A Project Report

On

ABSTRACTIVE TEXT SUMMARIZATION

BY

RUDRARAJU ASRITH VARMA(SE20UCSE150)

Under the supervision of

RAGHU KISHORE NEELISETTI

SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF

SE 422: PROJECT TYPE COURSE

MAHINDRA ECOLE CENTRALE

COLLEGE OF ENGINEERING

HYDERABAD

(JUNE 2023)

ACKNOWLEDGMENTS

We are very much thankful to the **MAHINDRA UNIVERSITY** for giving us the opportunity to undertake our Third Year Project. We would also like to take the opportunity to thank and express our deep sense of gratitude to **Shri Dr. Arya Kumar Bhattacharya**, HOD of Computer Science Engineering.

We would like to express our deepest appreciation to **Prof. Raghu Kishore Neelisetti**, Assistant Professor in the Department of Computer Science Engineering. His unwavering support, invaluable guidance, constructive suggestions, and positive and encouraging demeanour throughout all stages of the project have been instrumental in its successful completion. Without his contributions, this project would not have been possible.

We hope that we can build upon the experience and knowledge that we have gained during the project and make it valuable for our coming future.

Mahindra Ecole Centrale

Hyderabad

CERTIFICATE

This is to certify that the project report entitled "**Abstractive Text Summarization**" submitted by **Rudraraju Asrith Varma(SE20UCSE150)** in partial fulfillment of the requirements of the course SE421/SE422, Project Course, embodies the work done by him/her under my supervision and guidance.

(Raghu Kishore Neelisetti & Signature)

Mahindra Ecole Centrale, Hyderabad.

Date:

(EXTERNAL NAME & Signature)

Mahindra Ecole Centrale, Hyderabad.

Date:

ABSTRACT

Abstractive text summarization involves generating a concise summary of a given text by understanding its meaning and generating new phrases, sentences, or even paragraphs that capture the essence of the original content. This approach differs from extractive text summarization, which involves selecting and rearranging existing sentences or phrases from the source text to create a summary.

This project aims to enhance the movie-watching experience by providing personalized movie summaries based on the user's viewing history. It utilizes abstractive text summarization, employing the fine-tuned PEGASUS model developed by Google with the IMDB dataset. By considering the user's ten previous movies, the summary for their next movie choice is customized to their preferences, incorporating elements from their past viewing experiences. This personalized approach improves decision-making and ensures the user receives summaries aligned with their interests.

The impact of this project extends to over-the-top (OTT) platforms and push notifications. OTT platforms, such as streaming services, can implement the personalized movie summaries to deliver more relevant and engaging content recommendations to their users. Moreover, the project's implementation in push notifications enables timely and personalized movie recommendations, by utilizing abstractive text summarization, push notifications can deliver concise and captivating summaries directly to the user's device, encouraging them to explore new movies that align with their interests. This not only improves the user's engagement with the platform but also enhances the effectiveness of push notifications in driving user actions.

The project's experimental evaluation demonstrates the effectiveness of generating personalized movie summaries that reflect user preferences using abstractive text summarization techniques. By leveraging deep learning and the pre-trained PEGASUS model, this project highlights the potential of customized movie recommendations. This has a positive impact on the OTT industry and enhances the effectiveness of push notifications, leading to improved user satisfaction, engagement, and more effective content recommendations. Overall, the project significantly enhances the personalized movie-watching experience, and its impact extends to OTT platforms and push notifications.

CONTENTS

1. INTRODUCTION

In today's digital age, where an overwhelming amount of information is available at our fingertips, movie enthusiasts often face the daunting task of choosing the perfect movie to watch. To assist users in making informed decisions, movie summaries play a crucial role. However, existing summaries often fail to capture the individual preferences and interests of each user, resulting in a generic and impersonalized movie-watching experience.

Summarization, the process of condensing a piece of text while preserving its key information, holds great significance in addressing this challenge. There are two primary types of summarization techniques: abstractive and extractive.

Abstractive summarization involves generating concise summaries by interpreting and synthesizing the source text, often using natural language generation techniques. This method allows for the creation of new sentences that capture the essence of the original content, even if the exact wording may differ. On the other hand, extractive summarization involves selecting and assembling important sentences or phrases directly from the source text to form a summary.

In the context of personalized movie summarization, our project focuses on the utilization of abstractive text summarization techniques. This approach allows us to generate personalized movie summaries by understanding the user's viewing history and creating summaries that reflect their unique preferences and interests. By leveraging deep learning and the pre-trained PEGASUS model, we aim to provide movie enthusiasts with summaries that go beyond a mere extraction of sentences and encapsulate the essence of the movies in a more engaging and tailored manner.

One of the significant advantages of abstractive summarization over extractive summarization is its ability to generate novel and coherent summaries. Unlike extractive methods that rely solely on existing sentences, abstractive summarization can generate new sentences that effectively capture the key information while maintaining the overall coherence of the summary. This allows for a more comprehensive representation of the movie's themes, plot points, and emotional aspects, resulting in summaries that are not only concise but also captivating.

Current movie summaries often fall short in terms of engagement, as they tend to be generic and lack the personalized touch that resonates with individual viewers. These summaries often follow a cookie-cutter template, providing a brief overview of the plot without capturing the unique aspects that make a movie compelling. Moreover, they frequently fail to convey the emotional depth and impact of a film, focusing solely on the plot and missing out on the emotional journey and thematic elements. Additionally, current summaries may provide an incomplete representation of a movie, overlooking crucial elements such as character development, underlying messages, or stylistic aspects. This limited representation can result in users missing out on movies that align with their specific preferences and interests. Furthermore, the lack of personalization in these summaries prevents users from connecting with the content, as they do not reflect their unique cinematic tastes. As a result, users may end up watching movies that do not meet their expectations, leading to disappointment and disengagement.

By harnessing the power of abstractive text summarization, our project aims to revolutionize the movie-watching experience by providing users with personalized movie summaries which are inspired by top rated reviews of other users as we're using IMDB dataset which contain all the reviews of a particular movie listed by genre, to fine-tune the PEGASUS model. These summaries are tailored to each user's viewing history, incorporating their unique preferences and interests. This personalized approach ensures that users receive summaries that resonate with their cinematic tastes, ultimately leading to a more satisfying and immersive movie-watching journey.

The real-world application of our project extends to the realm of Over-the-Top (OTT) platforms, such as popular streaming services. These platforms constantly strive to enhance user satisfaction and engagement by delivering personalized content recommendations. By incorporating personalized movie summaries into their recommendation systems, OTT platforms can further refine the user experience, offering summaries that are precisely aligned with each user's individual preferences and viewing habits.

Additionally, our project has implications for push notifications, which play a vital role in delivering timely updates and recommendations to users. By leveraging abstractive text summarization techniques, push notifications can provide concise and captivating movie summaries directly to users' devices, driving their interest and encouraging them to explore new movies that align with their unique tastes.

Through the integration of abstractive text summarization and deep learning techniques, our project strives to provide innovative solutions to the prevalent issue of generic and impersonalized movie summaries. By empowering users with personalized and captivating movie summaries, we aim to enhance the movie-watching experience, foster user satisfaction, and facilitate the discovery of cinematic gems tailored precisely to their preferences and interests.

2. PROBLEM STATEMENT

Our project aims to address the following key questions related to personalized movie summarization:

1. **How can we create movie summaries that are tailored to an individual's viewing history?**
    - By leveraging deep learning and abstractive text summarization techniques, our project seeks to generate personalized movie summaries that incorporate the essence of a user's previous movie-watching experiences by fine-tuning the PEGASUS model with the top-rated reviews of the movies present in the history based on genre. We have chosen only top-rated reviews because those are the ones which are already approved/loved by many people, now the model will only focus on these reviews and produces more engaging reviews tailored to each user's history.
2. **Can we enhance the decision-making process by providing summaries that align with the user's preferences?**
    - Through the utilization of the pre-trained PEGASUS model and fine-tuning it with the IMDB dataset, our project aims to deliver movie summaries that capture the user's interests and align with their viewing preferences, ultimately empowering them to make more informed movie choices.
3. **How can we improve user engagement and satisfaction with OTT platforms and push notifications?**
    - By implementing personalized movie summaries within OTT platforms and push notifications, our project seeks to enhance user engagement and satisfaction by delivering relevant and captivating summaries that pique their interest, driving them to explore new movies and stay connected with the platform.
4. **How can abstractive text summarization techniques revolutionize the movie-watching experience?**
    - Our project showcases the potential of abstractive text summarization in revolutionizing the movie-watching experience by offering customized and immersive summaries. By incorporating deep learning and the user's viewing history, we aim to provide a unique and tailored movie discovery process that aligns with the individual's preferences and interests.

Through our project, we aspire to provide innovative solutions to these questions, enabling users to discover movies that resonate with their personal tastes, thereby enhancing their overall movie-watching experience.

3. BACKGROUND AND RELATED WORK

There are several approaches to abstractive text summarization, the following are the research papers which we have referred to before starting the implementation as suggested by our project mentor. The summaries of each research paper are mentioned below,

Abstractive Sentence Summarization with Attentive Recurrent Neural Networks (2016):

This paper by Sumit Chopra, Michael Auli, and Alexander Rush proposed a new approach to abstractive text summarization that used an attentive recurrent neural network (RNN) model. The model was trained on a dataset of human-written summaries and was able to generate summaries that were both concise and fluent.

The attentive RNN model was composed of two main parts: an encoder and a decoder. The encoder was responsible for reading the input text and generating a sequence of hidden states. The decoder was then responsible for generating the summary, one word at a time.

The attention mechanism was used to allow the decoder to focus on specific parts of the input text when generating the summary. This was done by calculating a score for each word in the input text, and then using these scores to weight the hidden states from the encoder.

The model was trained using a maximum likelihood objective. This meant that the model was trained to generate summaries that were as similar as possible to the human-written summaries in the training dataset.

The model was evaluated on a test dataset of human-written summaries. The results showed that the model was able to generate summaries that were both concise and fluent. The model was also able to outperform previous state-of-the-art methods for abstractive text summarization. The major issue with this approach was the inability of RNN models to learn long-range dependencies and allowing parallel computation, this problem is solved by Transformer architecture as discussed in the below research paper.

Attention is all you need (2017):

This paper by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin introduced the transformer architecture, which is now a standard approach for abstractive text summarization. The transformer architecture is a neural network architecture that uses attention to learn long-range dependencies in the input text. Attention is a mechanism that allows the model to focus on specific parts of the input text when generating the summary.

The transformer architecture is composed of two main parts: the encoder and the decoder. The encoder is responsible for reading the input text and generating a sequence of hidden states. The decoder is then responsible for generating the summary, one word at a time.

The attention mechanism is used to allow the decoder to focus on specific parts of the input text when generating the summary. This is done by calculating a score for each word in the input text, and then using these scores to weight the hidden states from the encoder.

The transformer architecture was trained using a masked language modelling objective. This meant that the model was trained to predict the next word in a sequence, given the previous words in the sequence.

The transformer architecture was evaluated on several natural language processing tasks, including machine translation, text summarization, and question answering. The results showed that the transformer architecture was able to outperform previous state-of-the-art methods on all these tasks.

The paper concludes by summarising the contributions of the Transformer model and highlighting its advantages over traditional recurrent and convolutional neural networks. The authors suggest that attention mechanisms can be a powerful alternative for sequence transduction tasks in terms of parallelising computation and learning long-range dependencies.

PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization (2020):

This paper by Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu proposed a new method for pre-training transformer models for abstractive text summarization. The method involves first pre-training a transformer model on a large corpus of text. Then, the model is fine-tuned on a dataset of human-written summaries. The fine-tuning process involves filling in gaps in the summaries. This helps the model to learn to generate summaries that are more fluent and informative.

The PEGASUS model was pre-trained on a dataset of 500 billion words. The model was trained using a masked language modelling objective. This meant that the model was trained to predict the next word in a sequence, given the previous words in the sequence.

The PEGASUS model was fine-tuned on a dataset of 1 million human-written summaries. The fine-tuning process involved filling in gaps in the summaries. This helped the model to learn to generate summaries that were more fluent and informative.

The PEGASUS model was evaluated on a test dataset of human-written summaries. The results showed that the PEGASUS model was able to generate summaries that were more fluent and informative than previous state-of-the-art methods for abstractive text summarization.

The field of natural language processing (NLP) has witnessed remarkable advancements in recent years, with the introduction of transformer-based architectures marking a significant milestone. Transformers, which were first introduced in 2017, and have revolutionized NLP tasks by capturing global dependencies in an efficient and parallelizable manner. However, the concept of pre-training

transformers is relatively new, and models like PEGASUS, introduced in 2020, have further pushed the boundaries of abstractive text summarization.

The availability of resources and research on pre-trained transformers and abstractive summarization is relatively limited compared to other deep learning and NLP techniques. As these technologies continue to evolve rapidly, research papers serve as invaluable resources for understanding the underlying principles and approaches.

By delving into these research papers, we have gained a comprehensive understanding of the advancements and strengths of abstractive summarization models. While these research papers do not directly address our problem of personalized movie summarization based on user viewing history, they provide valuable insights into the advancements and strengths of abstractive summarization models. The Transformer architecture, with its self-attention mechanisms and parallelizable computations, has shown remarkable performance in various NLP tasks, making it an ideal choice for our personalized movie summarization project. Furthermore, the PEGASUS model's pre-training and fine-tuning framework has demonstrated its potential for enhancing abstractive summarization performance (as transformers in general perform very well on sequence-to-sequence[seq-to-seq] tasks like text translation, summarization, sentence completion and so on).

While the relative novelty of pre-trained transformers and abstractive summarization presents challenges in terms of available resources, it also opens exciting opportunities for innovation and exploration in this emerging field. By leveraging the knowledge gained from these research papers, we aim to contribute to the application of abstractive text summarization in the context of personalized movie recommendations based on user viewing history. Finally, we have decided to use IMDB dataset which nearly 1 million unique movie reviews from 1150 different IMDB movies, for fine-tuning the PEGASUS model (as it produced state-of-the-art results as per the above research paper in abstractive text summarization tasks).

4. IMPLEMENTATION

a. IMDb movie reviews dataset and data preprocessing:

To implement personalized movie summarization based on user viewing history, we utilized the IMDb Movie Reviews dataset, which can be accessed from the provided link (https://ieee-dataport.org/open-access/imdb-movie-reviews-dataset). This dataset consists of nearly 1 million unique movie reviews from 1150 different IMDb movies spread across 17 IMDb genres, including Action, Adventure, Animation, Biography, Comedy, Crime, Drama, Fantasy, History, Horror, Music, Mystery, Romance, Sci-Fi, Sport, Thriller, and War. Each movie entry in the dataset also contains additional metadata such as the date of release, run length, IMDb rating, movie rating (PG-13, R, etc.), number of IMDb raters, and the number of reviews per movie.

We have chosen Python as the programming language for this project as it is a popular choice for NLP tasks because it has a wide range of libraries that make it easy to perform common NLP tasks. These libraries provide pre-trained models and algorithms that can be used to perform these tasks, which saves time and effort. One of the most popular libraries for NLP in Python is Transformers, which is part of the Hugging Face library. Transformers provides several pre-trained language models that can be used for a variety of NLP tasks, such as text summarization, question answering, and natural language generation. We have used Transformers library to import PEGASUS model in our code.

Firstly, we imported the necessary libraries, including **pandas as pd, numpy as np, os, re, and glob**. Then, initially we have assumed that the user's history only has movies from Romance Genre. We read the movie reviews specific to the Romance genre from the provided dataset using the **pd.read_csv** function. This resulted in a D**ata Frame named romance_df** containing information about the Romance movies.

To create a list of romance movie names, we initialized an empty Data Frame named **romance_movie_names**. We combined the 'name' column and 'year' column of romance_df into a single string using the pd.concat function and applied the lambda function to join the two columns with a space separator. **The resulting Data Frame contained movie names along with their release years.**

Next, we created an empty Data Frame named **combined_reviews_romance with columns 'Movie', 'Reviews', and 'Helpful_Reviews'** to store the combined reviews and helpful reviews for each movie in the Romance genre i.e., we have dropped all other unnecessary columns other than those mentioned above.

For each movie name in romance_movie_names, we constructed the path to the corresponding CSV file containing the movie reviews. We checked if the file exists using the **os.path.isfile** function. If the file exists, we loaded the CSV file using **pd.read_csv** and dropped unnecessary columns such as 'username', 'rating', 'total', 'date', and 'title' using the **drop function** with the axis=1 argument.

To combine all the reviews for a movie into a single string, we used the ' '. join function on the 'review' column of the reviews Data Frame, resulting in the variable all_reviews. We also sorted the reviews based on their helpfulness scores in descending order using the sort_values function and stored the top 5 reviews in the variable top_5_reviews by joining them with a period ('.').

For the remaining reviews, we extracted them using slicing (sorted_reviews.iloc[5:] ['review']) and concatenated them into a single string called remaining_reviews_concatenated using the ' '. join function.

To create a new Data Frame with the current movie's data, we initialized a new Data Frame named new_df with columns 'Movie', 'Reviews', and 'Helpful_Reviews' and assigned the corresponding values. We then appended this new Data Frame to combined_reviews_romance using the pd.concat function with ignore_index=True.

After processing all the movies in the Romance genre, we dropped the 'Movie' column from combined_reviews_romance using the drop function with the axis=1 argument, as it is no longer needed.

After all these now we have a Data Frame named **combined_reviews_romance** which has two columns named '**Reviews' and 'Helpful_Reviews',** the 'Helpful_Reviews' column contains top-5 most rated reviews of each movie in each row of it from the Romance genre, the 'Reviews' column on the other hand has all the reviews except the top-5 most rated reviews of each movie in each row of it from the Romance genre. This is the final Data Frame which we're going to use for fine-tuning the PEGASUS model, where '**Reviews' is independent variable/input variable(x) and 'Helpful_Reviews' is dependent variable/output variable(y).**

Now, to clean the text in the 'Reviews' and 'Helpful_Reviews' columns, we used regular expressions to remove unwanted characters such as **'
', '$', '@', '&', and '?'** from the strings using the **str.replace function using the 're' library**.

b. Fine-tuning the PEGASUS-xsum model:

```
class PegasusDataset(torch.utils.data.Dataset):
  def __init__(self, encodings, decodings):
    self.encodings = encodings
    self.decodings = decodings

  def __getitem__(self, idx):
    item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
    item['labels'] = torch.tensor(self.decodings['input_ids'][idx])
    return item

  def __len__(self):
    return len(self.encodings['input_ids'])


def prepare_data(model_name,
                 train_texts, train_labels,
                 val_texts=None, val_labels=None,
                 test_texts=None, test_labels=None):
 """
 Prepare input data for model fine-tuning
 """
 tokenizer = PegasusTokenizer.from_pretrained(model_name)

 prepare_val = False if val_texts is None or val_labels is None else True
 prepare_test = False if test_texts is None or test_labels is None else True

 def tokenize_data(texts, labels):
  encodings = tokenizer(texts, truncation=True, padding=True)
  decodings = tokenizer(labels, truncation=True, padding=True)
  dataset_tokenized = PegasusDataset(encodings, decodings)
  return dataset_tokenized

 train_dataset = tokenize_data(train_texts, train_labels)
 val_dataset = tokenize_data(val_texts, val_labels) if prepare_val else None
 test_dataset = tokenize_data(test_texts, test_labels) if prepare_test else None

 return train_dataset, val_dataset, test_dataset, tokenizer


def prepare_fine_tuning(model_name, tokenizer, train_dataset,
val_dataset=None, freeze_encoder=False, output_dir='./results'):
 """
 Prepare configurations and base model for fine-tuning
```

```
  """
  torch_device = 'cuda' if torch.cuda.is_available() else 'cpu'
  model =
PegasusForConditionalGeneration.from_pretrained(model_name).to(torch_devic
e)

  if freeze_encoder:
    for param in model.model.encoder.parameters():
      param.requires_grad = False

  if val_dataset is not None:
    training_args = TrainingArguments(
      output_dir=output_dir,          # output directory
      num_train_epochs=100,          # total number of training epochs
      per_device_train_batch_size=1,   # batch size per device during training, can
increase if memory allows
      per_device_eval_batch_size=1,    # batch size for evaluation, can increase if
memory allows
      save_steps=500,               # number of updates steps before checkpoint saves
      save_total_limit=5,           # limit the total amount of checkpoints and deletes
the older checkpoints
      evaluation_strategy='steps',     # evaluation strategy to adopt during training
      eval_steps=100,               # number of update steps before evaluation
      warmup_steps=500,             # number of warmup steps for learning rate
scheduler
      weight_decay=0.01,            # strength of weight decay
      logging_dir='./logs',         # directory for storing logs
      logging_steps=10,
    )

    trainer = Trainer(
      model=model,                     # the instantiated □ Transformers model to be
trained
      args=training_args,              # training arguments, defined above
      train_dataset=train_dataset,        # training dataset
      eval_dataset=val_dataset,          # evaluation dataset
      tokenizer=tokenizer
    )

  else:
    training_args = TrainingArguments(
      output_dir=output_dir,          # output directory
```

```
    num_train_epochs=100,         # total number of training epochs
    per_device_train_batch_size=1,   # batch size per device during training, can
increase if memory allows
    save_steps=500,               # number of updates steps before checkpoint saves
    save_total_limit=5,           # limit the total amount of checkpoints and deletes
the older checkpoints
    warmup_steps=500,             # number of warmup steps for learning rate
scheduler
    weight_decay=0.01,            # strength of weight decay
    logging_dir='./logs',         # directory for storing logs
    logging_steps=10,
  )

  trainer = Trainer(
    model=model,                  # the instantiated 🤗 Transformers model to be
trained
    args=training_args,           # training arguments, defined above
    train_dataset=train_dataset,        # training dataset
    tokenizer=tokenizer
  )

 return trainer


# Extract the input and output columns from the dataframe
train_texts = combined_reviews_romance["Reviews"]
train_labels = combined_reviews_romance["Helpful_Reviews"]

# Specify the model name and prepare the data
model_name = 'google/pegasus-xsum'
train_dataset, _, _, tokenizer = prepare_data(model_name, train_texts.tolist(),
train_labels.tolist())

# Prepare the fine-tuning trainer
trainer = prepare_fine_tuning(model_name, tokenizer, train_dataset)
trainer.train()
```

We have used the **NVIDIA DGX-1 Supercomputer** available in our college to fine-tune large language models such as Pegasus-XSum. Since these models are very compute-intensive and cannot be run on personal computers. The DGX-1 is a powerful Supercomputer that is specifically designed for deep learning tasks. It has 8 GPUs that can be used to train and fine-tune large language models. The DGX-1 has allowed us to train and fine-tune Pegasus-XSum on a massive dataset of text and code. We would like to thank our project mentor Prof. Raghu Kishore Neelisetti, Dr. Arya Kumar Bhattacharya, for granting us permission to use the DGX-1. We would also like to thank the CSE department at our college for providing us with access to this valuable resource.

Now let us explain the code above. Firstly, we imported the required libraries for fine-tuning the PEGASUS-xsum model, including PegasusForConditionalGeneration, PegasusTokenizer, Trainer, TrainingArguments, transformers, torch, torch.utils.data.Dataset, and sklearn.model_selection.train_test_split.

To fine-tune the PEGASUS model, we defined a custom dataset class named **PegasusDataset,** which inherits from **torch.utils.data.Dataset**. This class takes in **encodings and decodings as inputs** and implements the necessary methods such as **__getitem__ and __len__**.

We defined a **helper function named prepare_data** that takes in the **model's name, training texts, training labels, validation texts, validation labels, test texts, and test labels as input parameters**. This function prepares the input data for model fine-tuning by initializing the tokenizer and tokenizing the data using the **PegasusTokenizer**. I**t returns the train dataset, validation dataset, test dataset, and the tokenizer.**

To prepare the model and configurations for fine-tuning, we defined the **helper function prepare_fine_tuning.** This function takes in the **model's name, tokenizer, train dataset, validation dataset, freeze_encoder, and output directory as input parameters**. Inside this function, we determined the torch device to be used based on the availability of CUDA. We instantiated the **PegasusForConditionalGeneration model** and moved it to the torch device.

If the validation dataset is provided, we initialized the TrainingArguments for fine-tuning with the appropriate settings such as the output directory, number of training epochs, batch sizes, evaluation strategy, warm-up steps, and logging directory. We then instantiated the Trainer class with the model, training arguments, train dataset, validation dataset, and tokenizer.

If no validation dataset is provided, we initialized the TrainingArguments with the same settings but excluded the evaluation related parameters. We instantiated the Trainer class with the model, training arguments, train dataset, and tokenizer. **The function returns the trainer object.**

To prepare the data for fine-tuning, we extracted the **input ('Reviews') and output columns ('Helpful_Reviews') from the combined_reviews_romance Data Frame**. The

input texts were stored in the **train_texts** variable, and the output texts were stored in the **train_labels** variable.

We specified the **Pegasus model name as 'google/pegasus-xsum'** and called the **prepare_data function**, passing the **model's name, train texts, and train labels as input parameters to obtain the train dataset, tokenizer, and other required datasets as the output.**

Next, we called the **prepare_fine_tuning function**, passing the **model's name, tokenizer, train dataset, and other optional parameters as input parameters**. This prepared the fine-tuning trainer with the specified configurations.

The trainer was then trained using the trainer.train() function, which initiated the fine-tuning process on the PEGASUS model using the provided train dataset.

Finally, the fine-tuned model was **saved in the output directory** specified as '/home/u20010/Sem_Project/fine_tune_1' using the **trainer.save_model() function**. Next time when we need to load this fine-tuned model, we can load it directly from the specified output directory instead of training the model again and again (it took 62 minutes for this to run successfully on DGX-1 Supercomputer)

c. Testing the fine-tuned model:

We loaded the fine-tuned model from the saved directory using the PegasusForConditionalGeneration.from_pretrained() function and assigned it to the model_1 variable.

For testing purposes, we provided a sample input text which is the most rated review in the movie '3 Idiots'. We encoded the input text using the tokenizer and obtained the input IDs. Then, we used the fine-tuned model to generate a summary by calling **model_1.generate()** with the encoded input IDs. The generated summary IDs were decoded using the tokenizer, skipping special tokens, and stored in the summary variable.

Input summary to the fine-tuned model:

"Awesome film man. Saw this with my family in a theatre. Worth every penny n worth every minute. Never seen such an amazing film. It's about the pursuit of happiness. What is so great about the movie is that it moves you, it gives you hope. It is a simple film, yet it has an everlasting message. The last 30 mins were beautifully shot. Very good direction by Hirani. The star cast is good n Aamir khan steals the show. This guy is the best thing to happen to Bollywood. The editing is top notch. U don't feel bored for even a sec. Awesome screenplay. It's hands down the best film in the history of Bollywood."

The output generated summary of the fine-tuned model:

"Saw this at a special screening last night. The movie is indeed one of the most moving and touching movies I have seen in my life. The message that this film is dealing with is something that we should all take note of. Happiness is a fact that we should all take note of and use it to live."

5. RESULTS AND VALIDATION

To evaluate our model's result generated above we have used two validation techniques,

a. **<u>To check abstractive accuracy of the generated summary</u>:**

   To know if the generated summary is abstractive enough or not, we have used a simple technique. We have written a code in python that calculates the word similarity between two paragraphs. It first converts the paragraphs to lowercase and splits them into individual words. Using the Counter class from the collection's module, it counts the frequency of each word in both paragraphs. It then identifies the common words between the two paragraphs, excluding words with a length of 2 or less. The code calculates the percentage of similar words compared to the total unique words in both paragraphs, as well as the percentage of non-similar words. The common words are printed, and a pie chart is created to visualize the word similarity percentages. The chart displays the percentage of similar words and non-similar words, with the similar words slice exploded for emphasis. The resulting pie chart provides a visual representation of the word similarity between the two paragraphs.

   By this method we'll get similar words percentage in both the summaries i.e., the input and the generated summaries. The less the similarity percentage the better the model in terms of abstractive accuracy as we're not using the same words again from the input summary in the model generated summary i.e., contrary to extractive text summarization where the model has to generate a summary with most of the words taken from the input and making it shorter and preserving the overall meaning and context. Whereas in abstractive summarization we intend to use new words or phrases in our model generated summary so that it preserves the overall meaning and context while using new words or phrases.

   The code we've written in python for this task is,

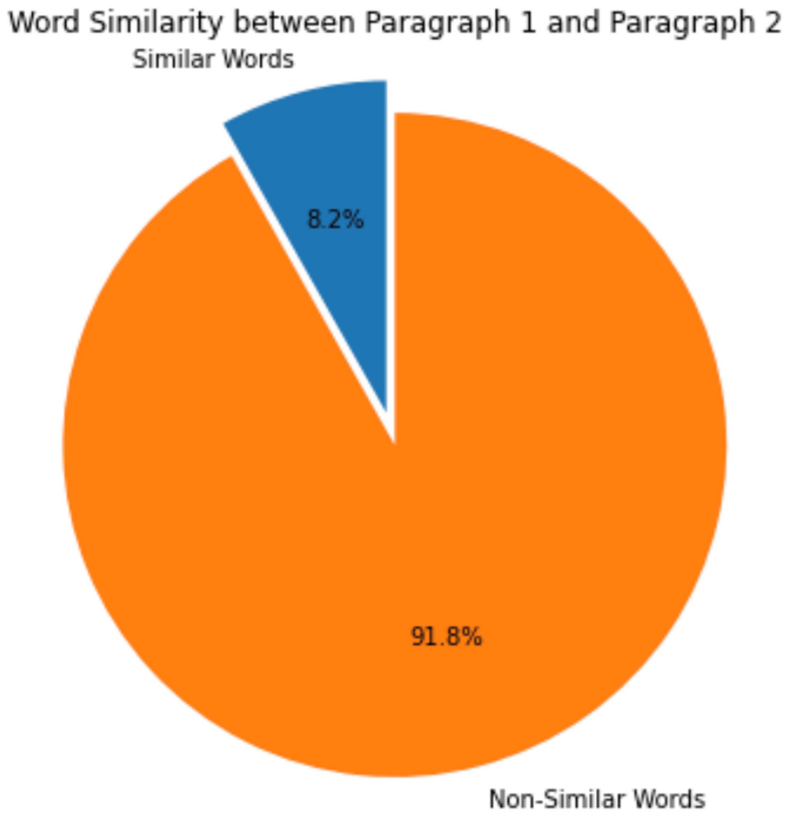
```
import matplotlib.pyplot as plt
from collections import Counter
```

```
def calculate_word_similarity(paragraph1, paragraph2):
    # Convert both paragraphs to lowercase
    paragraph1 = paragraph1.lower()
    paragraph2 = paragraph2.lower()

    # Split paragraphs into words
    words1 = paragraph1.split()
    words2 = paragraph2.split()

    # Count the frequency of each word in both paragraphs
    word_counts1 = Counter(words1)
    word_counts2 = Counter(words2)

    # Get the common words (case-insensitive)
    common_words = set(word_counts1.keys()).intersection(set(word_counts2.keys()))

    # Filter out words with length <= 2
    common_words = [word for word in common_words if len(word) > 2]

    # Count the number of similar words
    similar_words_count = len(common_words)

    # Calculate the percentages
    total_words_count = len(set(words1) | set(words2))
    similar_percentage = (similar_words_count / total_words_count) * 100
    non_similar_percentage = 100 - similar_percentage

    return similar_percentage, non_similar_percentage, common_words

# Example usage
paragraph1 = "Awesome film man. Saw this with my family in a theatre. Worth every
penny n worth every minute. Never seen such an amazing film.Its about the pursuit of
happiness. What is so great about the movie is that it moves you, it gives you hope. It is a
simple film, yet it has an everlasting message. The last 30 mins were beautifully shot.
Very good direction by Hirani. The star cast is good n Aamir khan steals the show. This
guy is the best thing to happen to Bollywood. The editing is top notch. U don't feel bore
for even a sec. Awesome screenplay. It's hands down the best film in the history of
Bollywood."
paragraph2 = "Saw this at a special screening last night. The movie is indeed one of the
most moving and touching movies I have seen in my life. The message that this film is
dealing with is something that we should all take note of. Happiness is a fact that we
should all take note of and use it to live"

similar_percentage, non_similar_percentage, common_words =
calculate_word_similarity(paragraph1, paragraph2)

# Display the similar words
print("Similar words:")
for word in common_words:
    print(word)
```

```
# Create the pie chart
labels = ['Similar Words', 'Non-Similar Words']
sizes = [similar_percentage, non_similar_percentage]
colors = ['#1f77b4', '#ff7f0e']
explode = (0.1, 0)  # Explode the first slice

plt.figure(figsize=(6, 6))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%',
startangle=90)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle
plt.title('Word Similarity between Paragraph 1 and Paragraph 2')
plt.show()
```

Output of the code:

Similar words: with, movie, saw, this, last, that, the, film, seen



So, out of total unique words in input and model generated summary only 8.2% of the words are similar, out of those similar ones there are only 2 to 3 nouns rest all are either pronouns or verbs which are inevitable to not repeat in most of the cases.

Therefore, our model is doing well in terms of abstractive accuracy.

b. <u>To check how close the generated summary is to the people's expectations</u>:

Now to know if the model generated summary is relevant or not, we have conducted an online survey using Microsoft forms where we asked the people to read both the surveys the input one and the model generated one, then we have asked them two important questions of Yes/No answer type. These are the statistics of the answers,

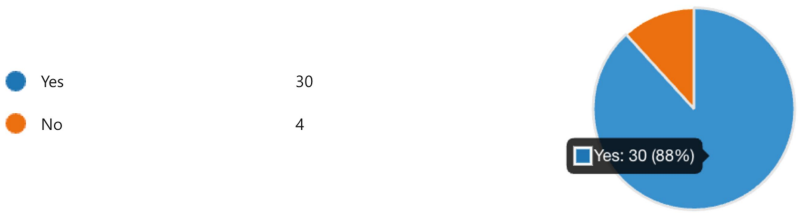
1. Read both the summaries that are given below.

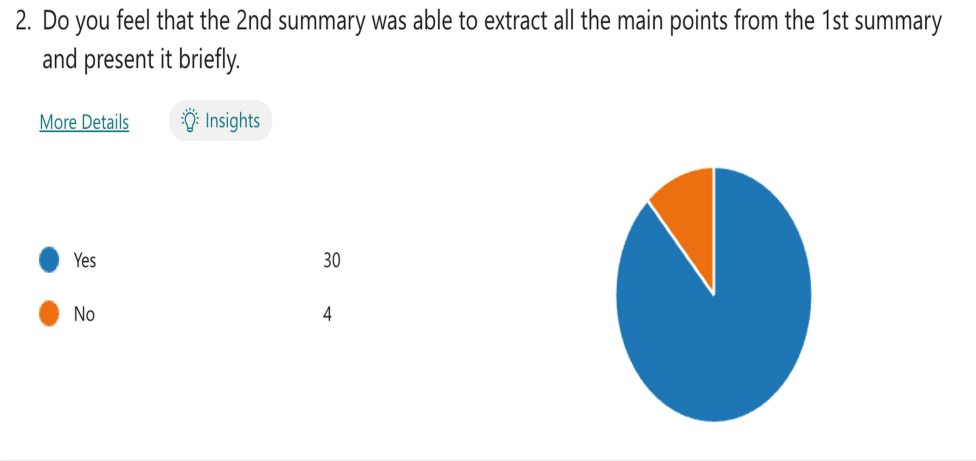
   1. Awesome film man. Saw this with my family in a theatre. Worth every penny n worth every minute. Never seen such an amazing film. Its about the pursuit of happiness. What is so great about the movie is that it moves you, it gives you hope. It is a simple film, yet it has an everlasting message. The last 30 mins were beautifully shot. Very good direction by Hirani. The star cast is good n Aamir khan steals the show. This guy is the best thing to happen to Bollywood. The editing is top notch. U don't feel bore for even a sec. Awesome screenplay. It's hands down the best film in the history of Bollywood.

   2. Saw this at a special screening last night. The movie is indeed one of the most moving and touching movies I have seen in my life. The message that this film is dealing with is something that we should all take note of. Happiness is a fact that we should all take note of and use it to live.

   Do you feel that the context is same for both the above summaries?

   More Details





88% of the people who have taken the survey feel that the summary generated by the model is satisfactory and is able to extract the context of the original input summary as well.

6. CONCLUSION

In conclusion, the abstractive summarization project aimed to generate concise and meaningful summaries using advanced natural language processing techniques. Through the utilization of models like Pegasus-XSum and leveraging the power of NVIDIA DGX-1 supercomputer, we were able to fine-tune large-scale models that are computationally intensive to run on personal computers. This enabled us to generate abstractive summaries that captured the essence of the original text in a more human-like manner.

While the results of the project were promising as per the validation techniques, and yielded satisfactory summaries, there is still a scope for improvement. The evaluation of abstractive summaries' accuracy remains a challenging task, as it involves subjective judgment and understanding of the original content. Looking ahead, we are excited about the prospects of this project. We intend to continue exploring advancements in language models, fine-tuning techniques, and evaluation methodologies to further enhance the abstractive summarization process. By staying at the forefront of research and incorporating user feedback, we aim to develop more accurate and coherent summaries that cater to specific domains and user preferences.

Ultimately, our goal is to contribute to the field of natural language processing by developing abstractive summarization techniques that can assist in information extraction, document understanding, and content summarization across various domains. The potential impact of this

project in streamlining information access and improving efficiency is a driving force behind our continued dedication to its development.

7. REFERENCES

[1] Ashish Vaswani-Google Brain, Naam Shazeer-Google Brain, Niki Parmar-Google Research, Jakob Uszkoreit-Google Research, Llion Jones-Google Research, Aidan N.Gomez-University of Toronto, Lukasz Kaiser-Google Brain, Illia Polosukhin. “Attention Is All You Need”.

[2] Alex Graves, “Generating sequences with recurrent neural networks.” arXiv preprint arXiv: 1308.0850, 2013.

[3] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. “Effective approaches to attention-based neural machine translation”. arXiv preprint arXiv: 1508.04025, 2015.

[4] Romain Paulus, Caiming Xiong, and Richard Socher. “A deep reinforced model for abstractive summarization”. arXiv preprint arXiv: 1705.04304, 2017.

[5] Sumit Chopra-Facebook AI Research, Micheal Auli-Facebook AI Research, Alexander M.Rush-Harvard SEAS. “Abstractive Sentence Summarization with Attentive Recurrent Neural Networks”, 2016.

[6] Jingqing Zhang, Yao Zhao, Mohammad Saleh, Peter J. Liu. “PEGASUS - Pre-training with Extracted Gap-sentences for Abstractive Summarization”, 2020.

[7] Goodman, S., Lan, Z., and Soricut, R. “Multi-stage pretrain-ing for abstractive summarization”, 2019.

[8] “IMDb Movie Reviews Dataset”.

Link: https://ieee-dataport.org/open-access/imdb-movie-reviews-dataset

[9] “Hugging Face Google/pegasus-xsum model”.

Link: https://huggingface.co/google/pegasus-xsum/tree/main

[10] “Pytorch script for fine-tuning Pegasus Large model”

Link: https://discuss.huggingface.co/t/fine-tuning-pegasus/1433