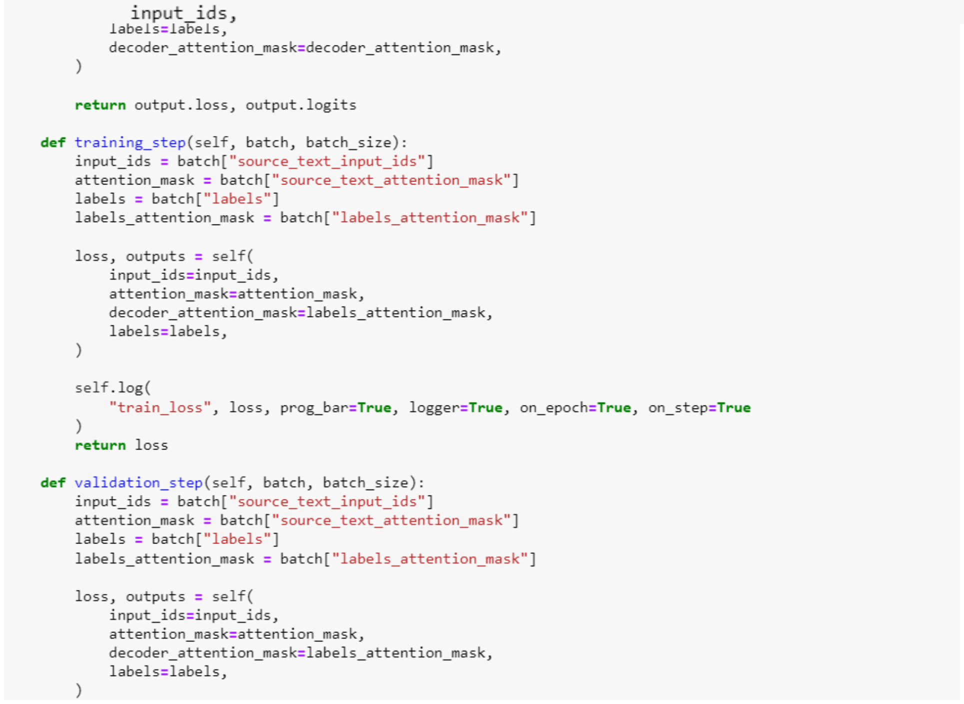
```
torch.cuda.empty_cache()
pl.seed_everything(42)
class PyTorchDataModule(Dataset):
    def __init__(
        self,
        data: pd.DataFrame,
        tokenizer: PreTrainedTokenizer,
        source_max_token_len: int = 512,
        target_max_token_len: int = 512,
    ):
        self.tokenizer = tokenizer
        self.data = data
        self.source_max_token_len = source_max_token_len
        self.target_max_token_len = target_max_token_len

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index: int):

        data_row = self.data.iloc[index]
        source_text = data_row["source_text"]

        source_text_encoding = self.tokenizer(
            source_text,
            max_length=self.source_max_token_len,
            padding="max_length",
            truncation=True,
            return_attention_mask=True,
            add_special_tokens=True,
            return_tensors="pt",
        )

        target_text_encoding = self.tokenizer(
            data_row["target_text"],
            max_length=self.target_max_token_len,
            padding="max_length",
            truncation=True,
            return_attention_mask=True,
            add_special_tokens=True,
            return_tensors="pt",
```



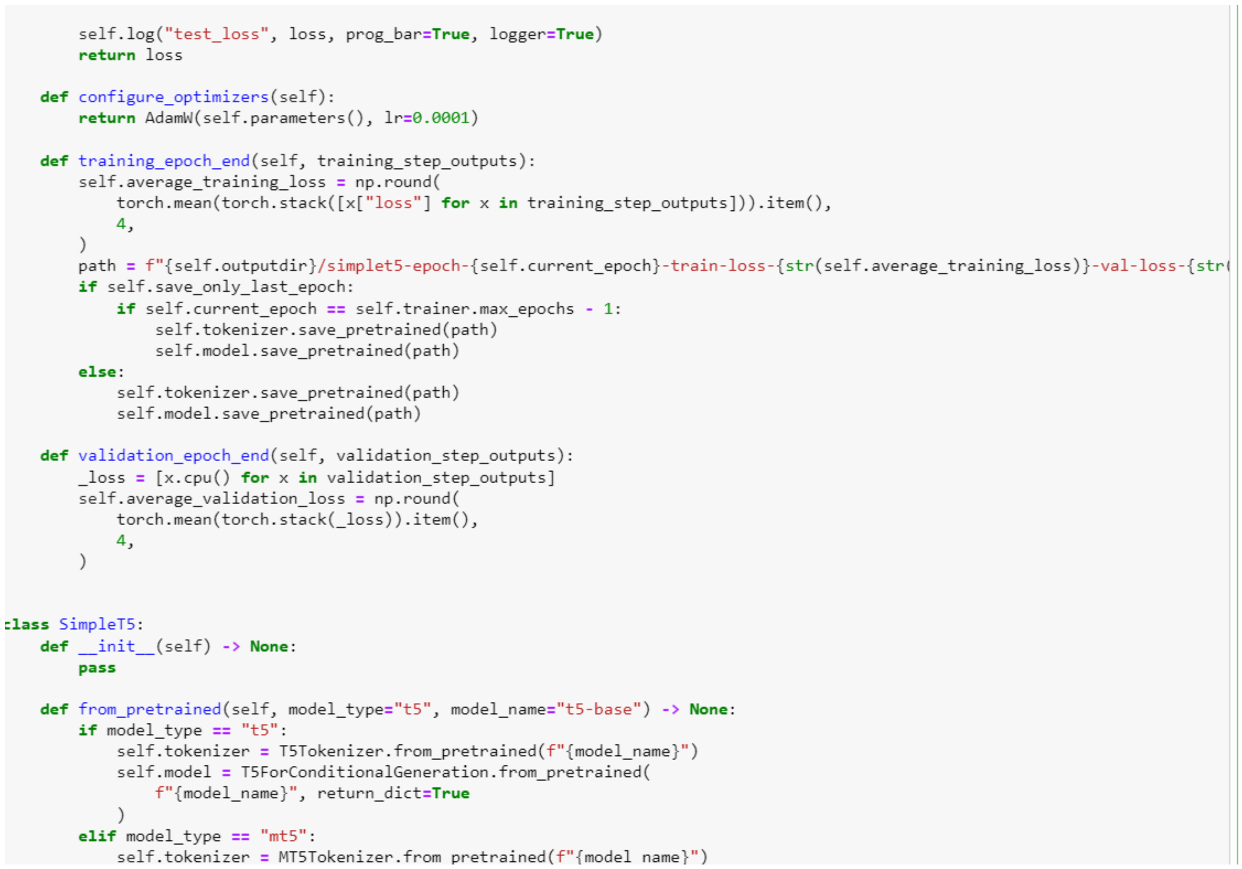
```python
        labels = target_text_encoding["input_ids"]
        labels[
            labels == 0
        ] = -100
        return dict(
            source_text_input_ids=source_text_encoding["input_ids"].flatten(),
            source_text_attention_mask=source_text_encoding["attention_mask"].flatten(),
            labels=labels.flatten(),
            labels_attention_mask=target_text_encoding["attention_mask"].flatten(),
        )
class LightningDataModule(pl.LightningDataModule):

    def __init__(
        self,
        train_df: pd.DataFrame,
        test_df: pd.DataFrame,
        tokenizer: PreTrainedTokenizer,
        batch_size: int = 4,
        source_max_token_len: int = 512,
        target_max_token_len: int = 512,
        num_workers: int = 2,
    ):
        super().__init__()

        self.train_df = train_df
        self.test_df = test_df
        self.batch_size = batch_size
        self.tokenizer = tokenizer
        self.source_max_token_len = source_max_token_len
        self.target_max_token_len = target_max_token_len
        self.num_workers = num_workers

    def setup(self, stage=None):
        self.train_dataset = PyTorchDataModule(
            self.train_df,
            self.tokenizer,
            self.source_max_token_len,
            self.target_max_token_len,
        )
        self.test_dataset = PyTorchDataModule(
            self.test_df,
            self.tokenizer,
            self.source_max_token_len,
```



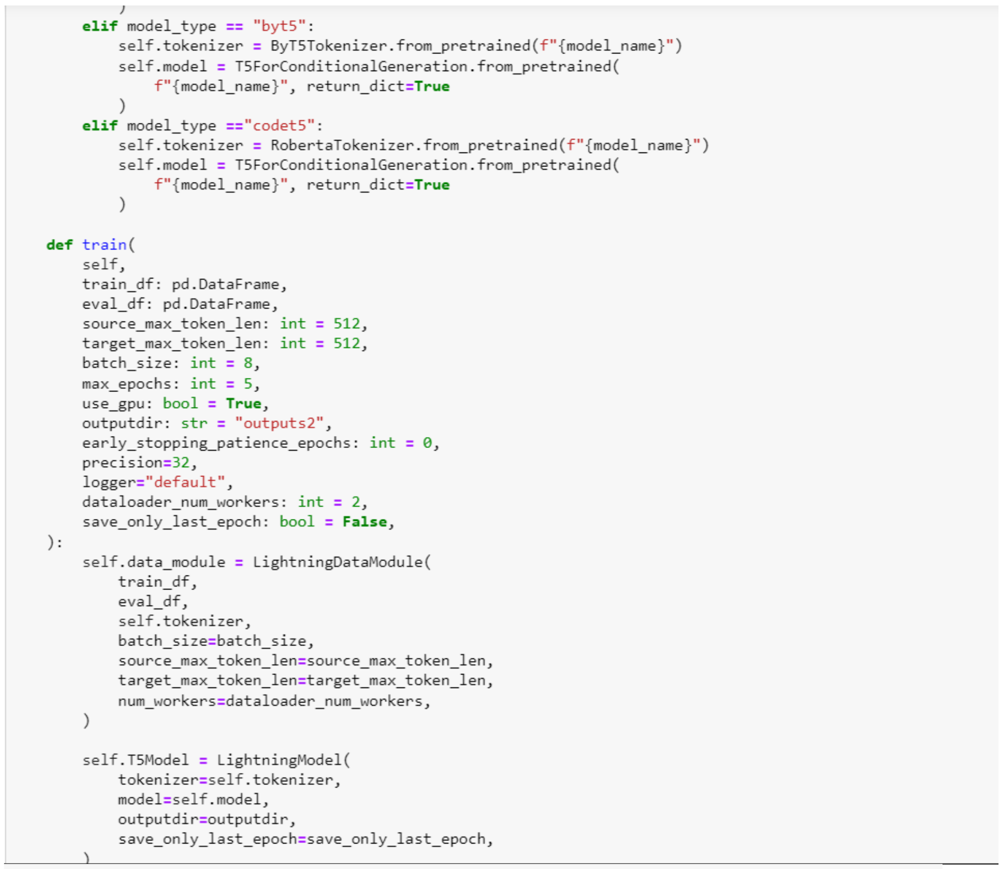
```
    def train_dataloader(self):
        return DataLoader(
            self.train_dataset,
            batch_size=self.batch_size,
            shuffle=True,
            num_workers=self.num_workers,
        )

    def test_dataloader(self):
        return DataLoader(
            self.test_dataset,
            batch_size=self.batch_size,
            shuffle=False,
            num_workers=self.num_workers,
        )

    def val_dataloader(self):
        return DataLoader(
            self.test_dataset,
            batch_size=self.batch_size,
            shuffle=False,
            num_workers=self.num_workers,
        )
lass LightningModel(pl.LightningModule):

    def __init__(
        self,
        tokenizer,
        model,
        outputdir: str = "outputs",
        save_only_last_epoch: bool = False,
    ):
        super().__init__()
        self.model = model
        self.tokenizer = tokenizer
        self.outputdir = outputdir
        self.average_training_loss = None
        self.average_validation_loss = None
        self.save_only_last_epoch = save_only_last_epoch

    def forward(self, input_ids, attention_mask, decoder_attention_mask, labels=None):
        output = self.model(
            input_ids,
```



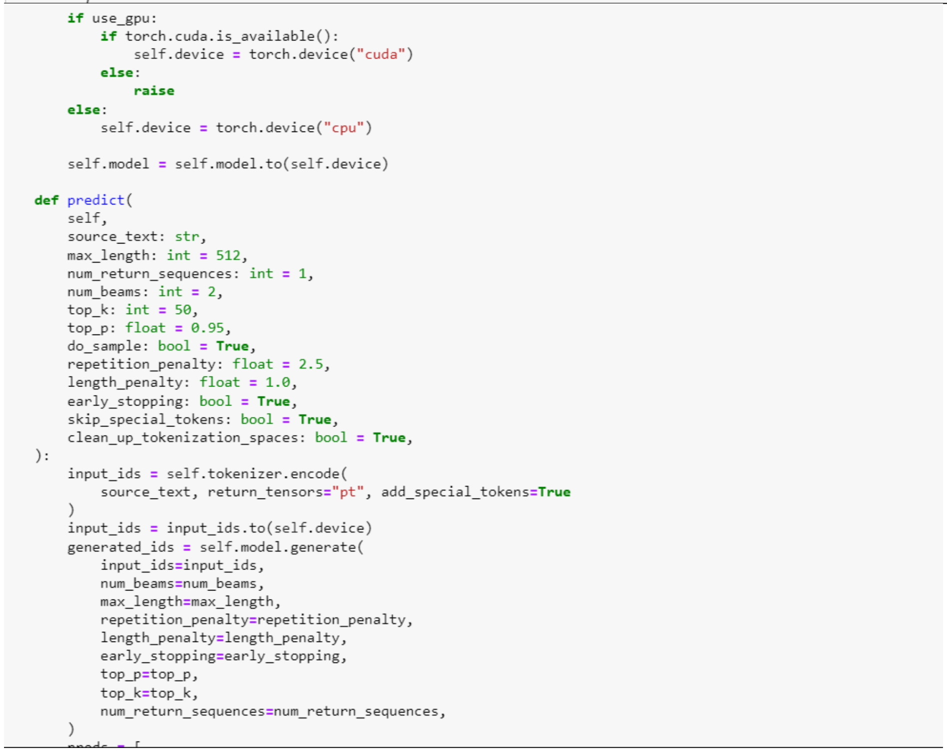
```
            labels=labels,
            decoder_attention_mask=decoder_attention_mask,
        )

        return output.loss, output.logits

    def training_step(self, batch, batch_size):
        input_ids = batch["source_text_input_ids"]
        attention_mask = batch["source_text_attention_mask"]
        labels = batch["labels"]
        labels_attention_mask = batch["labels_attention_mask"]

        loss, outputs = self(
            input_ids=input_ids,
            attention_mask=attention_mask,
            decoder_attention_mask=labels_attention_mask,
            labels=labels,
        )

        self.log(
            "train_loss", loss, prog_bar=True, logger=True, on_epoch=True, on_step=True
        )
        return loss

    def validation_step(self, batch, batch_size):
        input_ids = batch["source_text_input_ids"]
        attention_mask = batch["source_text_attention_mask"]
        labels = batch["labels"]
        labels_attention_mask = batch["labels_attention_mask"]

        loss, outputs = self(
            input_ids=input_ids,
            attention_mask=attention_mask,
            decoder_attention_mask=labels_attention_mask,
            labels=labels,
        )
```



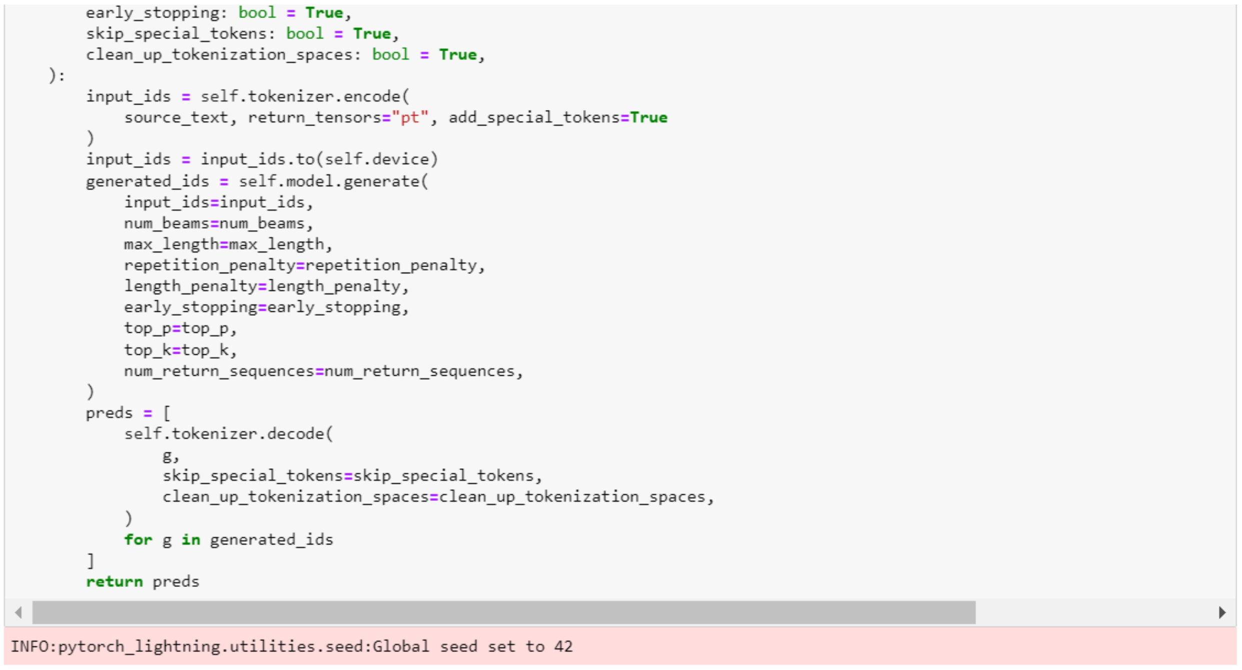
```
        self.log("test_loss", loss, prog_bar=True, logger=True)
        return loss

    def configure_optimizers(self):
        return AdamW(self.parameters(), lr=0.0001)

    def training_epoch_end(self, training_step_outputs):
        self.average_training_loss = np.round(
            torch.mean(torch.stack([x["loss"] for x in training_step_outputs])).item(),
            4,
        )
        path = f"{self.outputdir}/simplet5-epoch-{self.current_epoch}-train-loss-{str(self.average_training_loss)}-val-loss-{str(
        if self.save_only_last_epoch:
            if self.current_epoch == self.trainer.max_epochs - 1:
                self.tokenizer.save_pretrained(path)
                self.model.save_pretrained(path)
        else:
            self.tokenizer.save_pretrained(path)
            self.model.save_pretrained(path)

    def validation_epoch_end(self, validation_step_outputs):
        _loss = [x.cpu() for x in validation_step_outputs]
        self.average_validation_loss = np.round(
            torch.mean(torch.stack(_loss)).item(),
            4,
        )


class SimpleT5:
    def __init__(self) -> None:
        pass

    def from_pretrained(self, model_type="t5", model_name="t5-base") -> None:
        if model_type == "t5":
            self.tokenizer = T5Tokenizer.from_pretrained(f"{model_name}")
            self.model = T5ForConditionalGeneration.from_pretrained(
                f"{model_name}", return_dict=True
            )
        elif model_type == "mt5":
            self.tokenizer = MT5Tokenizer.from_pretrained(f"{model_name}")
```



```
            )
        elif model_type == "byt5":
            self.tokenizer = ByT5Tokenizer.from_pretrained(f"{model_name}")
            self.model = T5ForConditionalGeneration.from_pretrained(
                f"{model_name}", return_dict=True
            )
        elif model_type =="codet5":
            self.tokenizer = RobertaTokenizer.from_pretrained(f"{model_name}")
            self.model = T5ForConditionalGeneration.from_pretrained(
                f"{model_name}", return_dict=True
            )

    def train(
        self,
        train_df: pd.DataFrame,
        eval_df: pd.DataFrame,
        source_max_token_len: int = 512,
        target_max_token_len: int = 512,
        batch_size: int = 8,
        max_epochs: int = 5,
        use_gpu: bool = True,
        outputdir: str = "outputs2",
        early_stopping_patience_epochs: int = 0,
        precision=32,
        logger="default",
        dataloader_num_workers: int = 2,
        save_only_last_epoch: bool = False,
    ):
        self.data_module = LightningDataModule(
            train_df,
            eval_df,
            self.tokenizer,
            batch_size=batch_size,
            source_max_token_len=source_max_token_len,
            target_max_token_len=target_max_token_len,
            num_workers=dataloader_num_workers,
        )

        self.T5Model = LightningModel(
            tokenizer=self.tokenizer,
            model=self.model,
            outputdir=outputdir,
            save_only_last_epoch=save_only_last_epoch,
        )
```

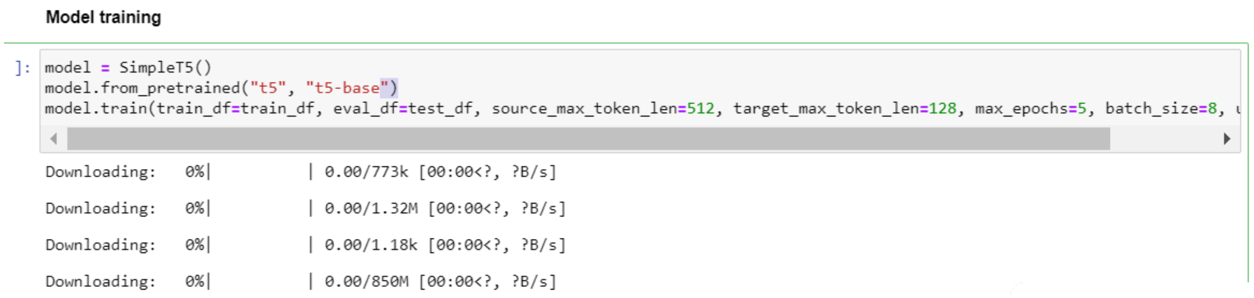


```
        if use_gpu:
            if torch.cuda.is_available():
                self.device = torch.device("cuda")
            else:
                raise
        else:
            self.device = torch.device("cpu")

        self.model = self.model.to(self.device)

    def predict(
        self,
        source_text: str,
        max_length: int = 512,
        num_return_sequences: int = 1,
        num_beams: int = 2,
        top_k: int = 50,
        top_p: float = 0.95,
        do_sample: bool = True,
        repetition_penalty: float = 2.5,
        length_penalty: float = 1.0,
        early_stopping: bool = True,
        skip_special_tokens: bool = True,
        clean_up_tokenization_spaces: bool = True,
    ):
        input_ids = self.tokenizer.encode(
            source_text, return_tensors="pt", add_special_tokens=True
        )
        input_ids = input_ids.to(self.device)
        generated_ids = self.model.generate(
            input_ids=input_ids,
            num_beams=num_beams,
            max_length=max_length,
            repetition_penalty=repetition_penalty,
            length_penalty=length_penalty,
            early_stopping=early_stopping,
            top_p=top_p,
            top_k=top_k,
            num_return_sequences=num_return_sequences,
        )
```



```
        early_stopping: bool = True,
        skip_special_tokens: bool = True,
        clean_up_tokenization_spaces: bool = True,
    ):
        input_ids = self.tokenizer.encode(
            source_text, return_tensors="pt", add_special_tokens=True
        )
        input_ids = input_ids.to(self.device)
        generated_ids = self.model.generate(
            input_ids=input_ids,
            num_beams=num_beams,
            max_length=max_length,
            repetition_penalty=repetition_penalty,
            length_penalty=length_penalty,
            early_stopping=early_stopping,
            top_p=top_p,
            top_k=top_k,
            num_return_sequences=num_return_sequences,
        )
        preds = [
            self.tokenizer.decode(
                g,
                skip_special_tokens=skip_special_tokens,
                clean_up_tokenization_spaces=clean_up_tokenization_spaces,
            )
            for g in generated_ids
        ]
        return preds
```

```
INFO:pytorch_lightning.utilities.seed:Global seed set to 42
```



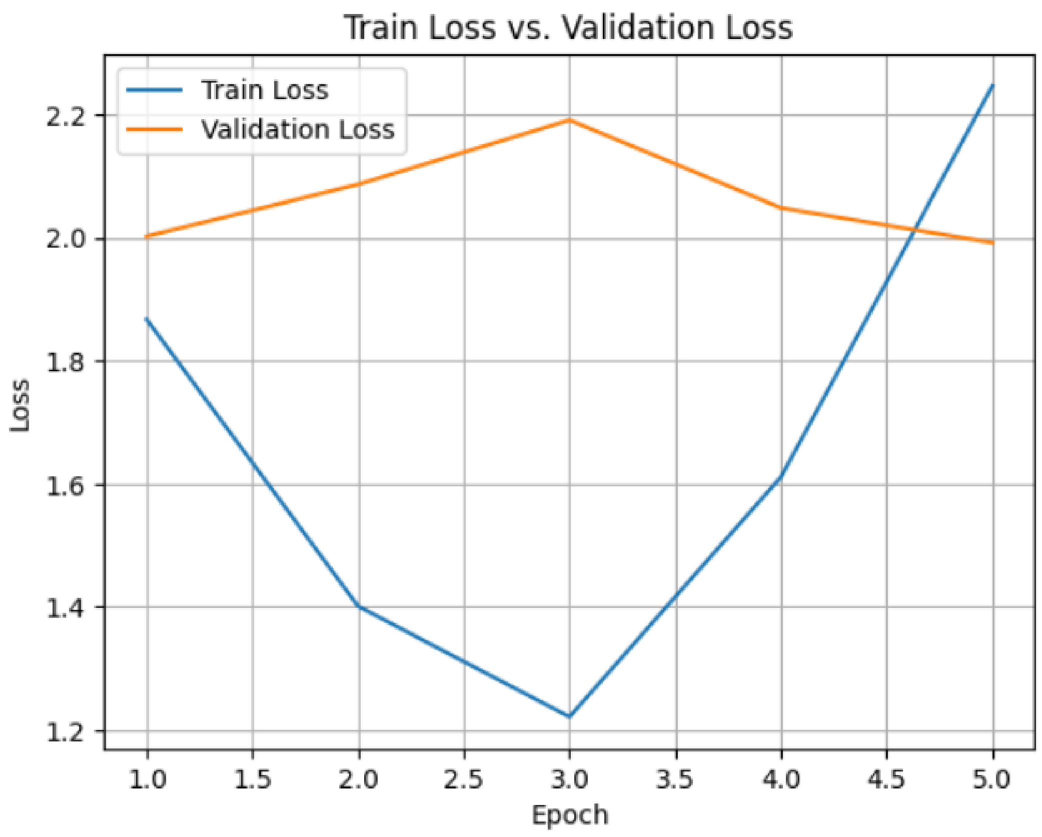
Model training

```
]: model = SimpleT5()
   model.from_pretrained("t5", "t5-base")
   model.train(train_df=train_df, eval_df=test_df, source_max_token_len=512, target_max_token_len=128, max_epochs=5, batch_size=8, u
```

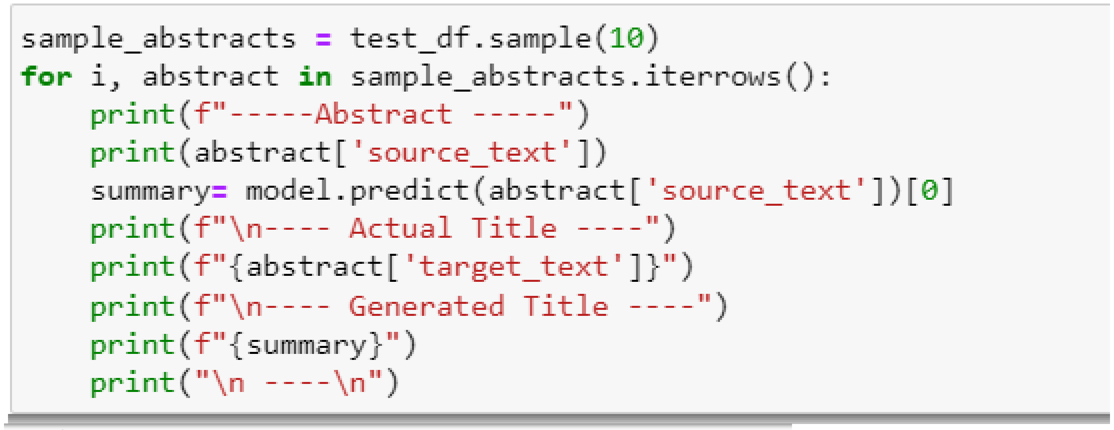
```
Downloading:   0%|          | 0.00/773k [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/1.32M [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/1.18k [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/850M [00:00<?, ?B/s]
```

SimpleT5() library is used to train a T5 (Text-to-Text transfer model) for generating the title.

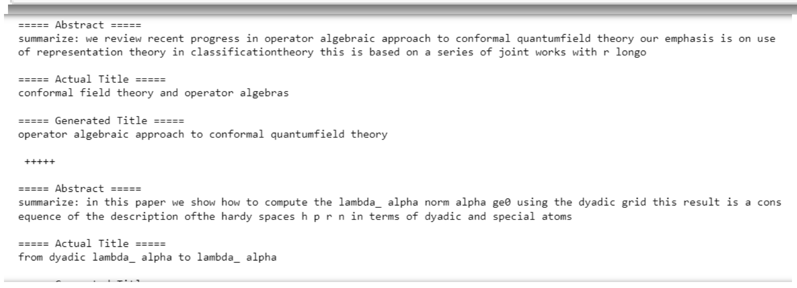
A pretrained T5 model called t5-base is used for this task, and trained with training data.
The model is trained for 5 epochs and the loss is depicted in the below graph.



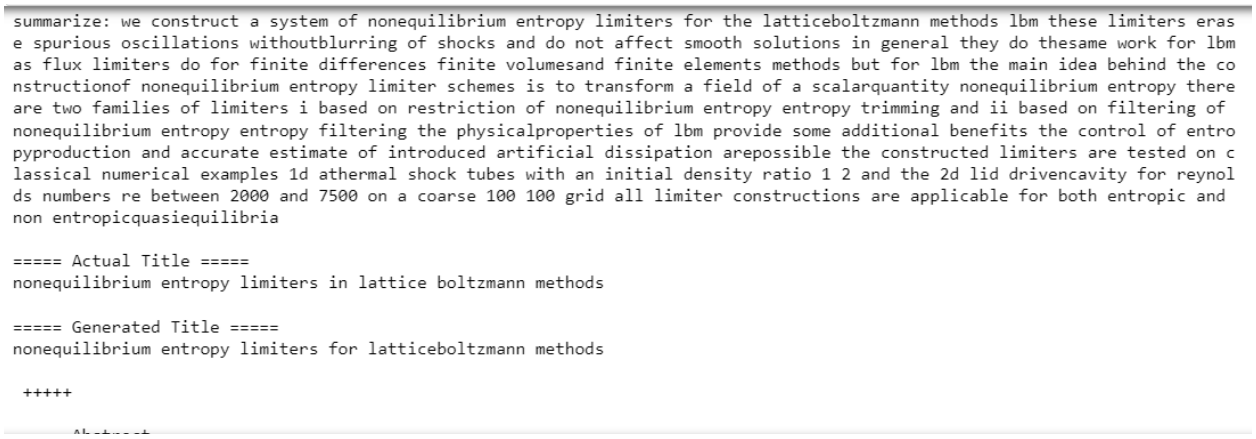
The trained model is tested by giving abstracts of few research paper and observing the results generated with the actual target_title.



```python
sample_abstracts = test_df.sample(10)
for i, abstract in sample_abstracts.iterrows():
    print(f"-----Abstract -----")
    print(abstract['source_text'])
    summary= model.predict(abstract['source_text'])[0]
    print(f"\n---- Actual Title ----")
    print(f"{abstract['target_text']}")
    print(f"\n---- Generated Title ----")
    print(f"{summary}")
    print("\n ----\n")
```

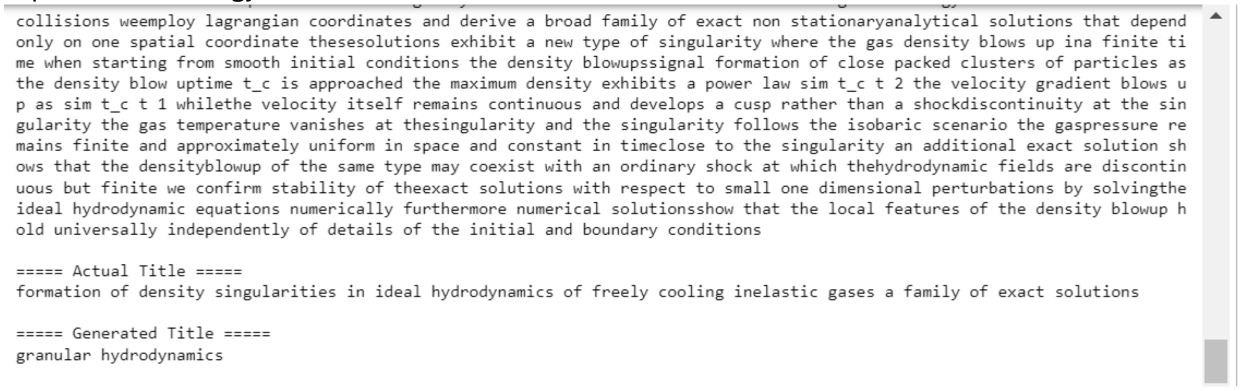


```
===== Abstract =====
summarize: we review recent progress in operator algebraic approach to conformal quantumfield theory our emphasis is on use
of representation theory in classificationtheory this is based on a series of joint works with r longo

===== Actual Title =====
conformal field theory and operator algebras

===== Generated Title =====
operator algebraic approach to conformal quantumfield theory

 +++++

===== Abstract =====
summarize: in this paper we show how to compute the lambda_ alpha norm alpha ge0 using the dyadic grid this result is a cons
equence of the description ofthe hardy spaces h p r n in terms of dyadic and special atoms

===== Actual Title =====
from dyadic lambda_ alpha to lambda_ alpha
```



```
summarize: we construct a system of nonequilibrium entropy limiters for the latticeboltzmann methods lbm these limiters eras
e spurious oscillations withoutblurring of shocks and do not affect smooth solutions in general they do thesame work for lbm
as flux limiters do for finite differences finite volumesand finite elements methods but for lbm the main idea behind the co
nstructionof nonequilibrium entropy limiter schemes is to transform a field of a scalarquantity nonequilibrium entropy there
are two families of limiters i based on restriction of nonequilibrium entropy entropy trimming and ii based on filtering of
nonequilibrium entropy entropy filtering the physicalproperties of lbm provide some additional benefits the control of entro
pyproduction and accurate estimate of introduced artificial dissipation arepossible the constructed limiters are tested on c
lassical numerical examples 1d athermal shock tubes with an initial density ratio 1 2 and the 2d lid drivencavity for reynol
ds numbers re between 2000 and 7500 on a coarse 100 100 grid all limiter constructions are applicable for both entropic and
non entropicquasiequilibria

===== Actual Title =====
nonequilibrium entropy limiters in lattice boltzmann methods

===== Generated Title =====
nonequilibrium entropy limiters for latticeboltzmann methods

 +++++
```

The model was successful in generating titles for the given summary, the resultant title seems to capture the ideology of the abstract and it is almost close to the actual title.

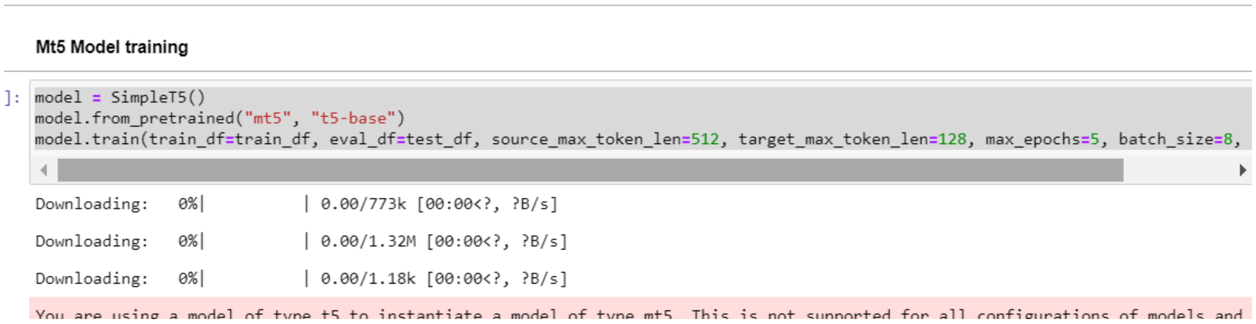


```
collisions weemploy lagrangian coordinates and derive a broad family of exact non stationaryanalytical solutions that depend
only on one spatial coordinate thesesolutions exhibit a new type of singularity where the gas density blows up ina finite ti
me when starting from smooth initial conditions the density blowupssignal formation of close packed clusters of particles as
the density blow uptime t_c is approached the maximum density exhibits a power law sim t_c t 2 the velocity gradient blows u
p as sim t_c t 1 whilethe velocity itself remains continuous and develops a cusp rather than a shockdiscontinuity at the sin
gularity the gas temperature vanishes at thesingularity and the singularity follows the isobaric scenario the gaspressure re
mains finite and approximately uniform in space and constant in timeclose to the singularity an additional exact solution sh
ows that the densityblowup of the same type may coexist with an ordinary shock at which thehydrodynamic fields are discontin
uous but finite we confirm stability of theexact solutions with respect to small one dimensional perturbations by solvingthe
ideal hydrodynamic equations numerically furthermore numerical solutionsshow that the local features of the density blowup h
old universally independently of details of the initial and boundary conditions

===== Actual Title =====
formation of density singularities in ideal hydrodynamics of freely cooling inelastic gases a family of exact solutions

===== Generated Title =====
granular hydrodynamics
```

And yes, there are cases where the generated title is too small and meaningless but relevant to the abstract.

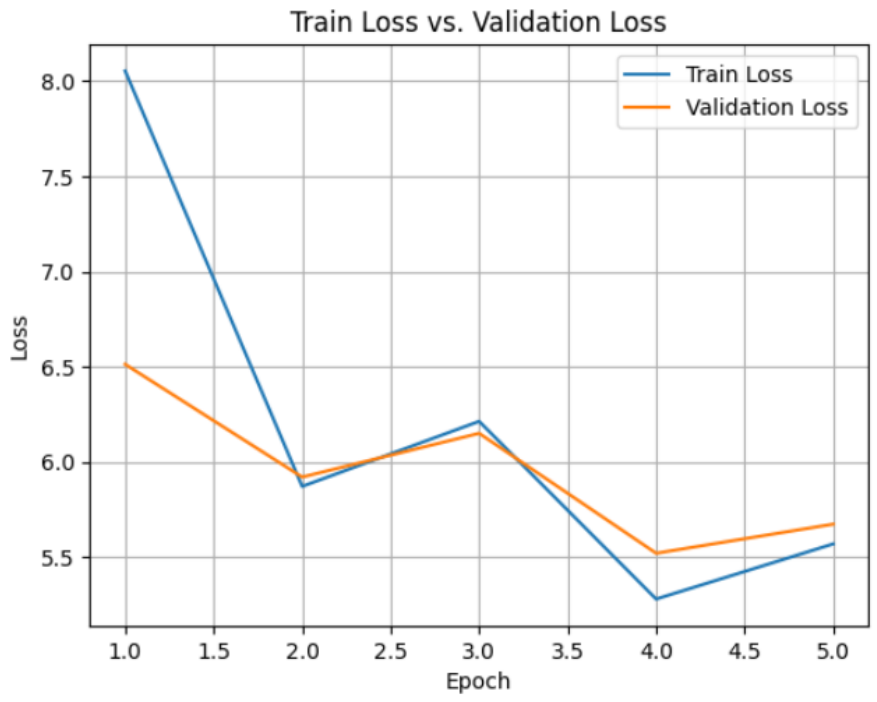
Mt5 Model:

The Multilingual T5 (mT5) model is an extension of the T5 architecture designed to handle multiple languages. This model not only allows cross-lingual title generation but also improves performance.



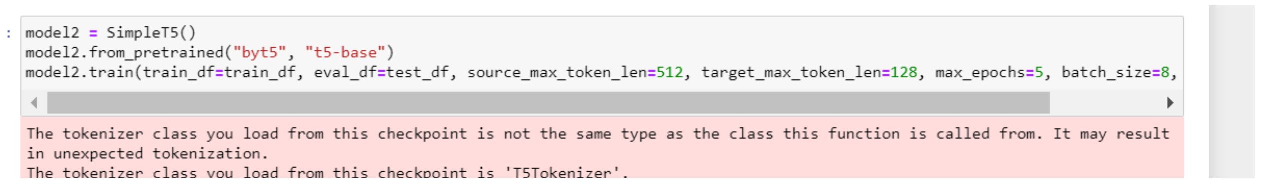
Mt5 Model training

```
]: model = SimpleT5()
   model.from_pretrained("mt5", "t5-base")
   model.train(train_df=train_df, eval_df=test_df, source_max_token_len=512, target_max_token_len=128, max_epochs=5, batch_size=8,

   Downloading:   0%|          | 0.00/773k [00:00<?, ?B/s]

   Downloading:   0%|          | 0.00/1.32M [00:00<?, ?B/s]

   Downloading:   0%|          | 0.00/1.18k [00:00<?, ?B/s]

   You are using a model of type t5 to instantiate a model of type mt5. This is not supported for all configurations of models and
```



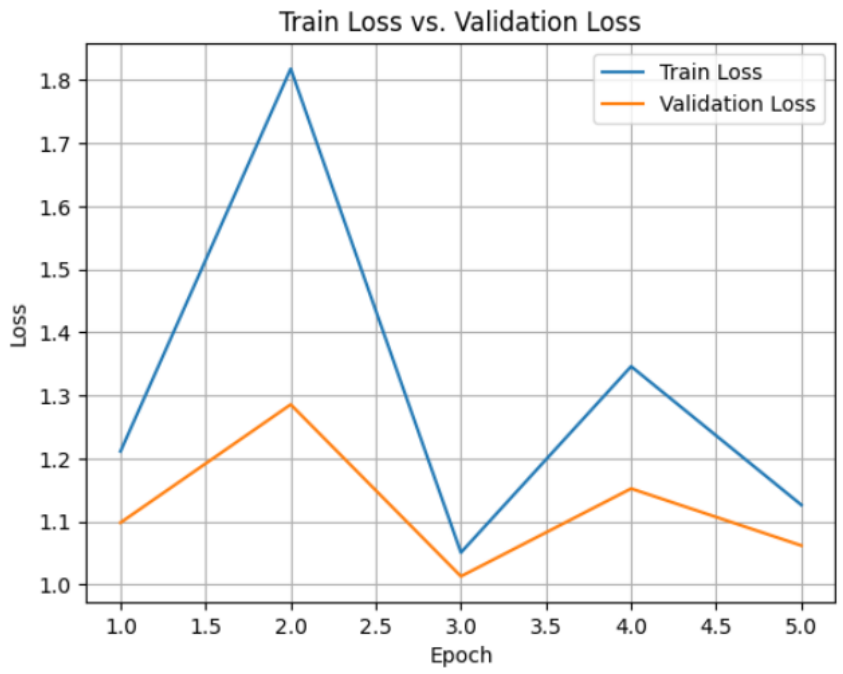
Byt5 Model:

Beyond T5 (Byt5) aims to enhance the capabilities of title generation further. Byt5 model introduces advancement in the transformer-based model by using attention mechanisms and regularization techniques.

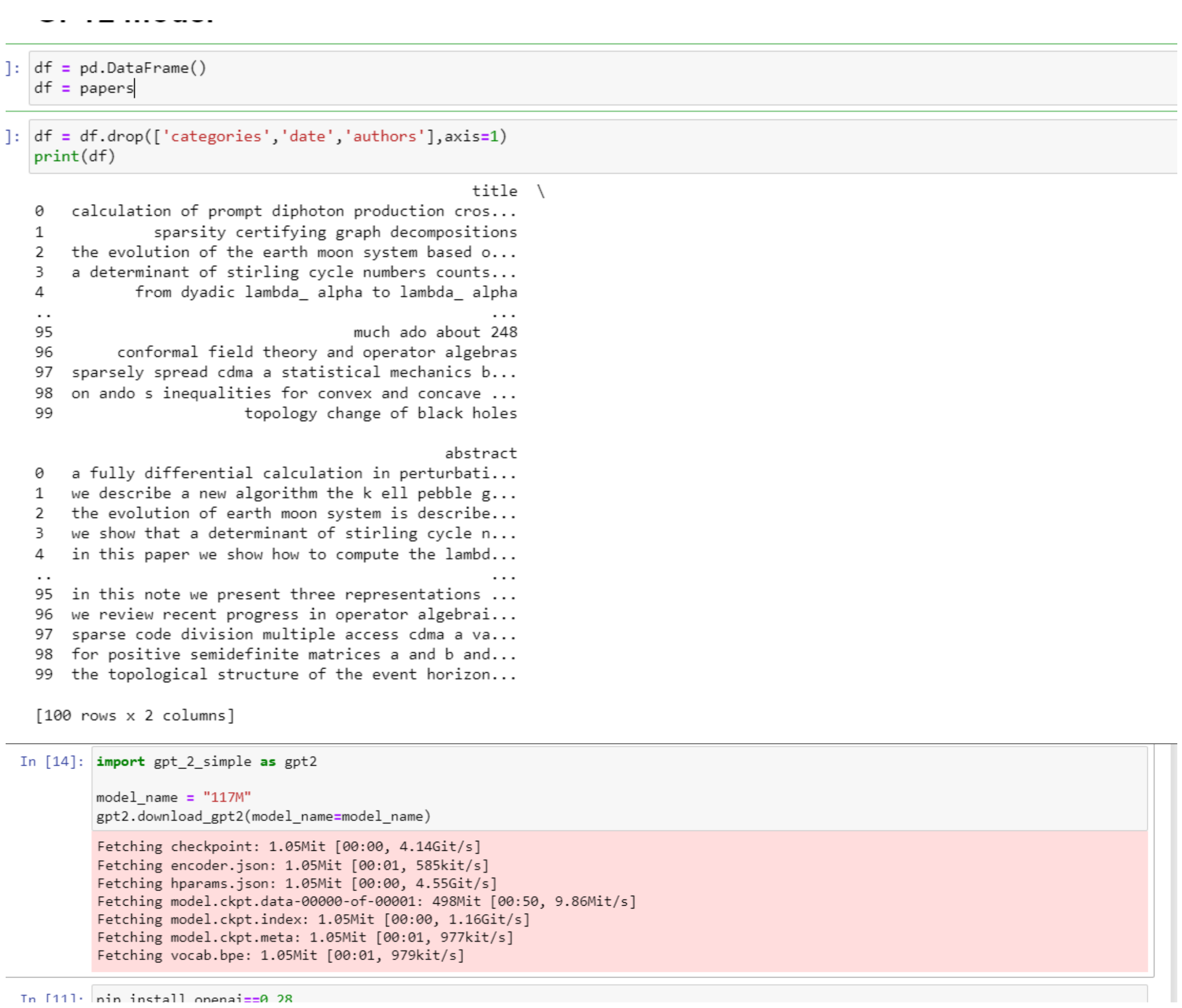


```
model2 = SimpleT5()
model2.from_pretrained("byt5", "t5-base")
model2.train(train_df=train_df, eval_df=test_df, source_max_token_len=512, target_max_token_len=128, max_epochs=5, batch_size=8,
```

```
The tokenizer class you load from this checkpoint is not the same type as the class this function is called from. It may result
in unexpected tokenization.
The tokenizer class you load from this checkpoint is 'T5Tokenizer'.
```


Train Loss vs. Validation Loss
Train Loss
Validation Loss
Loss
Epoch
1.0
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

GP2 Model:



```
]: df = pd.DataFrame()
   df = papers
```

```
]: df = df.drop(['categories','date','authors'],axis=1)
   print(df)
```

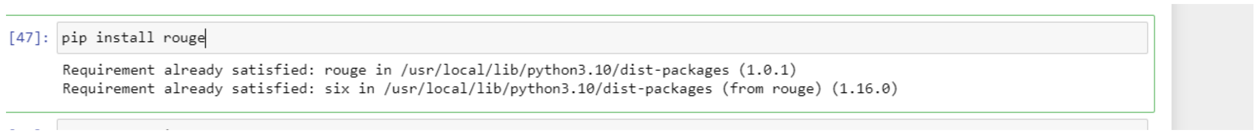
```
                                                title  \
0   calculation of prompt diphoton production cros...
1            sparsity certifying graph decompositions
2   the evolution of the earth moon system based o...
3   a determinant of stirling cycle numbers counts...
4         from dyadic lambda_ alpha to lambda_ alpha
..                                                ...
95                                 much ado about 248
96      conformal field theory and operator algebras
97  sparsely spread cdma a statistical mechanics b...
98  on ando s inequalities for convex and concave ...
99                        topology change of black holes

                                             abstract
0   a fully differential calculation in perturbati...
1   we describe a new algorithm the k ell pebble g...
2   the evolution of earth moon system is describe...
3   we show that a determinant of stirling cycle n...
4   in this paper we show how to compute the lambd...
..                                                ...
95  in this note we present three representations ...
96  we review recent progress in operator algebrai...
97  sparse code division multiple access cdma a va...
98  for positive semidefinite matrices a and b and...
99  the topological structure of the event horizon...

[100 rows x 2 columns]
```

```
In [14]: import gpt_2_simple as gpt2

         model_name = "117M"
         gpt2.download_gpt2(model_name=model_name)
```

```
Fetching checkpoint: 1.05Mit [00:00, 4.14Git/s]
Fetching encoder.json: 1.05Mit [00:01, 585kit/s]
Fetching hparams.json: 1.05Mit [00:00, 4.55Git/s]
Fetching model.ckpt.data-00000-of-00001: 498Mit [00:50, 9.86Mit/s]
Fetching model.ckpt.index: 1.05Mit [00:00, 1.16Git/s]
Fetching model.ckpt.meta: 1.05Mit [00:01, 977kit/s]
Fetching vocab.bpe: 1.05Mit [00:01, 979kit/s]
```

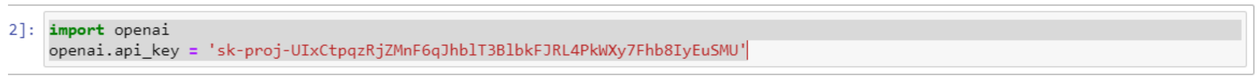
```
In [11]: pip install openai==0.28
```

gpt_2_simple is a python package that allows to use gpt 2 through API. 117M is a small model that is released by OpenAI and as the name suggest it has 117 million parameters and it can be used for text generation tasks.



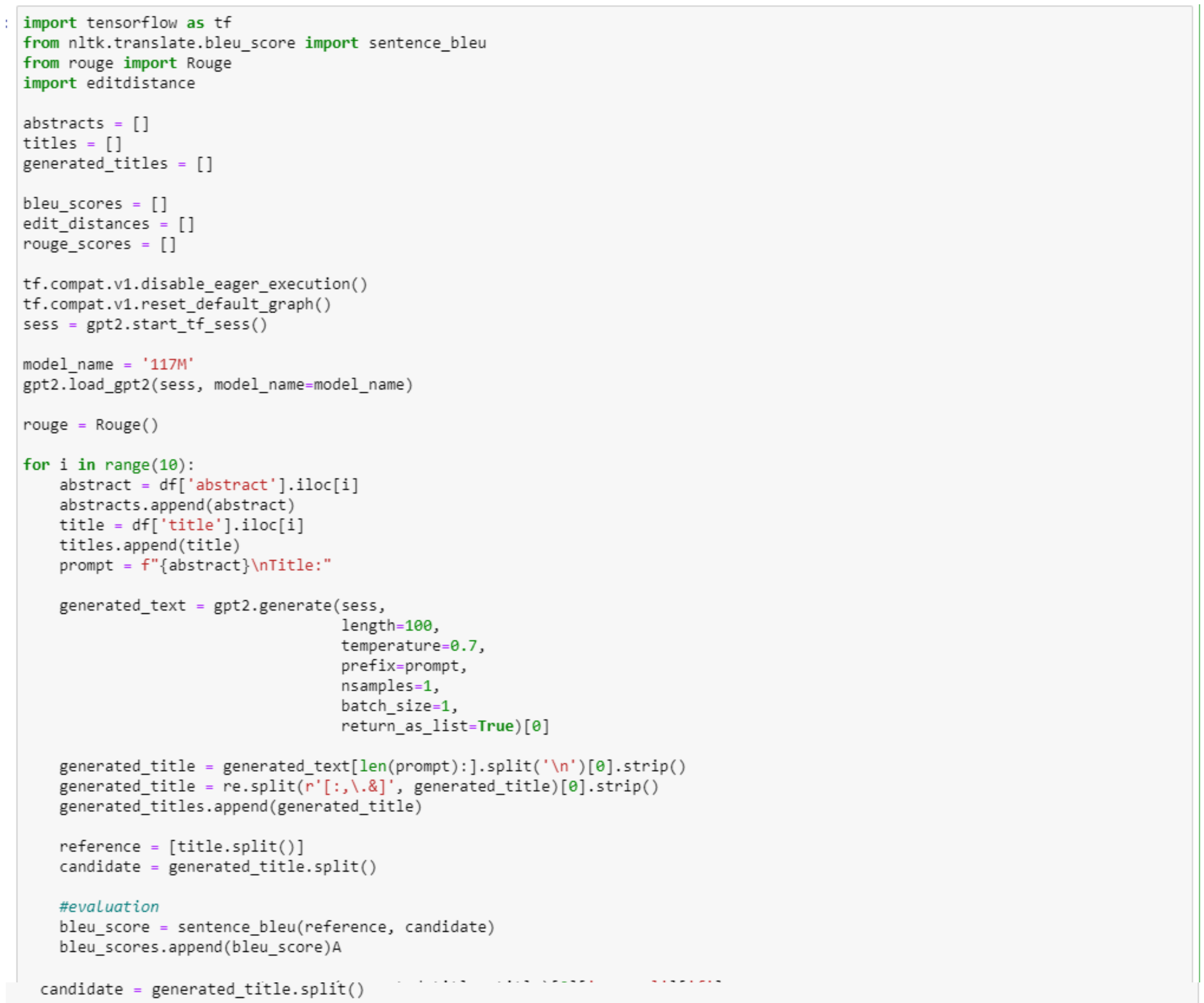
```
[47]: pip install rouge

      Requirement already satisfied: rouge in /usr/local/lib/python3.10/dist-packages (1.0.1)
      Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from rouge) (1.16.0)
```

Rouge python package is used to analyze and visualize the accuracy of model by comparing the generated title with actual title.

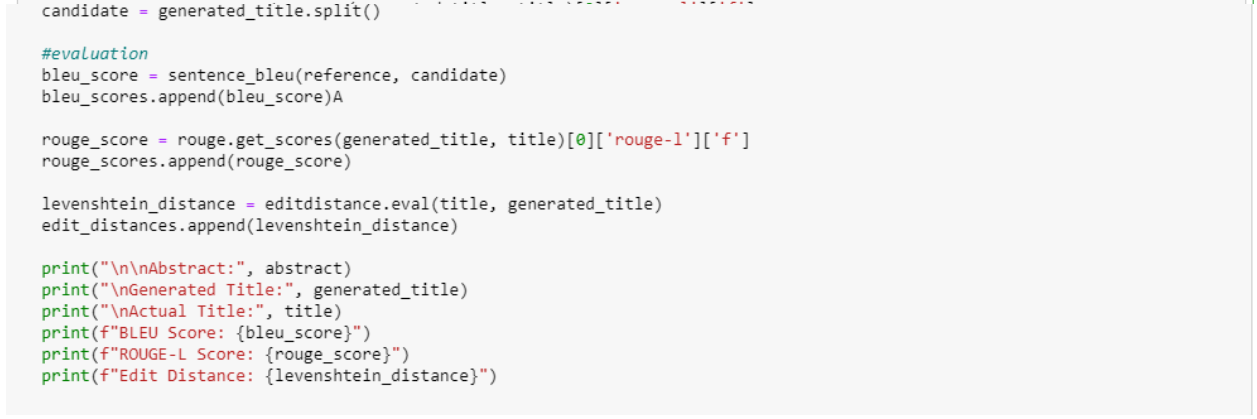


```
2]: import openai
    openai.api_key = 'sk-proj-UIxCtpqzRjZMnF6qJhblT3BlbkFJRL4PkWXy7Fhb8IyEuSMU'
```

The api key is generated from OpenAI website and allows us to authenticate and access the model.

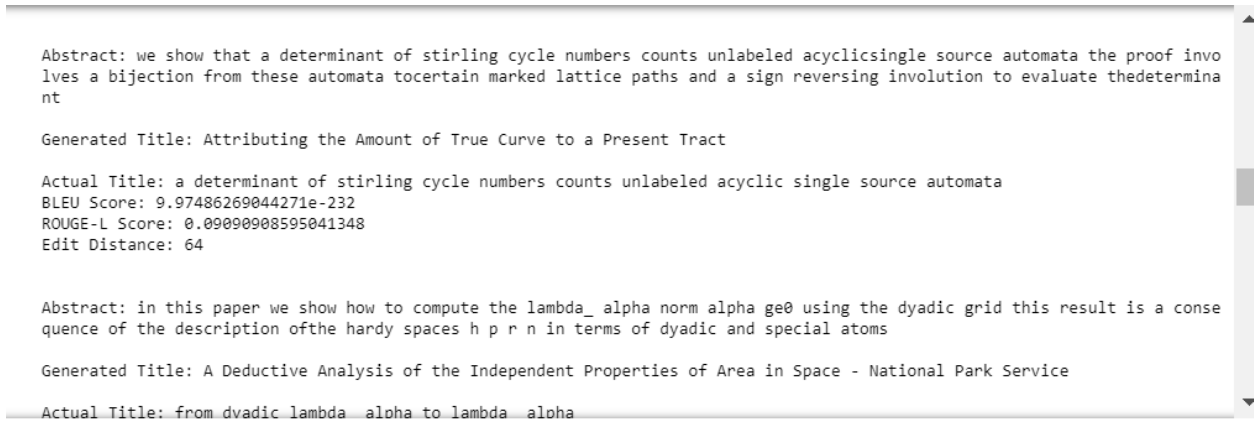


```
import tensorflow as tf
from nltk.translate.bleu_score import sentence_bleu
from rouge import Rouge
import editdistance

abstracts = []
titles = []
generated_titles = []

bleu_scores = []
edit_distances = []
rouge_scores = []

tf.compat.v1.disable_eager_execution()
tf.compat.v1.reset_default_graph()
sess = gpt2.start_tf_sess()

model_name = '117M'
gpt2.load_gpt2(sess, model_name=model_name)

rouge = Rouge()

for i in range(10):
    abstract = df['abstract'].iloc[i]
    abstracts.append(abstract)
    title = df['title'].iloc[i]
    titles.append(title)
    prompt = f"{abstract}\nTitle:"

    generated_text = gpt2.generate(sess,
                                   length=100,
                                   temperature=0.7,
                                   prefix=prompt,
                                   nsamples=1,
                                   batch_size=1,
                                   return_as_list=True)[0]

    generated_title = generated_text[len(prompt):].split('\n')[0].strip()
    generated_title = re.split(r'[:;\.&]', generated_title)[0].strip()
    generated_titles.append(generated_title)

    reference = [title.split()]
    candidate = generated_title.split()

    #evaluation
    bleu_score = sentence_bleu(reference, candidate)
    bleu_scores.append(bleu_score)A
```



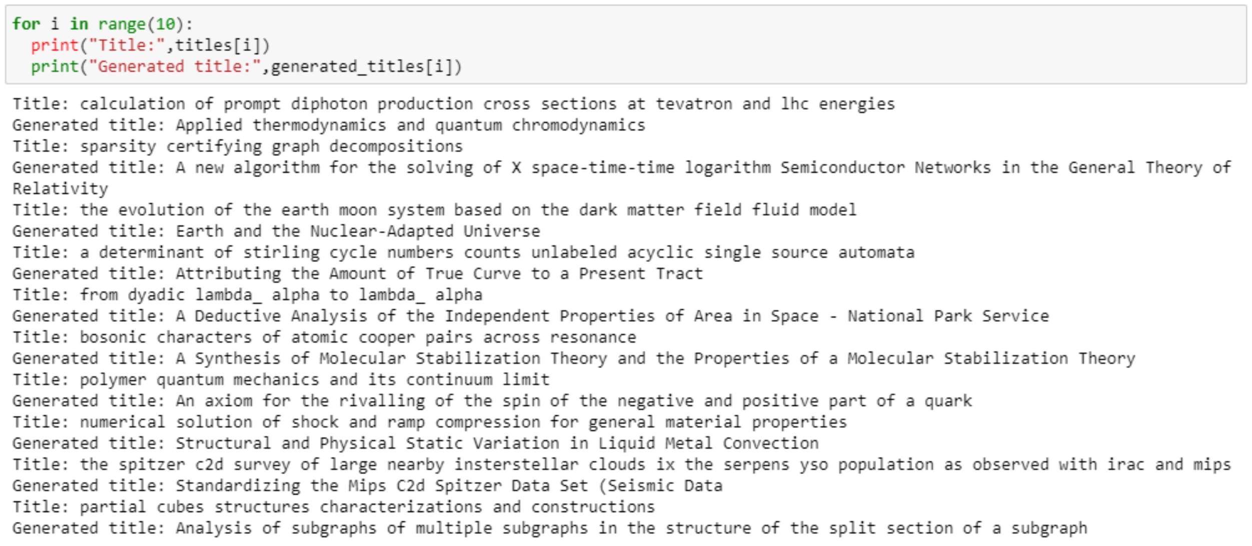
```
  candidate = generated_title.split()

  #evaluation
  bleu_score = sentence_bleu(reference, candidate)
  bleu_scores.append(bleu_score)A
```

```
  rouge_score = rouge.get_scores(generated_title, title)[0]['rouge-l']['f']
  rouge_scores.append(rouge_score)

  levenshtein_distance = editdistance.eval(title, generated_title)
  edit_distances.append(levenshtein_distance)

  print("\n\nAbstract:", abstract)
  print("\nGenerated Title:", generated_title)
  print("\nActual Title:", title)
  print(f"BLEU Score: {bleu_score}")
  print(f"ROUGE-L Score: {rouge_score}")
  print(f"Edit Distance: {levenshtein_distance}")
```



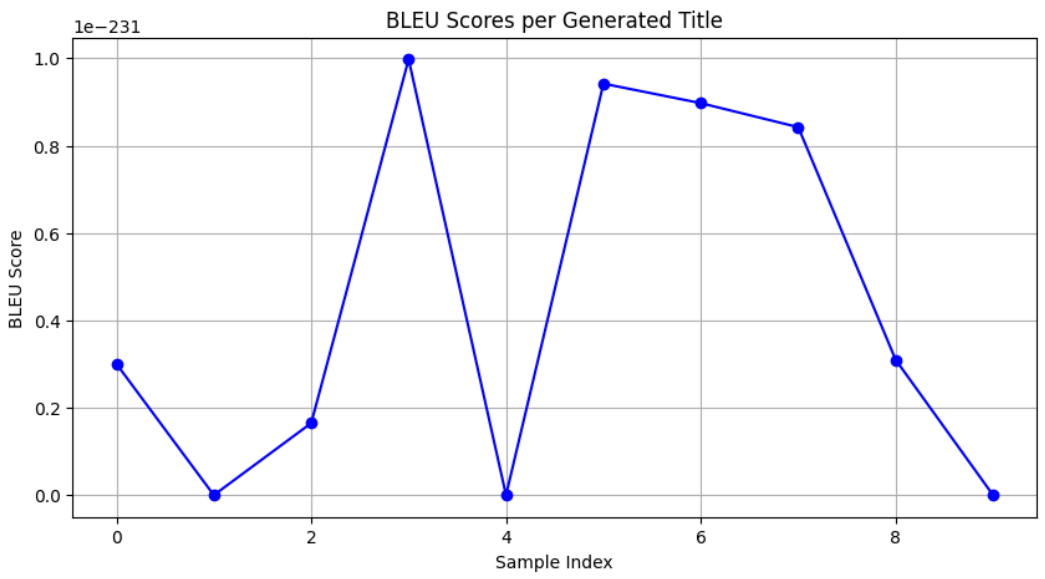
```
Abstract: we show that a determinant of stirling cycle numbers counts unlabeled acyclicsingle source automata the proof invo
lves a bijection from these automata tocertain marked lattice paths and a sign reversing involution to evaluate thedetermina
nt

Generated Title: Attributing the Amount of True Curve to a Present Tract

Actual Title: a determinant of stirling cycle numbers counts unlabeled acyclic single source automata
BLEU Score: 9.97486269044271e-232
ROUGE-L Score: 0.09090908595041348
Edit Distance: 64


Abstract: in this paper we show how to compute the lambda_ alpha norm alpha ge0 using the dyadic grid this result is a conse
quence of the description ofthe hardy spaces h p r n in terms of dyadic and special atoms

Generated Title: A Deductive Analysis of the Independent Properties of Area in Space - National Park Service

Actual Title: from dyadic lambda_ alpha to lambda_ alpha
```

In the above code snippet, the abstract is passed as a prompt to gpt2 and resultant title is compared with actual title. We concatenate the abstract with "title: " to create the prompt and get the title for each paper. For each of the title generated, we evaluate by calculating BLEU score, ROUGE-L score and edit distance between the generated and actual title.



```python
for i in range(10):
  print("Title:",titles[i])
  print("Generated title:",generated_titles[i])
```

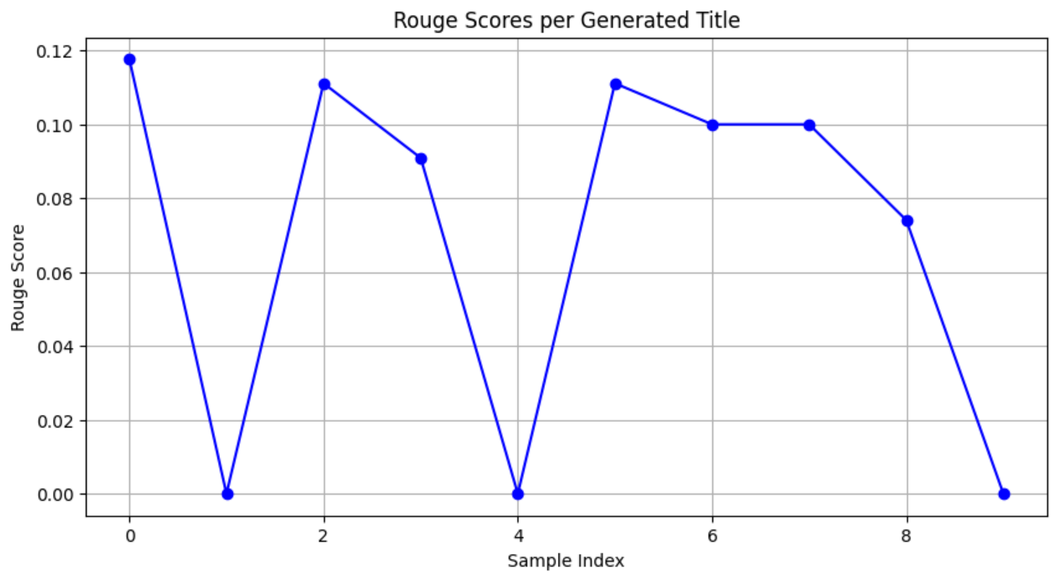
```
Title: calculation of prompt diphoton production cross sections at tevatron and lhc energies
Generated title: Applied thermodynamics and quantum chromodynamics
Title: sparsity certifying graph decompositions
Generated title: A new algorithm for the solving of X space-time-time logarithm Semiconductor Networks in the General Theory of
Relativity
Title: the evolution of the earth moon system based on the dark matter field fluid model
Generated title: Earth and the Nuclear-Adapted Universe
Title: a determinant of stirling cycle numbers counts unlabeled acyclic single source automata
Generated title: Attributing the Amount of True Curve to a Present Tract
Title: from dyadic lambda_ alpha to lambda_ alpha
Generated title: A Deductive Analysis of the Independent Properties of Area in Space - National Park Service
Title: bosonic characters of atomic cooper pairs across resonance
Generated title: A Synthesis of Molecular Stabilization Theory and the Properties of a Molecular Stabilization Theory
Title: polymer quantum mechanics and its continuum limit
Generated title: An axiom for the rivalling of the spin of the negative and positive part of a quark
Title: numerical solution of shock and ramp compression for general material properties
Generated title: Structural and Physical Static Variation in Liquid Metal Convection
Title: the spitzer c2d survey of large nearby interstellar clouds ix the serpens yso population as observed with irac and mips
Generated title: Standardizing the Mips C2d Spitzer Data Set (Seismic Data
Title: partial cubes structures characterizations and constructions
Generated title: Analysis of subgraphs of multiple subgraphs in the structure of the split section of a subgraph
]: plt.figure(figsize=(10, 5))
   plt.plot(bleu_scores, marker='o', linestyle='-', color='b')
   plt.title('BLEU Scores per Generated Title')
   plt.xlabel('Sample Index')
   plt.ylabel('BLEU Score')
   plt.grid(True)
   plt.show()
```



BLEU score is a evaluation metric used for quantifying the quality of the generated title. It measures the similarity between the generated title and the actual title based on the overlap of n-grams (contiguous sequences of n words) between them.

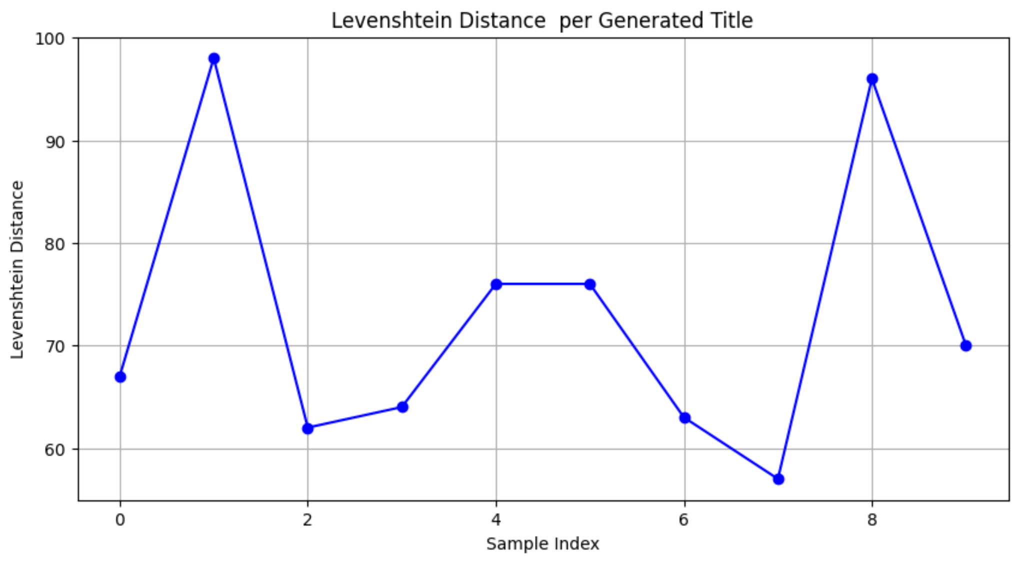
Higher BLEU signifies more similarity between the actual and generated title.

```
]: plt.figure(figsize=(10, 5))
   plt.plot(rouge_scores, marker='o', linestyle='-', color='b')
   plt.title('Rouge Scores per Generated Title')
   plt.xlabel('Sample Index')
   plt.ylabel('Rouge Score')
   plt.grid(True)
   plt.show()
```



ROUGE -Recall-Oriented Understudy for Gisting Evaluation score, It measures the overlap and similarity between the generated title and the actual title based on word overlap, n-gram overlap, and longest common subsequences. It is similar to BLEU score, higher value signifies more similarity between actual and generated title.

```
5]: plt.figure(figsize=(10, 5))
    plt.plot(edit_distances, marker='o', linestyle='-', color='b')
    plt.title('Levenshtein Distance  per Generated Title')
    plt.xlabel('Sample Index')
    plt.ylabel('Levenshtein Distance')
    plt.grid(True)
    plt.show()
```



Levenshtein Distance is the edit distance. It is the minimum number of single character edits required to change the generated text to the actual text. Lower Levenshtein Distance signifies high similarity between the generated and actual text.

Observation:

Among the three T5 models used, the mt5 model's performance is lower than the other two models. MT5 model's training loss seems to be higher than the other two models. ByT5 model performed well in terms of training loss.

Bonus: Real World deep learning Application:

Our project aims to save time for researchers, by helping them to formulate a relevant and precise title for their research. Our model tries to save the time spent in deciding the research title before publishing the article.

Team Member	Project Part	Contribution
Rohith Dinakaran	Dataset visualization, Mt5 model, Documentation	33.3%
Sankeerth Sridhar Narayan	T5, ByT5 model building and training, Documentation	33.3%
Ssneha Balasubramanium	Dataset preprocessing, GPT2 model	33.3%

References:

1. Assignment 1 Submission for UB CSE 676-B Spring 24, Surya Kumaran Sainath (UBIT: sainath), Sankeerth Sridhar Narayan (UBIT: ss676)
2. Assignment 0 Submission for UB CSE 676-B Spring 24, Sankeerth Sridhar Narayan (UBIT: ss676)
3. https://huggingface.co/docs/transformers/en/model_doc/t5
4. https://medium.com/geekculture/simplet5-train-t5-models-in-just-3-lines-of-code-by-shivanand-roy-2021-354df5ae46ba

Project By:

Rohith Dinakaran
Sankeerth Sridhar Narayan
Ssneha Balasubramanium