TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A Major Project Report

On

SlideIt
GENERATING VIDEO PRESENTATION FROM ARTICLES

Submitted By:

AAGAT POKHREL (PUL075BCT002)

ANISH AGARWAL (PUL075CT012)

BIPIN PURI (PUL075BCT023)

BIRAJ BIKRAM PATHAK (PUL075BCT024)

A PROJECT WAS SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE BACHELOR'S DEGREE IN COMPUTER ENGINEERING

(May 1, 2023)

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

A Major Project Report

On

SlideIt
GENERATING VIDEO PRESENTATION FROM ARTICLES

Submitted By:

AAGAT POKHREL (PUL075BCT002)

ANISH AGARWAL (PUL075CT012)

BIPIN PURI (PUL075BCT023)

BIRAJ BIKRAM PATHAK (PUL075BCT024)

Submitted To:

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

LALITPUR, NEPAL

(May 1, 2023)

PAGE OF APPROVAL

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING

The undersigned certifies that they have read and recommended to the Institute of Engineering for acceptance of a project report entitled **”Video Presentation from Articles”** submitted by Aagat Pokhrel, Anish Agrawal, Bipin Puri and Biraj Bikram Pathak in partial fulfillment of the requirements for the Bachelor’s degree in Electronics and Computer Engineering.

....	....
Supervisor	Internal examiner
Dr. Sanjeeb Prasad Panday	
Associate Professor	
Department of Electronics and	
Computer Engineering,	
Pulchowk Campus, IOE, TU	

....

External examiner

Date of approval:

COPYRIGHT

The author has agreed that the Library, Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering may make this report freely available for inspection. Moreover, the author has agreed that permission for extensive copying of this project report for scholarly purposes may be granted by the supervisors who supervised the project work recorded herein or, in their absence, by the Head of the Department wherein the project report was done. It is understood that the recognition will be given to the author of this report and to the Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering in any use of the material of this project report. Copying or publication or the other use of this report for financial gain without approval of to the Department of Electronics and Computer Engineering, Pulchowk Campus, Institute of Engineering and author's written permission is prohibited.
Request for permission to copy or to make any other use of the material in this report in whole or in part should be addressed to:

Head of Department
Department of Electronics and Computer Engineering
Pulchowk Campus, Institute of Engineering, TU
Lalitpur,Nepal.

ACKNOWLEDGEMENTS

Working with a variety of people whose thoughts and ideas have directly or indirectly aided or motivated us has been a wonderful pleasure. We'd want to express our gratitude to everyone who helped us finish this project, which was truly an academic experience.

We'd like to express our gratitude to our supervisor **Associate Professor Dr. Sanjeeb Prasad Panday** for giving us the information, invaluable suggestions, and assistance we needed to complete this project.

We would also like to acknowledge the **Department of Electronics and Computer Engineering** for their collaborative efforts and coordination in the successful completion of this project. Additionally, we express our heartfelt appreciation to our teachers and instructors for their guidance and support, which helped us to overcome various challenges we faced throughout the project.

Sincerely,
Aagat Pokhrel
Anish Agrawal
Bipin Puri
Biraj Bikram Pathak

ABSTRACT

This project proposes a method for generating video slide presentations from text articles. The proposed method involves text parsing, feature extraction, clustering, ranking, summarization, slide creation, speech synthesis, and video generation. The BART model finetuned on dataset is used for feature extraction and abstractive summarization. The K Medoids clustering algorithm is used for clustering the sentence features, and the KNN algorithm is used for ranking. The markdown syntax and MARP are used for slide creation, and the Azure cognitive speech services and FFMPEG are used for speech synthesis and video generation, respectively. The comparision is drawn among BART large and BART base models on the CNN/DM dataset. The results show that the BART-large model outperforms in terms of ROUGE scores and employing the further pipeline generates coherent and informative video slide presentations.

Keywords: Video slide presentation, Text articles, BART, CNN-DM dataset, K Medoids clustering, KNN algorithm, MARP, Azure cognitive speech services, FFMPEG.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

BART Bidirectional and Auto-Regressive Transformer. v, 19, 20, 49, 61, 63, 68

BERT Bidirectional Encoder Representations from Transformers. 5

CNN/DM CNN/Daily Mail dataset. 49, 61

FFMPEG Fast Forward MPEG. v, 58, 68, 69

KNN K-Nearest Neighbors. v

MARP Markdown Presentation Ecosystem. v, 2, 57, 68, 69

NLP Natural Language Processing. 2, 7

TTS Text-To-Speech. 39

1. INTRODUCTION

In recent years, the generation of video slide presentations has gained significant attention due to the increasing demand for online education and knowledge sharing. However, the manual creation of video slide presentations from text articles is a time-consuming and tedious task. Therefore, there is a need for an automated method.

Overall, our proposed method provides an efficient and automated way to generate video slide presentations from input articles, which can benefit various fields such as education, journalism, and research.

1.1. Background

The generation of video slide presentations from text articles has been studied in various research fields such as natural language processing, deep learning [1], and multimedia. Traditional methods for generating video slide presentations involve manual creation using software such as Microsoft PowerPoint or Adobe Premiere Pro. However, these methods are time-consuming and require a significant amount of effort.

To address this issue, several automated methods have been proposed, such as the use of summarization techniques and text-to-speech synthesis. Summarization techniques involve extracting the most important information from the input text and presenting it in a concise format. Text-to-speech synthesis involves converting the summarized text into speech, which can then be used to generate the audio for the video slide presentation.

Recently, deep learning-based approaches have shown promising results in generating video slide presentations from input text. These approaches use advanced language models such as BERT and GPT to extract features from the input text and generate abstractive summaries. However, these models require a large amount of training data and computational resources, which can be challenging to obtain for some applications.

1.2. Objectives

The main objective of this project is to propose an efficient and automated method for generating video slide presentations from input articles. Specifically, we aim to:

- Develop a method for feature extraction from input articles.
- Employ clustering and ranking algorithms to group and rank the extracted features.
- Perform abstractive summarization on each cluster using the BART model.
- Create slides using Markdown syntax and MARP.
- Generate audio using Azure Cognitive Speech Services and concatenate it with the slides using ffmpeg to generate a video.

1.3. Problem Statement

The manual creation of video slide presentations from input articles is a time-consuming and tedious task. Therefore, there is a need for an automated method that can generate video slide presentations efficiently and accurately.

Existing automated methods for generating video slide presentations from input articles have limitations such as low accuracy and poor quality of summaries. Therefore, there is a need for a method that can generate high-quality summaries and accurately present the information in the input articles.

1.4. Scope of Project

The scope of project involves developing an automated method to generate video slide presentations from input articles. The goal is to address the time-consuming and tedious process of manually creating video presentations. The project aims to utilize natural language processing NLP algorithms to analyze the content of input articles and extract relevant information using machine learning techniques. The system will incorporate computer vision algorithms to generate appropriate visuals that match the content of the input articles. A user-friendly interface will be designed and developed for users to upload articles and receive video slide presentations. The accuracy and quality of the generated video presentations will be evaluated using objective and subjective metrics. User testing will be conducted to ensure that the platform meets the needs and expectations of its target users. The system will be optimized for performance, enabling it to generate video slide presentations quickly and efficiently. Finally, the necessary documentation and support will be provided to users for effective use and integration of the platform into their workflow.

2. LITERATURE REVIEW

Recent advances in natural language processing and deep learning have led to the development of various techniques for text summarization, including extractive and abstractive methods. Similary the work on audio processing and speech synthesis through neural models have made it possible for human like speech.

2.1. Related Theory

2.1.1. Summarization Tasks

The history of automatic summarization dates back to the 1950s, when researchers began to explore the idea of using computers to automatically generate summaries of news articles. In the early days, summarization systems were rule-based and relied on hand-crafted rules to identify important sentences and extract key phrases.

One of the earliest summarization systems was the SLIP (Selective Logging and Integrated Processing) system, which was developed in the 1960s by Edmund Berkeley. The SLIP system was designed to automatically summarize news articles by selecting the most important sentences and phrases, and was based on a set of hand-crafted rules.

In the 1970s and 1980s, researchers began to explore statistical approaches to summarization. One of the early statistical models was the TF-IDF (Term Frequency-Inverse Document Frequency) model, which was used to rank sentences based on their importance. The TF-IDF model assigns a weight to each word in a document based on its frequency in the document and its rarity in the corpus as a whole. Sentences are then ranked based on the sum of the weights of the words they contain.

In the 1990s and 2000s, there was a growing interest in machine learning approaches to summarization. One of the early machine learning models was the SumBasic algorithm, which was introduced by Vanderwende in 2007. The SumBasic algorithm is a simple algorithm that assigns a weight to each sentence based on the frequency of the words it contains, and then selects the top-ranked sentences to create the summary.

In recent years, deep learning models have been applied to summarization with great success. One of the earliest deep learning models for summarization was the sequence-to-sequence model, which was introduced by Sutskever et al. in 2014. The sequence-to-sequence model uses a neural network to map the input text to a fixed-length vector, and then uses another neural network to generate the summary from the vector.

Another popular deep learning model for summarization is the transformer model, which was introduced in a 2017 paper by Vaswani et al. The transformer model is based on the self-attention mechanism, which allows the model to attend to different parts of the input sequence and learn representations that capture long-range dependencies. The transformer model has been used for both extractive and abstractive summarization, and has achieved state-of-the-art performance on many benchmarks.

One of the most popular abstractive text summarization models is the Bidirectional Encoder Representations from Transformers BERT model. BERT [2] has been used for various NLP tasks, including text classification, named entity recognition, and text summarization. However, BERT requires large amounts of data and training time, which can be a limitation in some applications.

Another model that has been used for text summarization is the BART (Bidirectional and Auto-Regressive Transformers) model [3]. BART is a sequence-to-sequence model that uses a denoising autoencoder objective to pretrain on large amounts of text data, and has been fine-tuned on the CNN/DM dataset for text summarization [4].

2.1.2. Speech Synthesis

The history of speech processing can be traced back to the 1950s, when researchers first began to investigate the use of computers to analyze and synthesize speech. One of the earliest speech processing systems was the IBM Shoebox, which was developed in 1952. The IBM Shoebox was a large machine that could recognize and synthesize individual phonemes, the smallest units of sound in human speech.

In the 1960s and 1970s, researchers began to develop more sophisticated speech processing systems that could recognize entire words and sentences. One of the early systems was the Hearsay II system, which was developed at Carnegie Mellon University in the 1970s. The Hearsay II system used a hidden Markov model to recognize spoken words and was one of the first speech recognition systems to achieve high accuracy.

In the 1980s and 1990s, there was a growing interest in machine learning approaches to speech processing. One of the early machine learning models was the backpropagation algorithm, which was used to train neural networks to recognize speech. Neural networks are a type of machine learning algorithm that are modeled after the structure of the human brain.

In recent years, deep learning models have been applied to speech processing with great success. One of the most successful deep learning models is the convolutional neural network (CNN), which was first used for speech recognition by Sainath et al. in 2013. CNNs are a type of neural network that are designed to process data with a grid-like structure, such as images or spectrograms (which are representations of sound waves).

Another popular deep learning model for speech processing is the recurrent neural network (RNN), which is designed to process sequences of data, such as time series data or natural language sentences. RNNs are able to capture temporal dependencies in data, which makes them well-suited for tasks such as speech recognition and speech synthesis.

The history of speech synthesis can be traced back to the early 20th century, when researchers first began to investigate the use of electrical signals to generate speech sounds. In the 1930s, Homer Dudley developed the first speech synthesis machine, known as the Voder. The Voder was able to synthesize human-like speech sounds by using a series of filters to manipulate electrical signals.

In the 1950s and 1960s, researchers began to develop more sophisticated speech synthesis systems that could generate entire words and sentences. One of the early systems was the Pattern Playback, which was developed at Bell Labs in the 1950s. The Pattern Playback used a series of tape loops to generate different speech sounds and was one of the first speech synthesis systems to be used for practical applications.

In the 1970s and 1980s, there was a growing interest in formant synthesis, which is a type of speech synthesis that uses a set of parameters, known as formants, to generate speech sounds. One of the early formant synthesis systems was the Klatt synthesizer, which was developed by Dennis Klatt in the 1970s. The Klatt synthesizer used a set of rules to generate speech sounds based on the formant parameters.

In recent years, deep learning models have been applied to speech synthesis with great success. One of the most successful deep learning models is the WaveNet model, which was introduced by DeepMind in 2016. The WaveNet model is a type of generative model that uses autoregressive modeling to generate speech waveforms. Autoregressive modeling is a statistical modeling technique that is used to predict a future value based on past values. In the case of speech synthesis, the WaveNet model uses autoregressive modeling to predict the next sample in a speech waveform given the previous samples.

The WaveNet model has been shown to produce high-quality, natural-sounding speech, and has been used in a variety of applications, including text-to-speech systems, virtual assistants, and interactive voice response (IVR) systems. Other different models that uses neural vocoders are Azure Speech Services, HifiNet etc.

2.2. Related Work

2.2.1. Beautiful AI

There are several other platforms that use NLP and AI to generate presentations or slides from text. One such platform is Beautiful.AI, which uses natural language processing algorithms to analyze the text and generate a visually appealing slide deck. The platform allows users to choose from a variety of templates and design elements to create a professional-looking presentation.

2.2.2. Zuru

Another platform that uses NLP and AI to generate presentations is Zuru, which is designed specifically for sales and marketing teams. Zuru uses AI to analyze the text and generate a presentation that is tailored to the target audience. The platform also includes features such as audience analytics and performance tracking to help users optimize their presentations for maximum impact.

2.2.3. Lumen5

Other such platform is Lumen5, which allows users to turn written content into engaging videos using a closed-end AI engine. The platform uses natural language processing NLP algorithms to analyze the text and generate visuals that match the content. Lumen5 has been successful in creating high-quality videos quickly and efficiently, making it a popular choice for content creators.

2.2.4. SlidesAI

Another platform that is similar to our proposed architecture is SlidesAI, which uses language models to break down sentences and match them with relevant slides. SlidesAI relies on AI to analyze the content and select the most appropriate visuals to accompany the text. The platform has been successful in generating engaging and informative presentations in a short amount of time.

Another area of development is the use of machine learning algorithms to analyze the performance of slides and presentations, and provide feedback to users on how to optimize their content for maximum impact. This could include recommendations on things like slide design, messaging, and pacing.

3. THEORITICAL BACKGROUND

3.1. Neural Networks

Neural networks [5] are a type of machine learning algorithm that is modeled after the structure and function of the human brain. They are widely used in a variety of fields, including computer vision, natural language processing, and speech recognition. In this article, we will discuss neural networks in depth, including their history, architecture, training methods, and applications.

3.1.1. Architecture

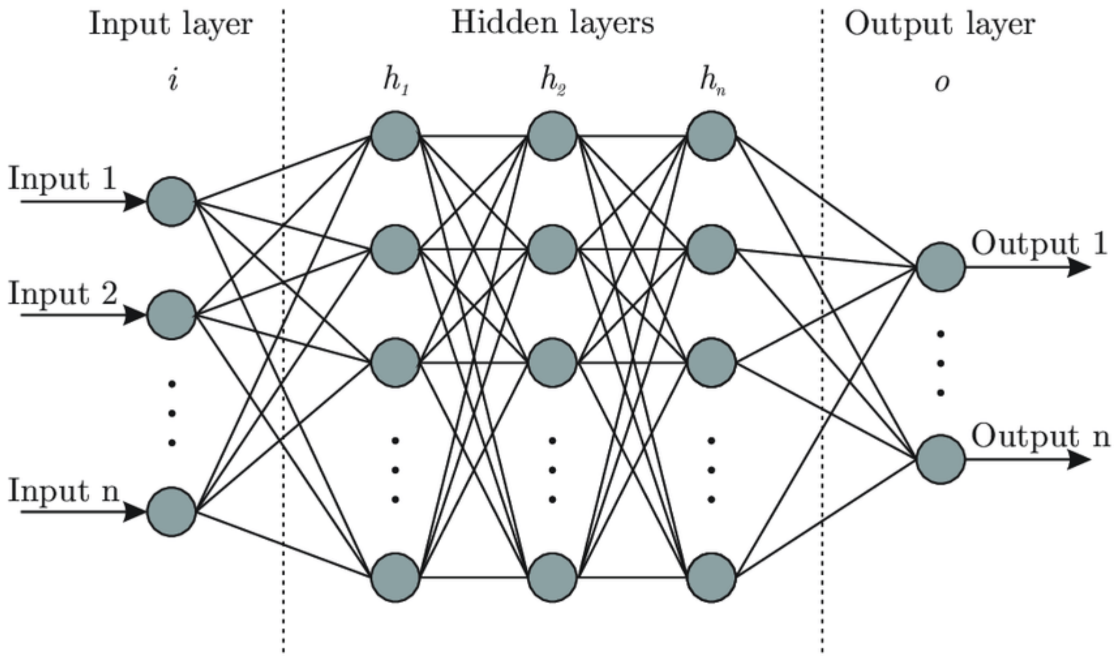


Figure 3.1: Neural Network

Image Source: Designing Neural Networks, Towards Data Science

A neural network consists of a series of connected layers of nodes, also known as neurons. Each neuron takes in inputs, processes them, and produces an output, which is passed on to the next layer of neurons. The layers in a neural network can be divided into three main types: input layers, hidden layers, and output layers.

The input layer is the first layer of neurons in a neural network, and it takes in the input data. The input data can be in the form of images, text, or numerical data. The input layer does not perform any calculations; it simply passes the input data on to the first hidden layer.

The hidden layers are the layers in between the input layer and the output layer. The number of hidden layers and the number of neurons in each hidden layer can vary depending on the complexity of the problem being solved. The neurons in the hidden layers perform calculations on the inputs they receive, and they pass their outputs on to the next layer of neurons.

The output layer is the final layer of neurons in a neural network, and it produces the output of the network. The output can be in the form of a classification (e.g., is this image a cat or a dog?) or a regression (e.g., what is the predicted price of this house?).

3.1.2. Training Methods

Neural networks are trained using a process called backpropagation, which involves adjusting the weights of the connections between neurons in the network. The goal of training a neural network is to adjust the weights in such a way that the network produces the correct output for a given input.

During training, the network is presented with a set of input/output pairs, also known as training examples. The network processes each input and produces an output, and the difference between the output and the desired output is calculated. This difference is known as the error, and it is used to adjust the weights in the network.

The optimization process can be done using a variety of techniques, including stochastic gradient descent (SGD), which updates the parameters based on the gradient computed for a random subset of the training data, and its variants such as Adam, Adagrad, and RMSprop. These optimization algorithms help the network to converge faster and avoid getting stuck in local minima.

3.1.3. Types

There are several types of neural networks, each designed for specific tasks and with different architectures. Some of the commonly used neural network types are:

- Feedforward Neural Networks: These networks are the simplest type of neural networks, consisting of a single input layer, one or more hidden layers, and an output layer. Each layer is composed of neurons that receive input from the previous layer and pass their output to the next layer, with no feedback connections. Feedforward neural networks are typically used for classification and regression tasks.

- Convolutional Neural Networks (CNNs): These networks are designed for processing data that has a grid-like structure, such as images or time series data. CNNs use a set of filters to extract features from the input data, and the output of each filter is passed through a non-linear activation function. CNNs are commonly used for image recognition, object detection, and natural language processing tasks.
- Recurrent Neural Networks (RNNs): These networks are designed for processing sequential data, such as time series or text. RNNs have feedback connections that allow the network to maintain an internal state and process inputs in a time-dependent manner. RNNs are commonly used for language modeling, speech recognition, and machine translation tasks.
- Long Short-Term Memory (LSTM) Networks: These networks are a type of RNN that can better handle long-term dependencies in sequential data. LSTMs use a memory cell that can selectively retain or forget information over time, allowing the network to remember important features of the input data for longer periods. LSTMs are commonly used for speech recognition, handwriting recognition, and language translation tasks.

3.2. Transformers

Transformers are a type of neural network architecture that was introduced in 2017 by Vaswani et al[6]. in the paper "Attention Is All You Need". Transformers have become a popular architecture in natural language processing (NLP) tasks, such as machine translation, text generation, and language understanding, due to their ability to handle long-range dependencies and parallel processing. Due to transformers many large language models are into existence [7].

The transformer architecture is based on a self-attention mechanism that allows the model to selectively attend to different parts of the input sequence, based on their relevance to the current context. This makes transformers more efficient and effective in modeling long-range dependencies than recurrent neural networks (RNNs), which can suffer from vanishing or exploding gradients.

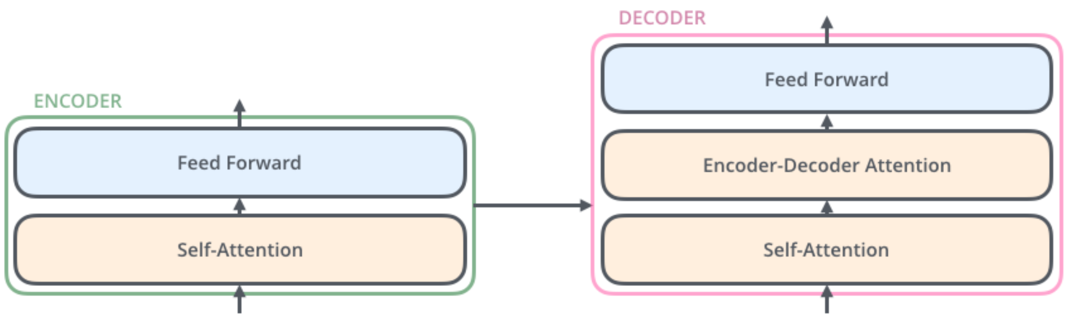


Figure 3.2: Transformer Component Architecture

Image Source: Illustrated Transformer, Jay Alamar

3.2.1. Architecture

The transformer architecture consists of two main components: the encoder and the decoder. Let's take a closer look at each of these components.

3.2.1.1 Encoder

The encoder is responsible for processing the input sequence and producing a sequence of hidden states that capture the semantic and syntactic information of the input. The encoder consists of a stack of N identical layers, each of which performs two main operations: self-attention and feedforward neural networks.

- **Self-Attention**: The self-attention mechanism is the core of the transformer architecture. It allows the model to selectively attend to different parts of the input sequence, based on their relevance to the current context. Self-attention computes a weighted sum of the input sequence, where the weights are learned during training.

  In the self-attention operation, each input element is transformed into three vectors: the query vector, the key vector, and the value vector. These vectors are computed using learned linear transformations of the input, which are shared across all positions in the sequence. The query vector represents the current position, while the key vectors represent all other positions. The value vectors represent the information that the model should attend to.

The weights for each position are computed by taking the dot product of the query vector with all the key vectors, scaled by the square root of the dimensionality of the key vectors. This dot product yields a scalar score, which is then passed through a softmax function to obtain a probability distribution over all positions in the sequence. The value vectors are then weighted by these probabilities and summed to obtain the output of the self-attention operation.

The self-attention operation is performed multiple times in parallel, with different learned weights, allowing the model to attend to different aspects of the input sequence at each layer.

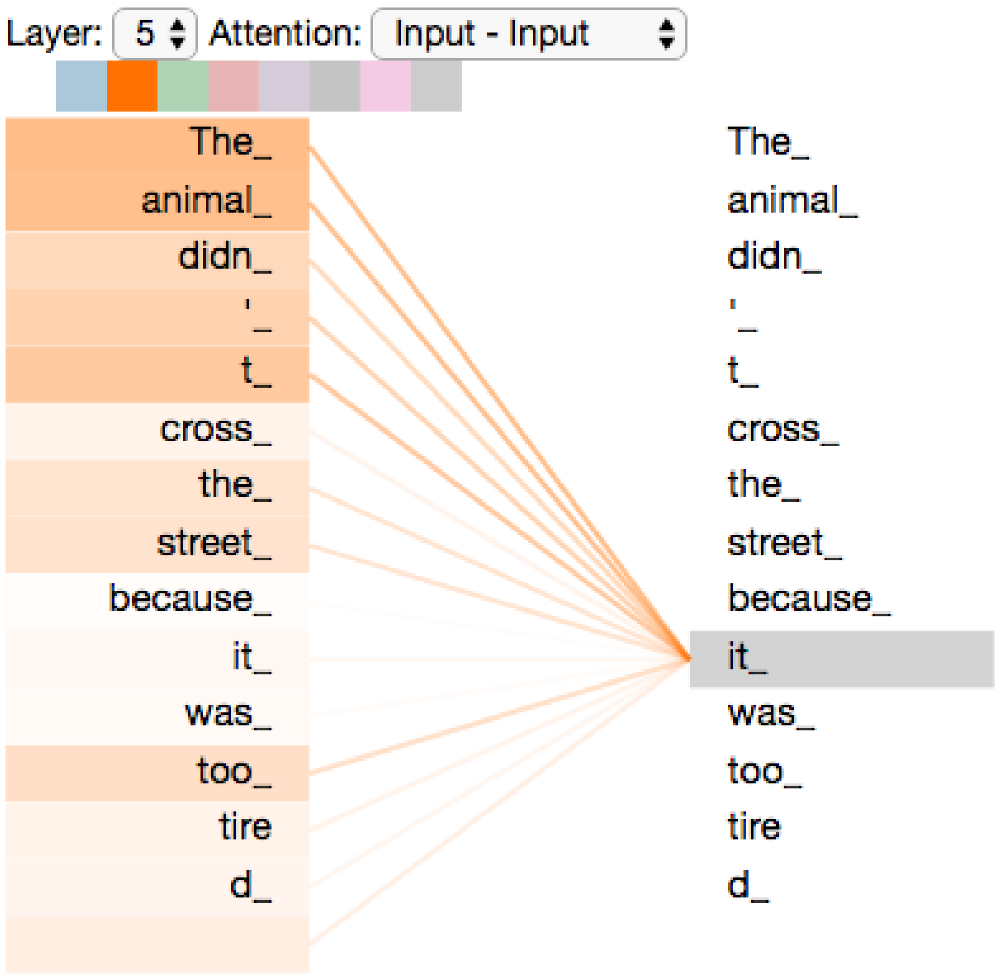


Figure 3.3: Self Attention Mechanism

Image Source: Illustrated Transformer, Jay Alamar

- **Feedforward Neural Networks**: After the self-attention operation, the encoder applies a feedforward neural network to each position in the sequence independently. The feedforward network consists of two linear transformations separated by a non-linear activation function, such as the Rectified Linear Unit (ReLU). The output of the feedforward network is then added to the output of the self-attention operation, using a residual connection, and normalized using layer normalization.

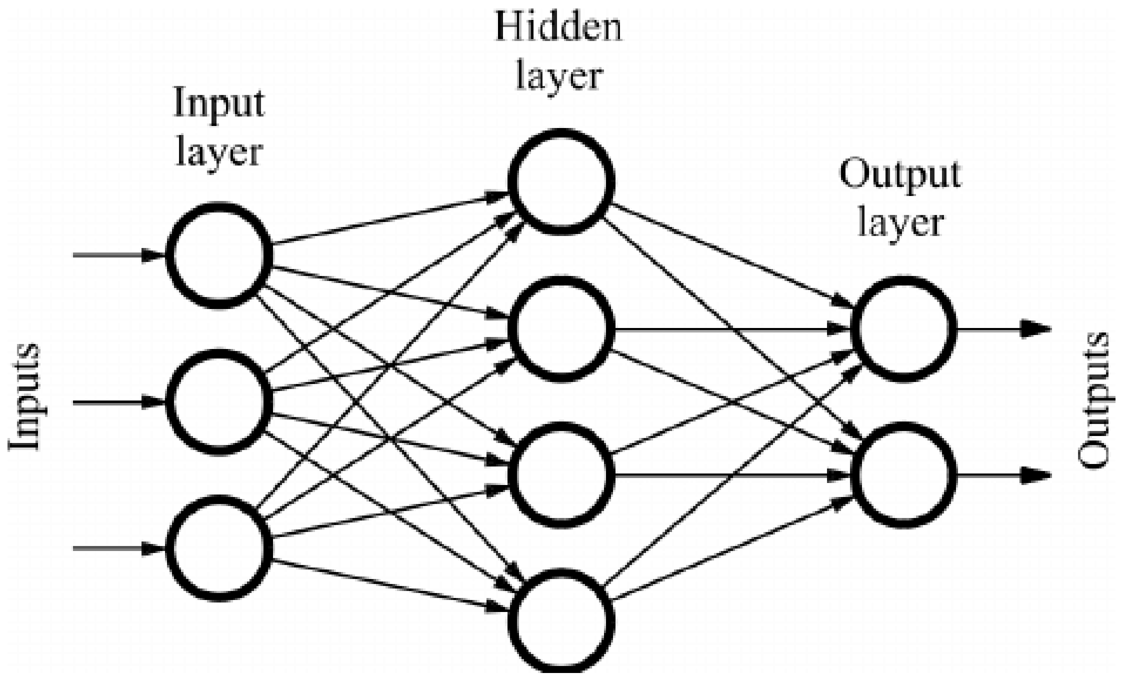


Figure 3.4: Feed Forward Network

Image Source: Towards Data Science, Feed Forward Network

3.2.1.2 Decoder

The decoder is responsible for generating the output sequence, given the hidden states produced by the encoder and a target sequence. The decoder consists of a stack of N identical layers, each of which performs three main operations: self-attention, encoder-decoder attention, and feedforward neural networks.

- **Self-Attention**: The self-attention mechanism in the decoder is similar to the one used in the encoder, except that it is masked to prevent the model from attending to future positions in the output sequence. This ensures that the model only has access to the previously generated tokens when generating the next token in the sequence.
-]**Encoder-Decoder Attention**: In addition to the self-attention operation, the decoder also applies an encoder-decoder attention operation to the hidden states produced by the encoder.

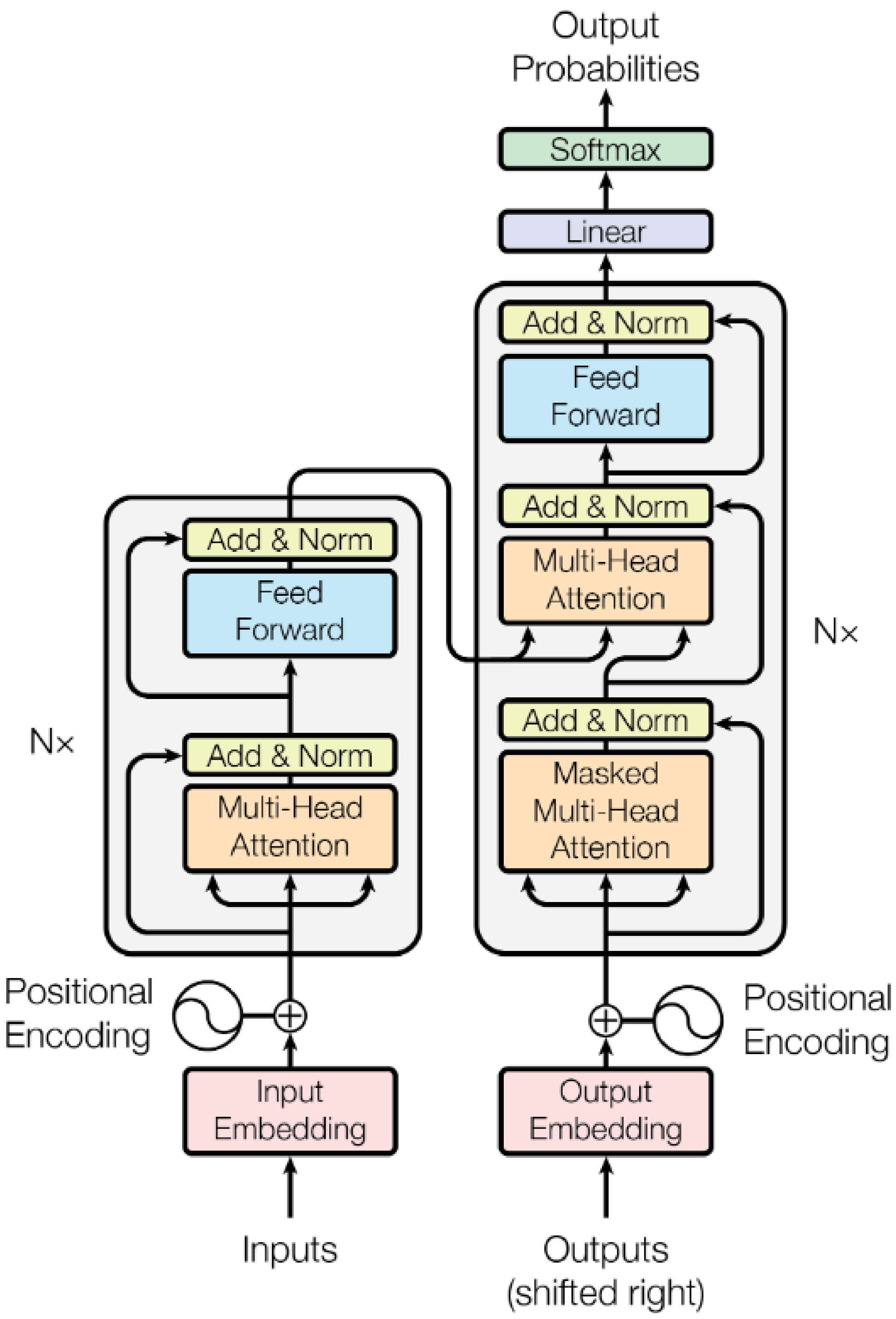


Figure 3.5: Transformer Architecture

Image Source: Transformer Explained, Vaswani Et. al

3.2.2. Training Process

The training process of the Transformer architecture involves optimizing a loss function that measures the discrepancy between the predicted output sequence and the ground truth sequence. This loss function is typically the cross-entropy loss, which measures the difference between the predicted probability distribution over the output vocabulary and the true distribution.

The training process of the Transformer architecture can be divided into two phases: pre-training and fine-tuning. In the pre-training phase, the model is trained on a large corpus of text data using an unsupervised learning objective, such as language modeling or masked language modeling. The pre-training objective is designed to encourage the model to learn useful representations of the input sequence, which can then be fine-tuned on downstream tasks.

In the fine-tuning phase, the pre-trained model is further trained on a specific downstream task using supervised learning. The fine-tuning process involves replacing the final layer(s) of the model with task-specific layers and optimizing the model parameters to minimize the task-specific loss.

3.3. Language Models

Language models[7] are an essential part of natural language processing (NLP) that have revolutionized the field in recent years. A language model is a type of artificial intelligence that can analyze, understand, and generate human language. It is a statistical model that learns the patterns and rules of a language by processing large amounts of text data. The model then uses this knowledge to make predictions about the probability of various sequences of words, given a specific context.

Language models are used for a wide range of tasks, including speech recognition, machine translation, question-answering systems, chatbots, and many others. They are essential for developing intelligent systems that can communicate with humans in a natural way.

There are two types of language models: statistical and neural. Statistical language models use traditional machine learning algorithms to identify patterns in language data.

They analyze the frequency of words and their relationships to each other to generate probability distributions that can be used to predict the likelihood of different sequences of words. These models are based on n-gram models, which are sequences of n words that are used to predict the next word in a sentence.

On the other hand, neural language models use deep learning algorithms to learn the patterns and rules of a language. They are based on artificial neural networks that are trained on large amounts of text data. Neural language models have become increasingly popular in recent years, as they can generate more accurate predictions than statistical models.

Neural language models use a type of neural network called a recurrent neural network (RNN). An RNN is a type of neural network that has a feedback loop, which allows it to store information from previous inputs. This feedback loop makes RNNs well-suited for modeling sequential data, such as language.

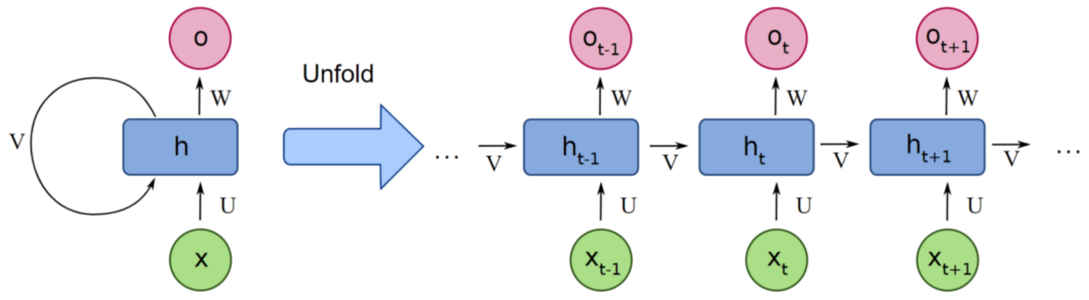


Figure 3.6: Recurrent Neural Network

Image Source: Analytics Vidya RNN

The most popular type of neural language model is the transformer model, which was introduced in a paper by Vaswani et al. in 2017. The transformer model is based on a self-attention mechanism that allows it to attend to different parts of a sentence, depending on the context. This mechanism allows the model to generate more accurate predictions than previous models.

Language models are trained on large amounts of text data, typically using a process called unsupervised learning. During training, the model is presented with a large corpus of text data and learns to predict the probability of the next word in a sentence given the previous words. The model is optimized to minimize the difference between its predicted probability and the actual probability of the next word.

The quality of a language model is determined by its ability to generate coherent and grammatically correct sentences. To evaluate the performance of a language model, researchers typically use metrics such as perplexity, which measures how well the model can predict the next word in a sentence.

One of the main challenges in developing language models is the problem of bias. Language models can learn biases from the text data they are trained on, which can lead to biased predictions. For example, a language model trained on text data that contains gender stereotypes may generate biased predictions when asked to complete a sentence about a person of a certain gender.

To address this problem, researchers have developed techniques such as debiasing, which involves removing biased language from the training data, and fine-tuning, which involves re-training the model on a smaller dataset that has been manually labeled to remove biases.

Another challenge in developing language models is the problem of language understanding. While language models are good at generating coherent sentences, they may not understand the meaning behind the words they are generating. This can lead to nonsensical or inappropriate responses in conversational systems.

To address this problem, researchers have developed techniques such as pre-training and transfer learning. Pre-training involves training a language model on a large amount of text data before fine-tuning it on a specific task. Transfer learning involves using a pre-trained language model as a foundation and performing a downstream task by finetuning the model.

3.4. BART Model

BART, or Bidirectional and Auto-Regressive Transformer, is a state-of-the-art neural network model for text generation and natural language processing tasks. It was introduced by Google AI Language and Facebook AI in 2019.BART is a transformer-based neural network architecture that is capable of generating high-quality text in a variety of natural language processing (NLP) tasks. The BART model was specifically designed to overcome some of the limitations of existing pre-trained language models such as BERT, GPT-2, and XLNet.

BART is a combination of two popular neural network architectures: transformers and sequence-to-sequence models. The transformer architecture [6] was introduced by Vaswani et al. in 2017, and is known for its ability to model long-term dependencies in sequential data. The sequence-to-sequence model was introduced by Sutskever et al. in 2014 [8], and is commonly used for tasks such as machine translation, text summarization, and question answering. BART training objective is to reconstruct the original input text, given a corrupted or incomplete version of the text. This training approach is known as denoising auto-encoding, and it allows the model to learn a deep understanding of the underlying structure of the text.

The BART model is a variant of the Transformer architecture that was specifically designed for language generation tasks such as summarization, paraphrasing, and text generation. The model consists of an encoder-decoder architecture, where the encoder encodes the input text and the decoder generates the output text. The encoder and decoder both consist of multiple layers of self-attention and feed-forward neural networks.

The BART model is unique in that it combines two types of training objectives: bidirectional and auto-regressive. The bidirectional objective involves training the model to predict missing tokens in a sentence by looking at both the left and right context. The auto-regressive objective involves training the model to generate the next token in a sequence based on the previous tokens. By combining these two objectives, the BART model is able to generate high-quality natural language text that is both coherent and informative.

The BART model consists of two parts: an encoder and a decoder. The encoder takes as input a sequence of tokens and encodes it into a sequence of hidden states. The decoder takes as input the encoded sequence and generates a new sequence of tokens. The encoder and decoder are trained together in a denoising autoencoder framework, where the model is trained to reconstruct the original sequence from a corrupted version of the sequence.

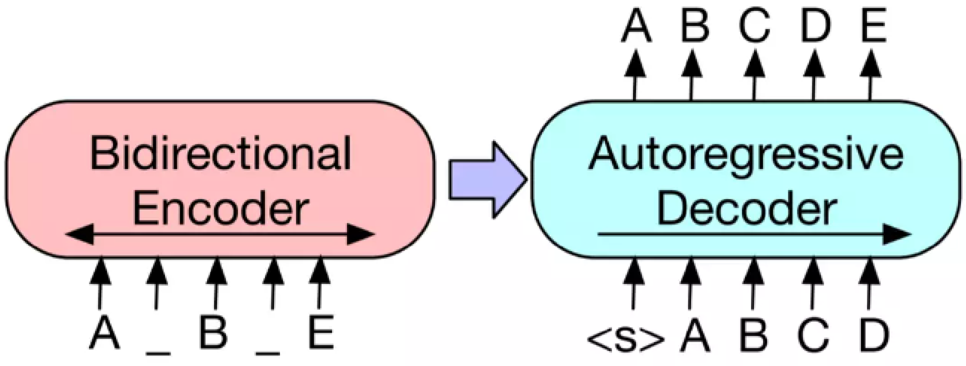


Figure 3.7: BART Architecture

Image Source: Project Pro, Transformers BART Model Explained for Text Summarization

One of the key advantages of the BART model is its ability to generate high-quality summaries of long documents. The model is able to capture the key information in the document and generate a concise summary that accurately reflects the content of the document.

Another advantage of the BART model is its ability to generate high-quality paraphrases of input text. The model is able to generate multiple different paraphrases of a given sentence, which can be useful in applications such as data augmentation and text simplification.

The BART model is trained on a large corpus of text data using a variant of the denoising autoencoder (DAE) framework. The training procedure consists of two main stages: pre-training and fine-tuning.

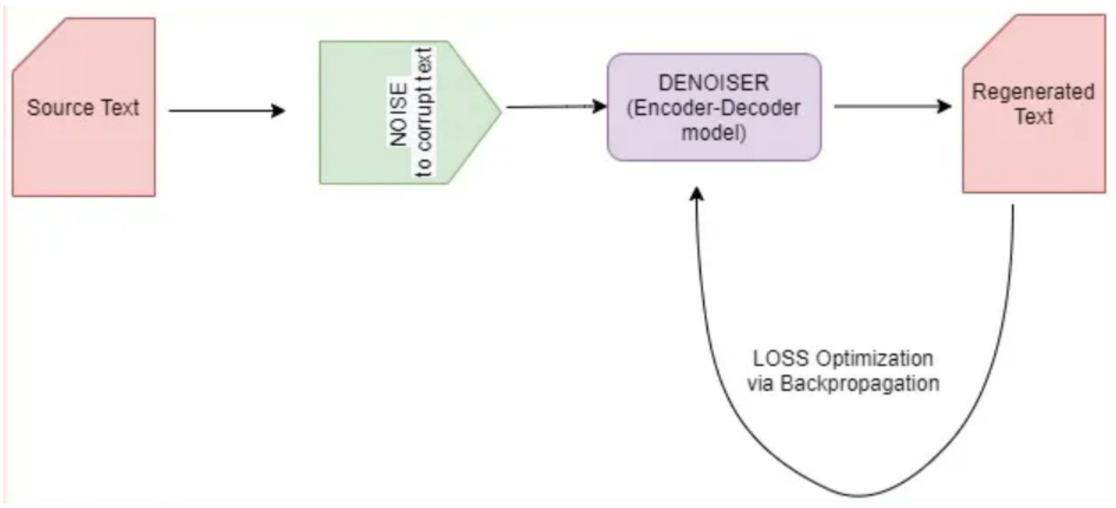


Figure 3.8: BART DAE training objective

Image Source: BART, A denoising objective for training

- During the pre-training stage, the BART model is trained on a large corpus of text data using an unsupervised training procedure. The goal of pre-training is to learn general-purpose language representations that can be used for a wide range of downstream NLP tasks. The pre-training procedure involves corrupting the input text by randomly masking tokens, shuffling them, and replacing them with random tokens. The model is then trained to reconstruct the original text from the corrupted version of the text.
- During the fine-tuning stage, the BART model is fine-tuned on a specific NLP task using a supervised training procedure. The fine-tuning procedure involves training the model on a smaller labeled dataset that is specific to the task at hand. The fine-tuning procedure allows the BART model to adapt its language representations to the specific requirements of the task and achieve state-of-the-art performance on the task.

BART was trained as a denoising autoencoder, so the training data includes “corrupted” or “noisy” text, which would be mapped to clean or original text. The training format is similar to the training of any denoising autoencoder. Just how in computer vision, we train autoencoders to remove noise or improve the quality of an image by having noisy images in the training data mapped with clean, original images as the target.

So, what exactly counts as noise for text data? The authors of BART settle on using some existing and some new noising techniques for pretraining. The noising schemes they use are Token Masking, Token Deletion, Text Infilling, Sentence Permutation, and Document Rotation. Looking into each of these transformations:

- Token Masking: Random tokens in a sentence are replaced with [MASK]. The model learns how to predict the single token based on the rest of the sequence.
- Token Deletion: Random tokens are deleted. The model must learn to predict the token content and find the position where the token was deleted from.
- Text Infilling: A fixed number of contiguous tokens are deleted and replaced with a single [MASK] token. The model must learn the content of the missing tokens and the number of tokens.
- Sentence Permutation: Sentences (separated by full stops) are permuted randomly. This helps the model to learn the logical entailment of sentences.
- Document Rotation: The document is rearranged to start with a random token. The content before the token is appended at the end of the document. This gives insights into how the document is typically arranged and how the beginning or ending of a document looks like..

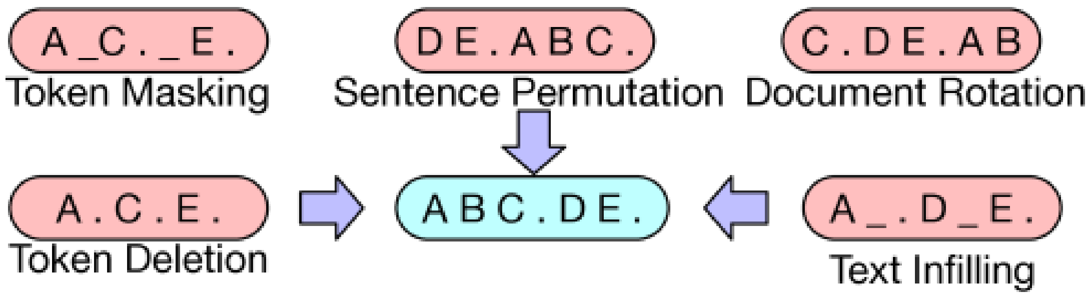


Figure 3.9: BART Steps

Image Source: BART white paper

However, not all transformations are employed in training the final BART model. Based on a comparative study of pre-training objectives, the authors use only text infilling and sentence permutation transformations, with about 30 percent of tokens being masked and all sentences permuted.

The BART model has been trained on a variety of large-scale text datasets, including the BookCorpus and English Wikipedia datasets. The model has also been fine-tuned on specific tasks such as summarization and question answering, achieving state-of-the-art performance on these tasks.

These transformations are applied to 160GB of text from the English Wikipedia and BookCorpus dataset. With this dataset, the vocabulary size is around 29000, and the maximum length of the sequences is 512 characters in the clean data.

The BART model has been applied to a wide range of NLP tasks, including text generation, summarization, question-answering, machine translation, and dialogue generation. The BART model has achieved state-of-the-art performance on many of these tasks, demonstrating the effectiveness of the model in capturing complex linguistic patterns and generating high-quality text.

One of the most notable applications of the BART model is in text summarization. The BART model has been shown to outperform other state-of-the-art models in abstractive summarization, which involves generating a summary that captures the key ideas and themes of a given text. BART has also been applied to other types of summarization tasks, such as multi-document summarization and summarization of long documents.

The BART model has also been applied to machine translation tasks, where it has achieved state-of-the-art performance in several languages. BART has been shown to be particularly effective in low-resource settings, where training data is scarce.

Another application of the BART model is in question-answering, where it has been used to generate answers to natural language questions. BART has also been applied to dialogue generation tasks, where it has been used to generate natural and engaging dialogue between humans and machines. The whole architecture can be briefly divided and explained into two parts bidirectional encoder and a left-to-right decoder:

3.4.1. Bidirectional encoder

The BART model is a pre-trained transformer-based neural network architecture that is capable of generating high-quality text in a variety of NLP tasks. The BART model consists of an encoder and a decoder, both of which contain multiple layers of self-attention and feed-forward neural networks. One of the most important components of the BART model is its bidirectional encoder, which allows the model to capture contextual information from both the left and the right of the input sequence.

Bidirectional encoding is a technique used in NLP to capture contextual information from both the left and the right of the input sequence. In traditional language models, such as the ones based on Markov chains or Hidden Markov Models (HMMs), the probability of the current word is based only on the previous words in the sequence. However, in natural language, the meaning of a word is often dependent on the words that come before and after it.

Bidirectional encoding addresses this problem by processing the input sequence in two directions: from left to right and from right to left. This allows the model to capture contextual information from both directions and incorporate it into the representation of each word. Bidirectional encoding has been shown to be effective in a variety of NLP tasks, including language modeling, machine translation, and named entity recognition.

The BART model uses a bidirectional encoder to capture contextual information from both directions of the input sequence. The encoder consists of multiple layers of self-attention and feed-forward neural networks. The self-attention mechanism allows the model to attend to different parts of the input sequence at different levels of granularity, while the feed-forward neural networks provide non-linear transformations of the input sequence.

In the BART model, the bidirectional encoder consists of a stack of N identical layers. Each layer has two sub-layers: a self-attention sub-layer and a feed-forward sub-layer. The self-attention sub-layer allows the model to capture contextual information from both directions by attending to the input sequence in a bidirectional manner. The feed-forward sub-layer applies a non-linear transformation to the output of the self-attention sub-layer.

The output of each layer in the bidirectional encoder is a sequence of hidden states, which are fed into the next layer. The final hidden states of the bidirectional encoder are used as input to the decoder, which generates the output sequence. The bidirectional encoder in the BART model allows the model to capture contextual information from both the left and the right of the input sequence, which is critical for many NLP tasks.

The first part of BART uses the bi-directional encoder of BERT to find the best representation of its input sequence. For every text sequence in its input, the BERT encoder outputs an embedding vector for each token in the sequence as well as an additional vector containing sentence-level information. In this way, the decoder can learn for both token and sentence-level tasks making it a robust starting point for any future fine-tuning tasks. The pre-training is done using the masked sequences as discussed previously and shown below.

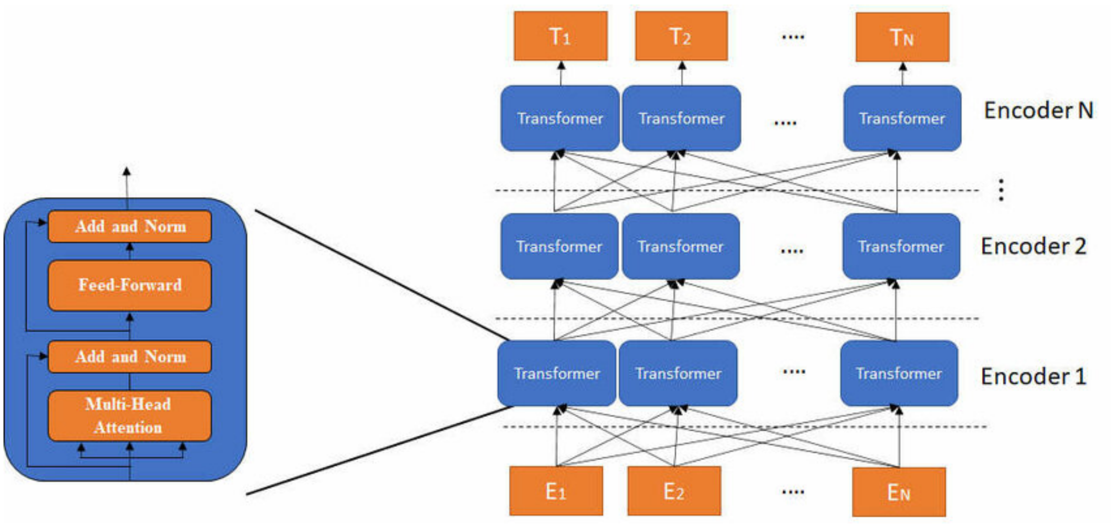


Figure 3.10: Encoder architecutre

Image Source: Article on Pseudocode Generation from Source Code Using the BART Model

Since the BART encoder is bidirectional, it processes the input sequence in both directions. This is achieved by having two separate stacks of self-attention layers: one that processes the input sequence from left to right, and another that processes it from right to left. The outputs of the two stacks are concatenated to form the final output of the encoder.

Each self-attention layer in the BART encoder has three sub-layers: multi-head attention, layer normalization, and feed-forward. In the multi-head attention sub-layer, the input sequence is mapped to queries, keys, and values, which are used to compute an attention distribution over the input sequence. The attention distribution is used to weight the values and produce a context vector, which is then concatenated with the input sequence and passed through a feed-forward network. The feed-forward sub-layer consists of two linear transformations with a ReLU activation in between.

The layer normalization sub-layer is applied after each of the other sub-layers. It normalizes the output of the sub-layer with respect to its mean and standard deviation across the hidden dimension.

The BART encoder also includes positional embeddings, which are added to the input embeddings to provide information about the position of each token in the sequence. The positional embeddings are learned during training, and are added to the input embeddings before being passed through the self-attention layers.

During fine-tuning, the BART model is typically trained using a combination of supervised and unsupervised learning. In supervised learning, the model is trained to minimize a task-specific loss function, such as cross-entropy loss for text classification. In unsupervised learning, the model is trained to reconstruct the input sequence from a corrupted version of itself, such as a sequence with randomly masked tokens.

Bidirectional encoding has several benefits for NLP tasks. One of the main benefits is the ability to capture contextual information from both directions of the input sequence. This allows the model to incorporate information about the past and the future of the input sequence into its representation of each word. This can be particularly useful in tasks such as language modeling, where the model needs to predict the next word in a sequence based on the context.

Another benefit of bidirectional encoding is its ability to capture long-term dependencies in the input sequence. In traditional language models, the probability of the current word is based only on the previous words in the sequence. However, in natural language, the meaning of a word is often dependent on the words that come before and after it. Bidirectional encoding allows the model to capture long-term dependencies by incorporating information about the future of the input sequence into its representation of each word.

Bidirectional encoding also helps to improve the quality of the output generated by the model. By capturing more contextual information from both directions of the input sequence, the model can produce more accurate and coherent output. This is particularly important in tasks such as machine translation, where the model needs to generate a translation that is faithful to the original input and sounds natural in the target language.

Another benefit of bidirectional encoding is its ability to handle variable-length input sequences. In natural language, the length of the input sequence can vary depending on the context. Bidirectional encoding allows the model to handle variable-length input sequences by processing the input sequence in both directions and generating a representation that takes into account the entire input sequence.

Although bidirectional encoding has several benefits for NLP tasks, it also poses several challenges. One of the main challenges is the increased computational complexity of bidirectional processing. Processing the input sequence in both directions requires twice as much computation as processing it in only one direction. This can make the training and inference of bidirectional models slower and more resource-intensive.

Another challenge of bidirectional encoding is the potential for overfitting to the training data. Bidirectional models have the ability to capture more complex patterns in the input sequence, which can lead to overfitting if the training data is not representative of the test data. This can result in poor generalization performance and reduced model robustness.

The bidirectional encoder is a critical component of the BART model that allows the model to capture contextual information from both directions of the input sequence. Bidirectional encoding has several benefits for NLP tasks, including the ability to capture long-term dependencies, handle variable-length input sequences, and produce more accurate and coherent output.

3.4.2. Decoder

The decoder is an essential component of the BART model that is responsible for generating the output sequence from the encoded input sequence. The decoder uses an autoregressive approach to generate the output sequence, where each token is generated based on the previous tokens in the sequence.

The decoder in the BART model is composed of a stack of transformer layers similar to the encoder. Each transformer layer in the decoder has two sub-layers: a masked multi-head attention layer and a feedforward neural network layer. The masked multi-head attention layer takes as input the output of the previous layer and a mask that prevents the decoder from attending to future tokens in the output sequence. The feedforward neural network layer takes as input the output of the masked multi-head attention layer and produces a new representation of the output sequence.

The decoder also includes a positional embedding layer that encodes the position of each token in the output sequence. The positional embedding layer is added to the input of each transformer layer to allow the model to capture the order of the tokens in the output sequence.

Finally, the decoder includes a linear layer and a softmax layer that converts the output of the last transformer layer into a probability distribution over the vocabulary of possible output tokens. The token with the highest probability is selected as the next token in the output sequence.

The working of the decoder can be divided into two phases: training and inference. During the training phase, the decoder is trained to generate the correct output sequence given the input sequence. The training process involves minimizing a loss function that measures the difference between the generated output sequence and the ground truth output sequence.

During the inference phase, the decoder is used to generate the output sequence given a new input sequence. The inference process involves feeding the input sequence through the encoder to obtain the encoded representation of the input sequence. The encoded representation is then fed into the decoder along with a start token that indicates the beginning of the output sequence.

The decoder generates the output sequence token-by-token using an autoregressive approach. At each step, the decoder attends to the encoded input sequence using the masked multi-head attention layer and produces a new representation of the output sequence using the feedforward neural network layer. The output of the last transformer layer is passed through the linear and softmax layers to obtain a probability distribution over the possible output tokens. The token with the highest probability is selected as the next token in the output sequence, and the process is repeated until an end-of-sequence token is generated or a maximum sequence length is reached.

After we get the token and sentence-level representation of an input text sequence, a decoder needs to interpret these to map with the output target. However, by using a similarly designed decoder, tasks such as next sentence prediction or token prediction might perform poorly since the model relies on a more comprehensive input prompt. In these cases, we need model architectures that can be trained on generating the next word by only looking at the previous words in the sequence. Hence, a causal or autoregressive model that looks only at the past data to predict the future comes in handy.

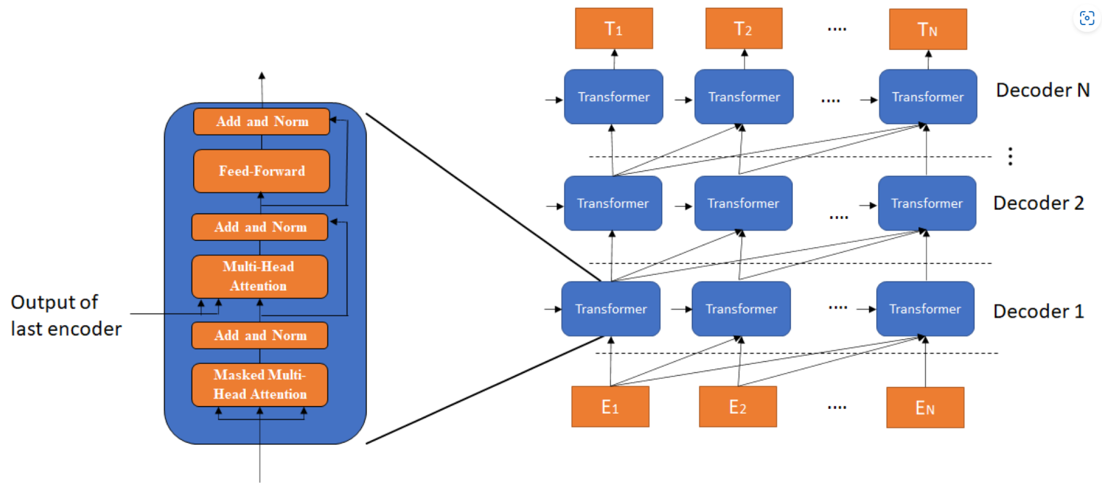


Figure 3.11: Decoder architecutre

Image Source: Pseudocode Generation from Source Code Using the BART Model

The BART decoder is based on the transformer decoder architecture, which is a stack of transformer decoder layers. Each transformer decoder layer has two sub-layers, namely the masked multi-head attention layer and the encoder-decoder attention layer, followed by a position-wise feed-forward network. These sub-layers are connected by residual connections, and layer normalization is applied to the outputs of each sub-layer.

The masked multi-head attention sub-layer takes the decoder input embeddings as queries, keys, and values, and applies the masked self-attention mechanism to the queries. The masked self-attention mechanism prevents the decoder from attending to future positions in the target sequence during training. During decoding, the decoder input embeddings are replaced with the tokens generated by the decoder in the previous time steps.

The encoder-decoder attention sub-layer takes the output of the masked multi-head attention sub-layer and the output of the encoder as queries and keys, respectively, and generates the attention distribution over the encoder outputs. This sub-layer allows the decoder to attend to the relevant parts of the input sequence while generating the summary or continuation.

The position-wise feed-forward network applies a linear transformation followed by a non-linear activation function, such as ReLU, to each position in the sequence independently. This sub-layer introduces non-linearity to the model and helps capture complex relationships between the input and output sequences.

In the BART decoder, the decoder input embeddings are first passed through an embedding layer, which maps the input tokens to continuous vectors of a fixed dimension. These embeddings are then passed through the stack of transformer decoder layers, which generate the output sequence token by token. During training, the decoder is optimized to minimize the negative log-likelihood of the target sequence given the input sequence. During decoding, the tokens with the highest probabilities are selected at each time step.

The decoder in the BART model has several benefits for NLP tasks. One of the main benefits is its ability to generate output sequences that are coherent and semantically meaningful. The decoder uses the encoded representation of the input sequence to generate the output sequence, which allows it to capture the context of the input sequence and produce output that is consistent with the input.

Another benefit of the decoder is its ability to handle variable-length output sequences. The autoregressive approach used by the decoder allows it to generate output sequences of arbitrary length, which is important for tasks such as text generation and machine translation where the length of the output sequence can vary depending on the input.

Finally, the decoder also allows the model to learn from the generated output sequence during training. This is achieved using a technique called teacher forcing, where the ground truth output sequence is used as input to the decoder during training instead of the generated output sequence. This allows the model to learn from the correct output sequence and helps to prevent the model from generating incorrect output sequences.

The decoder is an essential component of the BART model that is responsible for generating the output sequence from the encoded input sequence. It uses a stack of transformer layers, each with a masked multi-head attention layer and a feedforward neural network layer, to generate the output sequence token-by-token in an autoregressive manner. The decoder benefits from its ability to generate coherent and semantically meaningful output sequences, handle variable-length output sequences, and learn from the generated output sequence during training using teacher forcing.

The decoder is a key component of the BART model that has shown impressive results on a range of NLP tasks. Its ability to generate high-quality output sequences and handle variable-length sequences makes it well-suited for tasks such as text generation, machine translation, and summarization. The decoder's autoregressive approach also allows it to capture the context of the input sequence and generate output that is consistent with the input. Overall, the decoder plays a critical role in the BART model and its success in many NLP applications.

3.5. K Medoids Clustering Algorithm

K Medoids clustering algorithm [9] is a partition-based clustering algorithm that aims to divide a set of data points into K number of clusters. The algorithm utilizes a concept called "medoids" which are the most representative data points of a cluster. Unlike other clustering algorithms, the K Medoids algorithm does not use the mean values of the data points in a cluster, but rather chooses a data point as the medoid that is closest to the center of the cluster.

K-medoids is a clustering algorithm that is widely used in data mining, pattern recognition, and machine learning. It is a variant of the K-means algorithm that can handle non-numeric data and is robust to noise and outliers.

Clustering is the process of dividing a set of data points into groups, or clusters, based on the similarity of the data points. The goal of clustering is to group similar data points together and to separate dissimilar data points. Clustering algorithms are widely used in data analysis, machine learning, and pattern recognition. One such clustering algorithm is K-medoids.

K-medoids is a partitioning clustering algorithm that partitions a set of n data points into k clusters. The algorithm minimizes the sum of dissimilarities between each data point and its assigned cluster medoid, where a medoid is a representative point of a cluster. The dissimilarity measure used can be any metric such as Euclidean distance, Manhattan distance, or cosine similarity.

The algorithm proceeds as follows:

- Initialize k medoids randomly from the data points.
- Assign each data point to the nearest medoid.
- For each cluster, compute the sum of dissimilarities between the medoid and all other data points in the cluster.
- For each medoid, swap it with each non-medoid data point in its cluster and compute the total sum of dissimilarities. If the sum of dissimilarities is lower than the current medoid, swap the medoid with the non-medoid data point.
- Repeat steps 2-4 until convergence.

The algorithm terminates when the medoids do not change between iterations or a maximum number of iterations is reached. The K-medoids algorithm is a simple and efficient algorithm that can handle non-numeric data and is robust to noise and outliers. The algorithm works by iteratively assigning data points to the nearest medoid and updating the medoids based on the total sum of dissimilarities. The algorithm converges when the medoids do not change between iterations or a maximum number of iterations is reached.

The K-medoids algorithm has several advantages over other clustering algorithms such as K-means. One advantage is its ability to handle non-numeric data, such as text, categorical data, or images. Another advantage is its robustness to noise and outliers, as the algorithm uses the medoids to represent the cluster instead of the mean, which is sensitive to outliers. Additionally, the algorithm can handle a large number of data points and is computationally efficient.

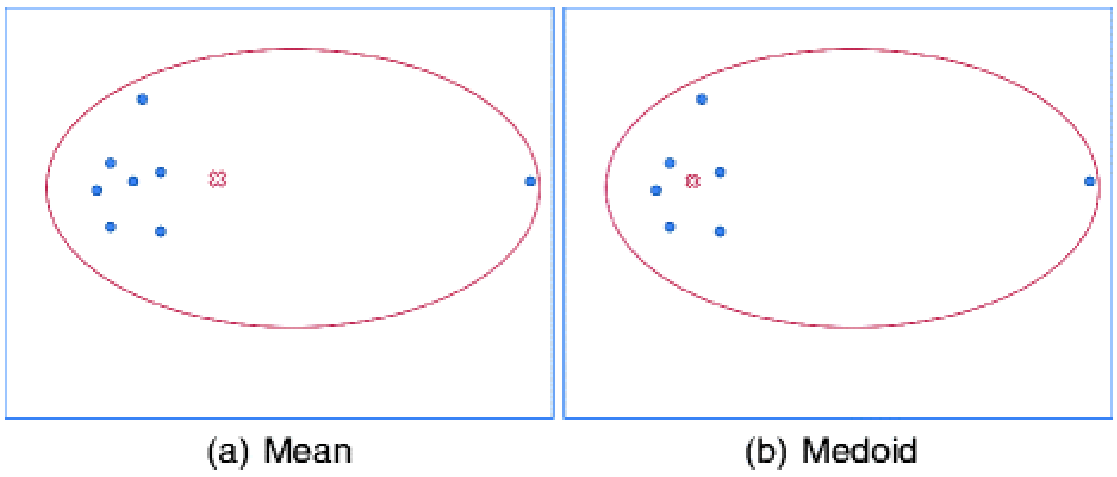


Figure 3.12: KMeans vs KMediods

Image Source: Kmedidos Clustering, Springer Link

Although K-medoids is a powerful clustering algorithm, it has some limitations. One limitation is its sensitivity to the initial medoids, as the algorithm can converge to a suboptimal solution if the initial medoids are not chosen carefully. Another limitation is its requirement for a fixed number of clusters, which may not be known beforehand. Finally, the algorithm may converge to a local minimum, which may not be the global optimum.

K-medoids is a versatile algorithm that has several applications in data mining, pattern recognition, and machine learning. One application is in image segmentation, where the algorithm is used to group similar pixels together based on their color or texture. Another application is in text clustering, where the algorithm is used to group similar documents based on their content or keywords. The algorithm is also used in customer segmentation, where it is used to group customers based on their purchasing behavior or demographic information.

In conclusion, K-medoids is a clustering algorithm that is widely used in data mining, pattern recognition, and machine learning. It is a variant of the K-means algorithm that can handle non-numeric data and is robust to noise and outliers. The algorithm partitions a set of n data points into k clusters by minimizing the sum of dissimilarities between each data point and its assigned cluster medoid.

K-medoids has several advantages over other clustering algorithms such as K-means, including its ability to handle non-numeric data, robustness to noise and outliers, and computational efficiency. However, the algorithm has some limitations such as sensitivity to initial medoids, requirement for a fixed number of clusters, and convergence to a local minimum.

The algorithm has several applications in image segmentation, text clustering, and customer segmentation. K-medoids is a versatile algorithm that can be applied to a wide range of data mining and machine learning problems.

3.6. KNN Algorithm

The K-nearest neighbors (KNN) algorithm [10] is a popular and effective machine learning algorithm for classification and regression problems. It is a non-parametric algorithm that works by identifying the K nearest neighbors to a given data point and making a prediction based on their classification or regression values.

Machine learning is an area of computer science that aims to develop algorithms that can learn from data and make predictions or decisions based on that data. One popular type of machine learning algorithm is the K-nearest neighbors (KNN) algorithm. KNN is a simple and intuitive algorithm that can be used for both classification and regression problems. The basic idea behind KNN is to find the K nearest neighbors to a given data point and make a prediction based on their values. In this report, we will describe the principles, methods, and applications of the KNN algorithm.

The KNN algorithm is based on the principle that similar data points tend to be close to each other in the feature space. The feature space is a mathematical space in which each data point is represented by a set of features or attributes. The KNN algorithm works by calculating the distance between each data point and its nearest neighbors in the feature space. The distance measure used can vary depending on the nature of the problem and the type of data being used. Some common distance measures used in KNN include Euclidean distance, Manhattan distance, and Minkowski distance.

Once the distances between each data point and its neighbors have been calculated, the algorithm selects the K nearest neighbors to the given data point. The value of K is a hyperparameter that must be specified before the algorithm is run. The choice of K can have a significant impact on the performance of the algorithm. A smaller value of

K will lead to a more flexible model that is better able to capture local patterns in the data. However, a smaller value of K may also lead to overfitting, where the model is too complex and does not generalize well to new data. A larger value of K will lead to a more rigid model that is less prone to overfitting but may also miss local patterns in the data.

Once the K nearest neighbors have been identified, the algorithm makes a prediction based on their classification or regression values. For classification problems, the most common method is to use a majority vote, where the predicted class is the one that occurs most frequently among the K nearest neighbors. For regression problems, the most common method is to use a weighted average, where the predicted value is the weighted average of the regression values of the K nearest neighbors.

The KNN algorithm can be implemented using several methods, including brute force, KD-trees, and ball trees. The brute force method is the simplest and most straightforward method. It works by calculating the distance between each data point and every other data point in the feature space. While this method is easy to implement, it can be computationally expensive, especially for large datasets. The KD-tree method is a more efficient method that works by partitioning the feature space into smaller subspaces using binary trees. Each node in the tree represents a subspace of the feature space, and the tree is constructed recursively by partitioning each subspace into two smaller subspaces. This method is more efficient than the brute force method and can be used for datasets with up to millions of data points.

The ball tree method is another efficient method that works by partitioning the feature space into a set of nested hyperspheres. Each node in the tree represents a hypersphere that contains a set of data points, and the tree is constructed recursively by partitioning each hypersphere into two smaller hyperspheres. This method is particularly

In a typical KNN algorithm, the distance between the query point and each of the data points in the training set is calculated using a distance metric such as Euclidean distance. The K data points that are closest to the query point are then selected, and the class label for the query point is assigned based on the majority class of those K points.

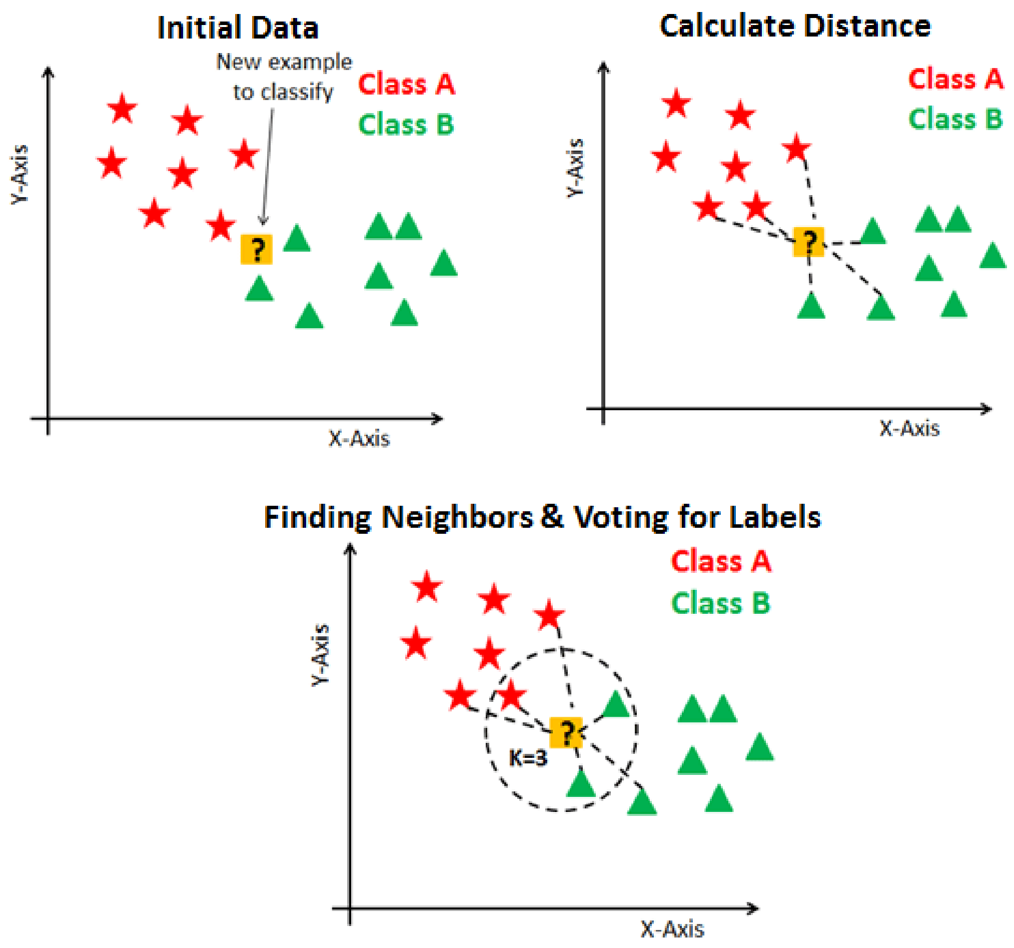


Figure 3.13: KNN Algorithm

Image Source: Medium, K-Nearest Neighbor(KNN) Algorithm Explained

In our project, we use the KNN algorithm to rank the sentences in each cluster and select the most important ones for summarization. Specifically, we use the cosine similarity metric to calculate the distance between each sentence in a cluster and a query point. The query point is chosen as the centroid of the cluster, which is the average of all the feature vectors of the sentences in the cluster. Then, we select the K sentences that are closest to the centroid based on the cosine similarity score, and use them to generate a summary for that cluster.

3.7. Neural Text-To-Speech

Text-to-speech (TTS) technology is an application of speech synthesis that allows text to be converted into spoken language. TTS technology has been around for several decades and has seen significant improvements in recent years due to advancements in neural networks and deep learning. Neural TTS, in particular, has shown impressive results in generating natural-sounding speech that can mimic human speech patterns and intonations. In this report, we will describe the principles, methods, and applications of neural TTS.

Neural Text-To-Speech (NTTS) is a technology that enables computers to generate human-like speech from written text. NTTS systems use deep neural networks to model the mapping between text and speech, allowing for more natural and expressive synthetic voices.

Speech is a fundamental means of human communication, and as such, it plays a crucial role in many applications. For example, text-to-speech (TTS) technology can be used to provide speech output for people with visual impairments, to generate audio books, or to create realistic virtual assistants. However, traditional TTS systems have limitations in terms of voice quality and expressiveness. Neural Text-To-Speech (NTTS) technology aims to overcome these limitations by using deep neural networks to model the mapping between text and speech.

NTTS technology is based on the principle that speech can be generated by modeling the relationship between text and speech in a large corpus of audio recordings. This is achieved through the use of deep neural networks, which are powerful machine learning models that can learn complex patterns in the data. The goal of NTTS is to train a neural network to predict the speech waveform from a given input text. The neural network architecture used in NTTS systems typically consists of an encoder-decoder structure, similar to that used in machine translation models. The encoder component takes as input the text to be synthesized and encodes it into a high-dimensional vector representation, which captures the meaning and structure of the text. The decoder component takes the encoded text as input and generates the corresponding speech waveform.

The training process for NTTS models involves optimizing the parameters of the neural network to minimize the difference between the predicted speech waveform and the target waveform. This is done using a combination of supervised learning and data augmentation techniques, such as adding noise or varying the speaking rate. The resulting model can then be used to generate synthetic speech from new text inputs.

There are several different methods used in NTTS research, each with its own strengths and limitations. One approach is to use a traditional TTS system to generate a large corpus of speech data, which can then be used to train a neural network model. This approach has the advantage of leveraging existing TTS technology, but it can be computationally expensive and may not capture all of the nuances of human speech.

Another approach is to use a speech synthesis dataset, such as the Blizzard Challenge dataset, which is designed specifically for NTTS research. These datasets typically contain thousands of hours of high-quality speech recordings, along with their corresponding text transcripts. This approach can be more efficient than using a traditional TTS system, but it may not generalize well to new domains or languages.

Recent advances in NTTS research have focused on improving the naturalness and expressiveness of synthetic speech. One approach is to use generative adversarial networks (GANs) to generate speech, which can produce more realistic and diverse outputs than traditional models. Another approach is to use style transfer techniques to modify the prosodic features of speech, such as pitch and speaking rate, to make the synthetic voice more expressive.

Neural TTS consists of three major components: the text analyzer, the neural acoustic model, and the neural vocoder. To generate natural synthetic speech from text, text is first input into the text analyzer, which provides output in the form of phoneme sequence. A phoneme is a basic unit of sound that distinguishes one word from another in a particular language. A sequence of phonemes defines the pronunciations of the words provided in the text. Next, the phoneme sequence goes into the neural acoustic model to predict acoustic features that define speech signals. Acoustic features include the timbre, the speaking style, speed, intonations, and stress patterns. Finally, the neural vocoder converts the acoustic features into audible waves, so that synthetic speech is generated.

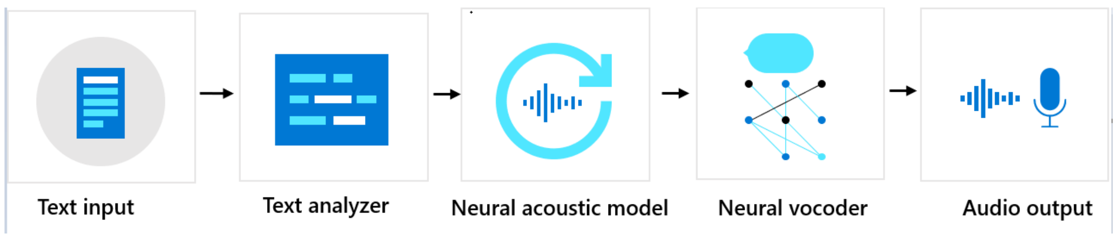


Figure 3.14: Neural TTS Pipeline

Image Source: Microsoft Azure Documentation, The basics of Custom Neural Voice

NTTS technology has a wide range of applications in various fields, including assistive technology, entertainment, and education. One application is to provide speech output for people with visual impairments, allowing them to access written information more easily. Another application is to create realistic virtual assistants, which can interact with users in a natural and intuitive way. NTTS technology can also be used to generate synthetic voices for video games, movies, and other entertainment media.

In the field of education, NTTS technology can be used to create audio books or to provide speech feedback for language learning applications. NTTS systems can also be used to create synthetic voices for individuals who have lost their ability to speak due to medical conditions, such as ALS or laryngeal cancer. Despite the recent advances in NTTS research, there are still several challenges that need to be addressed. One challenge is to improve the robustness of NTTS systems to different speaking styles, accents, and languages. This requires large and diverse datasets, as well as more sophisticated modeling techniques.

Another challenge is to improve the naturalness and expressiveness of synthetic speech. While NTTS systems can produce high-quality speech, there is still room for improvement in terms of intonation, rhythm, and other prosodic features. This requires further research into advanced modeling techniques and the use of expressive speech corpora.

In the future, NTTS technology is expected to have a significant impact on various industries, including entertainment, education, and healthcare. For example, synthetic voices may be used to create more personalized and interactive virtual assistants, or to provide speech therapy for individuals with speech disorders.

3.8. Markdown

Markdown is a lightweight markup language that is widely used for writing documentation, creating web pages, and writing content for blogs. It was developed in 2004 by John Gruber and Aaron Swartz with the aim of providing a simple syntax for formatting text that could be easily converted to HTML. Markdown has gained popularity over the years, thanks to its ease of use and the fact that it can be used to create content that is both readable and easy to understand.

The syntax of Markdown is designed to be simple and intuitive. It uses a combination of special characters and punctuation marks to format text. For example, to create a heading in Markdown, you simply add one or more hash symbols (#) at the beginning of the line. The number of hash symbols you use indicates the level of the heading. For example, one hash symbol creates a level 1 heading, two hash symbols create a level 2 heading, and so on.

Another useful feature of Markdown is its ability to create links. To create a link, you simply surround the text you want to use as the link's anchor text with square brackets, and then follow it with the link URL in parentheses. For example, to create a link to Google, you would write Google. Markdown also supports inline links, which are created by surrounding the link text with angle brackets, like so: https://www.google.com/.

Markdown also has support for formatting text with emphasis and strong emphasis. To create italicized text, you simply surround the text with a single asterisk (*) or an underscore (_). To create bold text, you surround the text with two asterisks (**) or two underscores (__).

In addition to these basic features, Markdown also supports a wide range of advanced formatting options, such as creating tables, adding images, and creating footnotes. Tables are created by using pipes (|) and hyphens (-) to create a grid of cells. Images are inserted using the same syntax as links, but with an exclamation mark (!) at the beginning. Footnotes are created by placing a caret (^) at the end of the text you want to create the footnote for, followed by the footnote text in square brackets.

One of the great things about Markdown is that it can be easily converted to other formats, such as HTML, PDF, or Word. This is done using a tool called a Markdown parser. A Markdown parser is a program that reads Markdown text and converts it into another format. There are many different Markdown parsers available, both online and offline, and they vary in their features and capabilities.

One popular application of Markdown is for generating presentation. It is a popular technique for creating visually appealing and engaging presentations. By using markdown syntax, it becomes easy to create compelling presentations that can be displayed on a variety of platforms.

There are several tools available for creating presentations using markdown, such as MARP, Remark, and reveal.js. MARP is a popular markdown presentation tool that allows users to create high-quality presentations with minimal effort. In this article, we will discuss how to use MARP to create presentations using markdown syntax.

3.9. Video

Video is a sequence of images, also known as frames, that are shown in quick succession to create the illusion of motion. It is an important medium for entertainment, communication, and education, and is widely used in various industries, including television, film, advertising, and social media.
There are various ways where video might be generated or captured like filming through input devices, making animations through graphics, combining image sequences with audios etc.
We'll only be discussing about the theory of generating video through combining audio and images, as it is the process in the project's pipeline. Some of the steps in the video generation are :

- The first step in this process is to collect and organize the image frames that will be used to generate the video. These images can be in various formats such as JPEG, PNG, or BMP. It is essential to ensure that all the images are of the same resolution and aspect ratio to avoid any distortion in the final video.

- Once the images are organized, the next step is to select an appropriate audio track. The audio can be in various formats such as MP3, WAV, or AAC. It is important to select an audio track that is compatible with the video format and is of high quality.

- After selecting the images and audio, the next step is to use a software tool to concatenate the images and audio to generate the final video. One such popular tool is FFMPEG. FFMPEG is a cross-platform software that is widely used for video manipulation and conversion. It is an open-source tool that supports various video and audio formats.

4. METHODOLOGY

4.1. Process Model

The AGILE process model was employed throughout the lifetime of the project.

- **Planning and Research:** Since this is a heavy project involving comparision and review of different models. We had to plan and research every tech stack that will be need to be used.
- **Analysis:** Requirements were selected for our project. The thorough comparision of every tech and its alternative was done to match our requirements.
- **Design:** In designing phase, we initiated designing elements for the project by considering the requirements.
- **Implementation:** We started working on our project, to deploy a working product.
- **Testing:** The quality of the product was examined as well as the performance. Bugs were also detected in this phase to solve.

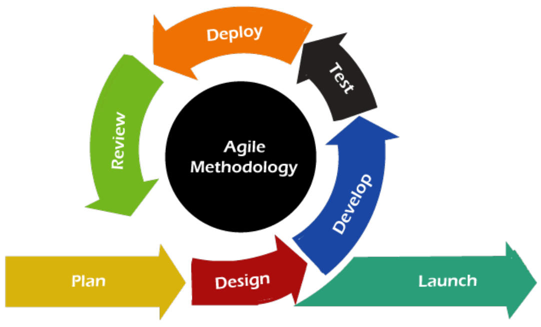


Figure 4.1: Agile Model

This process model was used for 5 iterations. In the first iteration, indepth planning and research was done for the overall cycles of project. And then the process was carried out. Starting from next iterations, we started from analysis phase of the process model upto testing the product. The first two iteration were done in the first 6 months of the conception of the project. And the last three iterations were done after the 6 months period.

4.2. Planning and Research

Since two primary tasks are performed in this project i.e, Summarization tasks (NLP) and Speech Synthesis. So research on those tasks must be performed. This includes analyzing the dataset to identify the text format and language, determining the desired length and style of the summary, and selecting appropriate evaluation metrics. The summarization method must also be chosen based on the specific requirements of the task. The dataset must be preprocessed, including removing unnecessary information and performing text normalization, and suitable summarization techniques must be applied to generate summaries that are coherent and informative.
Appropriate datasets and model should be used. Models should be chosen differently for different techniques like Extractive or Abstractive Summarization. In this case we have chose BART as our primary model because it can be used for both Extractive and Abstractive Summarization.

Similarly speech synthesis planning and research includes analyzing the dataset to identify the language and quality of the audio data, determining the desired tone and style of the synthesized speech, and selecting appropriate evaluation metrics. The synthesis method must also be chosen based on the specific requirements of the task. The dataset must be preprocessed, including removing unnecessary information and performing audio normalization, and suitable speech synthesis techniques must be applied to generate speech that is natural-sounding and intelligible.
However due to overwhelming and easy access to more cloud services, it is relatively easy to use already deployed neural speech synthesis. In our case, the model we chose was Azure Speech services after thorough planning and research.

4.3. Technological Analysis

The analysis of library, tools and technologies must be performed to perform the project in a reliable way. Some primary libraries, tools and technologies that we have analyzed and used as a result are:

- NumPy is a powerful numerical computing library for Python that provides efficient ways to work with large multidimensional arrays and matrices. It includes a variety of mathematical functions and operators that make it easy to perform complex computations on large datasets. NumPy is widely used in data science, machine learning, and scientific computing applications. We have used numpy to store vector embeddings and representation from the model as it computationally fast.

- Scikit-learn is a popular machine learning library for Python that provides a variety of tools and algorithms for data analysis, classification, regression, and clustering. It includes a range of algorithms, such as decision trees, random forests, and support vector machines, that can be used for a wide range of machine learning tasks. Scikit-learn is known for its easy-to-use API and extensive documentation. Scikit learn was used to use KMediods algorithm and perform clustering.
- Torch is an open-source machine learning library for Python that is widely used for deep learning and neural networks. It provides a flexible and efficient platform for building, training, and deploying machine learning models. Torch includes a variety of tools and features that make it easy to work with large datasets, including GPU support for accelerated training. For tarining and generating summary, we have used torch library as it was well documented and contained easy to use functions.
- Transformers is a state-of-the-art natural language processing library for Python that is widely used for tasks such as language translation, sentiment analysis, and summarization. It provides pre-trained models for a variety of tasks and includes a variety of tools and features that make it easy to fine-tune models on specific datasets. Transformers is known for its speed and accuracy, and is widely used in both research and industry. We have extensively used transformers for loading BART model and tokenizer.
- Azure Speech Service library [11] is a easy to use cloud based speech synthesis library that provides RESTful APIs to communicate with applications. It allows developers to add text-to-speech capabilities to their applications and services, enabling users to listen to the content instead of reading it.
- AWS Learner's Lab is a cloud-based learning platform offered by Amazon Web Services (AWS). It provides a comprehensive set of resources and tools for learn-

ing about machine learning and deep learning. Therefore AWS learner lab can be easily used to train the models.

- MARP is a powerful open-source tool that enables users to create dynamic and professional presentations using Markdown syntax. MARP works by converting text written in Markdown syntax into HTML, which is then rendered by the browser. This process allows for a highly customizable and flexible presentation format that can be easily updated and shared across multiple platforms. Because of its simplicity, programmability and flexibility makes it the better choice for generating slides.
- FFmpeg is a powerful, cross-platform multimedia framework and command-line tool used to decode, encode, transcode, and stream audio and video files. FFmpeg can handle a wide range of file formats and codecs, including popular formats like MP4, AVI, WMV, MKV, FLV, and MOV. Because its free, simple and works well with python command line arguments, it is better suited for the project above others.

4.4. Dataset Analysis

For the downstream summarization tasks, since the learning is fully supervised we need large dataset to work on. The requirements of dataset that can fulfil the summarization task for the articles involve:

- The datasets should contain various domain of topics for handling various articles.
- The datasets' summaries must be written in consistent manner so that reliable results are produced. For article summary dataset, the most common framework for writing article is inverse pyramid structure as shown in figure below :

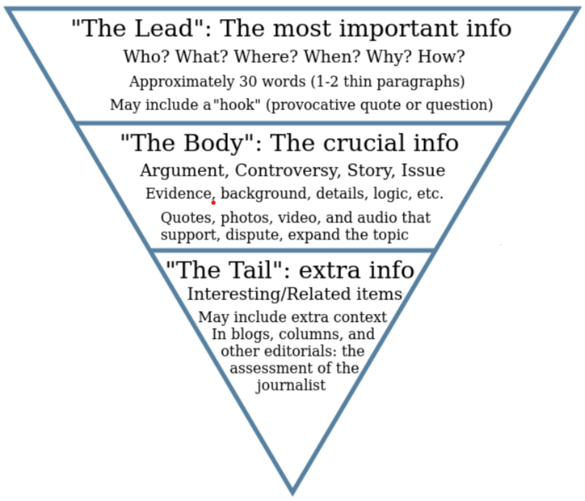
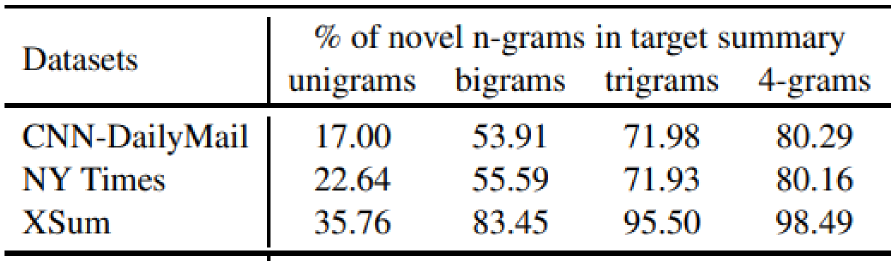

"The Lead": The most important info
Who? What? Where? When? Why? How?
Approximately 30 words (1-2 thin paragraphs)
May include a "hook" (provocative quote or question)
"The Body": The crucial info
Argument, Controversy, Story, Issue
Evidence, background, details, logic, etc.
Quotes, photos, video, and audio that support, dispute, expand the topic
"The Tail": extra info
Interesting/Related items
May include extra context
In blogs, columns, and other editorials: the assessment of the journalist

Figure 4.2: Inverse Pyramid Writing Style

Image Source: Paper on A Large Scale Text Summarization Dataset

- The datasets quality should be paramount to the objective score as well subjective standards by human evaluation.

Considering the requirements, different datasets like Xsum Dataset, NYT, and CNN/DM Dataset were analyzed. Comparision of dataset is shown below:



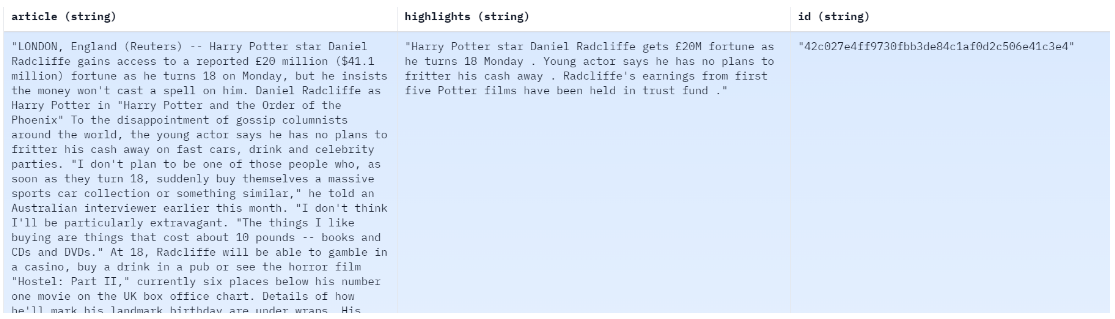
Datasets	% of novel n-grams in target summary			
	unigrams	bigrams	trigrams	4-grams
CNN-DailyMail	17.00	53.91	71.98	80.29
NY Times	22.64	55.59	71.93	80.16
XSum	35.76	83.45	95.50	98.49

Figure 4.3: Comparision of Datasets

Image Source: MultiXScience White Paper

We see that XSum has greater overlap in summary, but that might be the case because of the fact XSum is extreme summary i.e, it only summarizes the article to its heading. But the CNN/DM Dataset turned out to be the best possible dataset for our use case according to the overall requirements. The summary in CNN/DM is written by world-class writers from the news organizations Cable News Network (CNN) and Daily Mail (DM).

Format in which dataset is prepared is shown below here:



article (string)	highlights (string)	id (string)
"LONDON, England (Reuters) -- Harry Potter star Daniel Radcliffe gains access to a reported £20 million ($41.1 million) fortune as he turns 18 on Monday, but he insists the money won't cast a spell on him. Daniel Radcliffe as Harry Potter in "Harry Potter and the Order of the Phoenix" To the disappointment of gossip columnists around the world, the young actor says he has no plans to fritter his cash away on fast cars, drink and celebrity parties. "I don't plan to be one of those people who, as soon as they turn 18, suddenly buy themselves a massive sports car collection or something similar," he told an Australian interviewer earlier this month. "I don't think I'll be particularly extravagant. "The things I like buying are things that cost about 10 pounds -- books and CDs and DVDs." At 18, Radcliffe will be able to gamble in a casino, buy a drink in a pub or see the horror film "Hostel: Part II," currently six places below his number one movie on the UK box office chart. Details of how he'll mark his landmark birthday are under wraps. His	"Harry Potter star Daniel Radcliffe gets £20M fortune as he turns 18 Monday . Young actor says he has no plans to fritter his cash away . Radcliffe's earnings from first five Potter films have been held in trust fund ."	"42c027e4ff9730fbb3de84c1af0d2c506e41c3e4"

Figure 4.4: CNN/DM Dataset Format

Image Source: Kaggle CNN/DM Dataset

4.5. Training & Finetuning

We used PyTorch and Learner Lab environment to fine-tune the BART model on the CNN/DM dataset.

The training process involves the following steps:

- Forward Propagation: Given an input sequence, the model performs forward propagation through the layers to produce a prediction.
- Loss Calculation: The model calculates the cross-entropy loss [12] between the predicted probability distribution and the true probability distribution.
- Backward Propagation: The model calculates the gradients of the loss with respect to the model parameters using backpropagation [13] through the layers.
- Parameter Update: The model updates the parameters using an optimization algorithm such as Adam.
- Repeat: The model repeats the above steps for multiple epochs until the loss converges or a maximum number of epochs is reached.

The optimization algorithm, the learning rate, and the batch size for training must be defined according to the need

- Adam Optimizer: Adam [14] is a popular optimization algorithm used in deep learning for optimizing stochastic gradient descent (SGD). It combines the advantages of two other optimization algorithms: Adagrad and RMSProp. The Adam optimizer keeps an exponentially decaying average of past gradients and squared gradients, respectively. It is particularly effective when dealing with large datasets and high-dimensional parameter spaces. Transformers require a lot of computational resources to train, and the Adam optimizer has been found to be particularly effective for optimizing the large number of parameters in these models.
- Learning rate Scheduler: Learning rate scheduler is a technique used in training machine learning models to adjust the learning rate during training. The learning rate is a hyperparameter that controls the step size at each iteration during the optimization process
- Batch Size: The batch size is the number of examples processed in one iteration of training. Choosing an appropriate batch size is important for fine-tuning BART on the CNN/DM dataset. Here are a few considerations when selecting a batch size:
  - GPU Memory: The batch size should not exceed the available GPU memory. A larger batch size can result in faster training, but it can also lead to out-of-memory errors.
  - Training Time: A larger batch size can result in faster training, but it can also lead to slower convergence and lower performance.
  - Dataset Size: A larger dataset can support a larger batch size, while a smaller dataset may require a smaller batch size to prevent overfitting.

4.6. Model Analysis

The Language Models to use must be selected based on the objective score and requirements of the use case. Some of the basic requirements to select a model for summarization tasks on articles are:

- The language model should have high performance in terms of generating accurate and coherent summaries.

- The language model should be able to understand and summarize complex and diverse topics and be able to capture important details and context from the source article.
- The language model should be able to accept large input articles and provide a summary of specified length.
- The language model that is pretained should be adaptable for different down-stream tasks and should be of low resource on the user-end.

For out dataset, CNN/DM different models are employed for abstractive as well as extractive summarization tasks. comparitive analysis of different models are given as:

Table 1: ROUGE scores for various language models on summarization tasks

Model	ROUGE-1	ROUGE-2	ROUGE-L
T5	43.0	19.9	39.2
Pegasus	43.0	21.5	39.0
GPT-2	40.8	18.0	37.7
GPT-3	42.2	20.8	38.7
UniLM	42.8	20.5	38.4
XLNet	43.3	21.5	39.2

As we will in the results section, the score for the BART large beats the other model. Similarly BART base also performs as comparable to T5 Peagasus for CNN/DM on summarization tasks.

4.7. Evaluation - Rogue Score

ROGUE (Recall-Oriented Understudy for Gisting Evaluation) [15] is a set of metrics used to evaluate the quality of automatic summarization systems. ROGUE measures the similarity between the generated summary and the reference summaries by counting the number of overlapping units, such as words or n-grams, between the two.

There are several variations of ROGUE, including ROGUE-1, ROGUE-2, and ROGUE-L, which differ in the size of the units being compared. ROGUE-1 compares unigrams (single words), ROGUE-2 compares bigrams (pairs of adjacent words), and ROGUE-L uses the longest common subsequence (LCS) of words between the generated summary and the reference summaries.

To calculate ROGUE, we first need to define a set of reference summaries for each document or text. The reference summaries should represent the key information or main points of the document. Then, we compare the generated summary with each reference summary using one of the ROGUE variations. The final ROGUE score is the average of the scores for all reference summaries.

For example, to calculate ROGUE-1, we count the number of unigrams that appear in both the generated summary and each reference summary, and then divide by the total number of unigrams in the reference summaries. The formula for ROGUE-1 is:

ROGUE-1 = (# overlapping unigrams) / (# unigrams in reference summaries)

Similarly, for ROGUE-2, we count the number of bigrams that appear in both the generated summary and each reference summary, and divide by the total number of bigrams in the reference summaries.

ROGUE-L uses the longest common subsequence (LCS) of words between the generated summary and each reference summary. The LCS is the longest sequence of words that appear in both the generated summary and the reference summary in the same order. The formula for ROGUE-L is:

ROGUE-L = LCS / (# words in reference summaries)

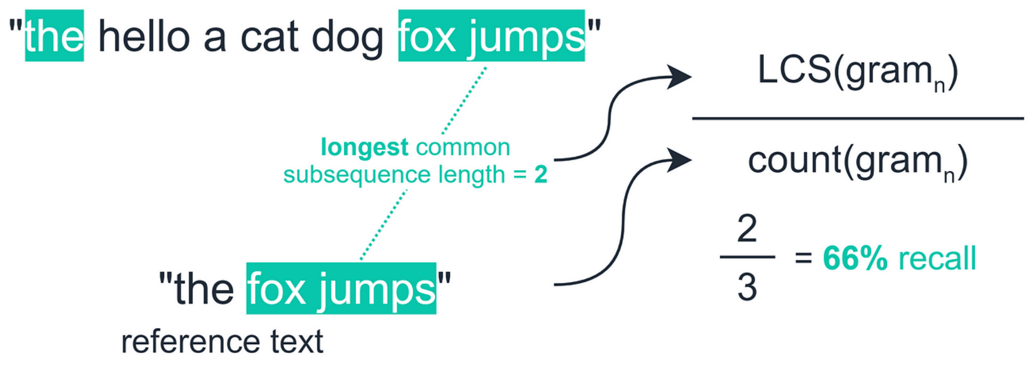

"the hello a cat dog fox jumps"
longest common subsequence length = 2
"the fox jumps"
reference text
$\frac{LCS(gram_n)}{count(gram_n)}$
$\frac{2}{3}$ = 66% recall

Figure 4.5: Rogue Score Calculation for long sequence

Image Source: The Ultimate Metric in NLP

Overall, ROGUE provides a useful way to measure the quality of summarization systems and compare different models or algorithms.

4.8. Mean Opinion Score

Mean Opinion Scores, as commonly used today, originated from polls of test subjects listening to audio or observing video. A number of current standards can be traced back to expert listeners and observers in distraction free quiet rooms subjectively logging experience scores. A MOS itself is a metascore, averaged from a number of individual components of session quality.

Nowadays, audio and video communications isn't scored by a panel of individuals, but by a number of algorithms (Objective Measurement Methods) that attempt to approximate human experience. ITU-T's P.800.1 discusses objective and subjective scoring of telephone transmission quality, while recommendations such as P.863 and J.247 cover speech and video quality, respectively.

The most commonly used rating scale is the Absolute Category Ranking (ACR) scale, which ranges from 1 to 5. The levels of the Absolute Category Ranking are:

- 5 Excellent
- 4 Good
- 3 Fair
- 2 Poor
- 1 Bad

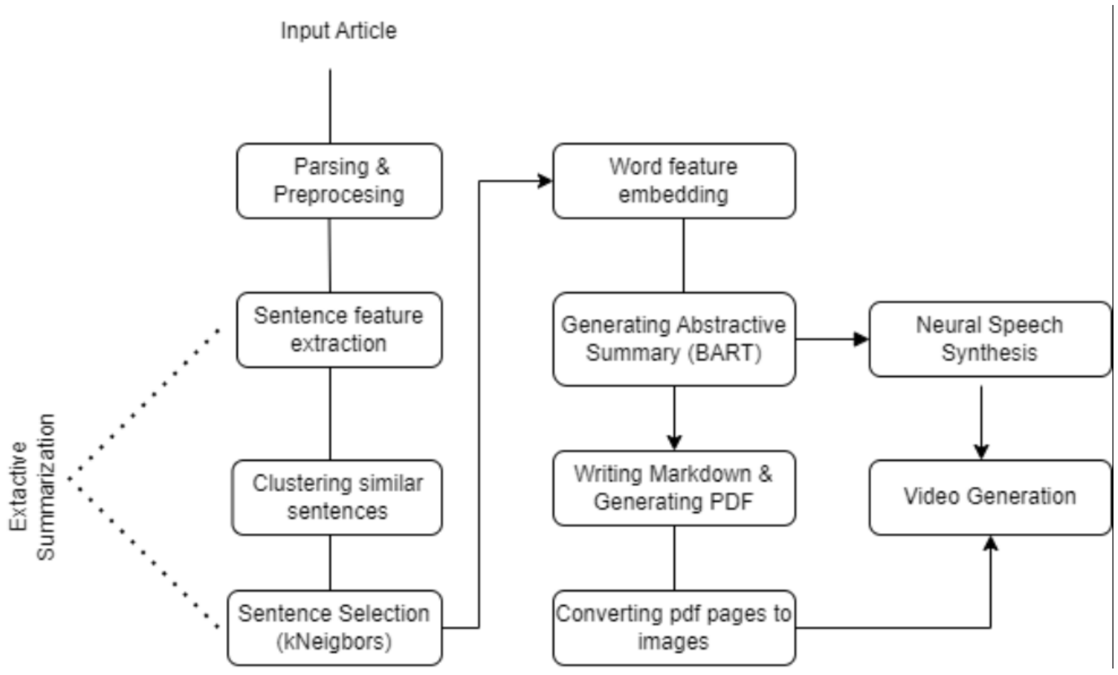
All links in the chain from sender to receiver can cause a drop in mean opinion score. Everything from a human's health to audio and video equipment to computer settings can cause a degradation in communications quality. However, network effects are most readily apparent and measurable on these calls - jitter, latency, and packet loss lend themselves to numerical measurement, and have a direct effect on perceived call quality.

Generically, a Mean Opinion Score can be employed anywhere human subjective experience and opinion is useful. In practice, it is often used to judge digital approximations of world phenomena.

Commonly employed domains where Mean Opinion Score is applied include static image compression (e.g. JPG, GIF), audio codecs (e.g. MP3, Vorbis, AAC, Opus), video codecs (e.g. H.264, VP8) and synthesis algorithms. It is also very commonly employed in streaming sessions where network effects can degrade communications quality.

5. SYSTEM DESIGN

Our project can be divided into the following stages and can be shown by the following system diagram:



Input Article
Parsing & Preprocesing
Sentence feature extraction
Clustering similar sentences
Sentence Selection (kNeigbors)
Extactive Summarization
Word feature embedding
Generating Abstractive Summary (BART)
Writing Markdown & Generating PDF
Converting pdf pages to images
Neural Speech Synthesis
Video Generation

Figure 5.1: Architecture

5.1. Parsing and Preprocessing

The parsing and preprocessing step is the initial stage in our pipeline. In this step, we collect articles from various sources such as news websites and scientific journals. Once we have the articles, we extract the relevant information like author, date, images, and text. We use various libraries like newspaper3k and pdfparser to extract text from the given URL or PDF. However, the extracted text may contain characters that are not in the UTF-8 format, which need to be preprocessed to obtain clean text.

To preprocess the text, we use techniques such as tokenization, stemming, and lemmatization. Tokenization involves splitting the text into individual tokens or words, while stemming and lemmatization involve reducing the word to its base form or root. These techniques help in removing noise and reducing the vocabulary size, making it easier to process and analyze the text.

5.2. Sentence Feature Extraction

The next step in our pipeline is to extract sentence-level feature embeddings from the text. We use the BART (Bidirectional and Auto-Regressive Transformer) model, which is pre-trained on a large corpus of data and then fine-tuned on the CNN/Daily Mail dataset [16]. The text is divided into sentence-level tokens and then fed into the encoder stage of the BART model. The output of the encoder stage provides the features and representation of each sentence in the text, which acts as the features of the sentence.

The features extracted in this step are crucial for the clustering and ranking processes. These features are used to measure the similarity between sentences, which helps in grouping them together in the clustering step and ranking them in the ranking step.

5.3. Clustering

Clustering is performed on the sentence features and embeddings using the K Medoids algorithm. The purpose of clustering is to group similar sentences together to form clusters based on the extracted features. We use the K Medoids algorithm instead of the K-means algorithm because K-means can be sensitive to outliers and noise that may be present in the input data, which can skew the clustering results.

Each cluster is then assigned a topic based on simple heuristics. The rule we use is based on the fact that the first sentence in the cluster will always be the background because of the nature of the article, which is written in an inverse pyramid style. The third cluster will most probably always contain the conclusion. The middle parts are where major details fall under.

5.4. Ranking

In this step, we rank the sentences in each cluster using the KNN (K-Nearest Neighbors) algorithm. The KNN algorithm ranks the sentences based on the features and embeddings extracted in the previous steps. We employ some heuristics to chop off some dissimilar sentences.

The ranking process is crucial because it helps in identifying the most important sentences in each cluster, which can then be used in the summarization step.

5.5. Summarization

After the ranking process, we perform abstractive summarization on each cluster using the BART model, which has been fine-tuned on the CNN/Daily Mail dataset. The generated sentences are written on points with 3 different sections. The input sequences vary based on the length of the cluster extracted but are generally kept below 512 words. The output words are generally 60-100 words per cluster.

Abstractive summarization involves generating a summary that is not a verbatim copy of any part of the original text. Instead, it captures the essence of the text and presents it in a concise and coherent manner. Abstractive summarization is a challenging task and requires sophisticated techniques such as natural language processing and machine learning.

5.6. Slide Generation

By using markdown syntax and MARP, we create slides from the presentation. Other information like authors, date and images are also included from the parsing steps and slide is generated.
To use MARP, users first write their presentation content using Markdown syntax. This involves using various formatting elements such as headings, bullet points, and images to create a visually appealing and engaging presentation. Once the content is written, users then use MARP to generate the HTML code or PDF needed to display the presentation in a web browser.

5.7. Speech Synthesis

Using Azure cognitive speech services, we perform speech synthesis on the abstractive text. For every slide chunk we perform this synthesis process using the Azure services [11].
This process involves sending a simple REST-based request to Azure Cognitive Sepech Service that can be easily integrated into any application or service. The API takes text input and generates an audio output in the form of an MP3 or WAV file. Different audio formats, sampling rates, and voice fonts are supported giving users flexibility to customize the speech output according to their needs.

5.8. Video Generation

We then convert the slides pdf to image frames and then, using FFMPEG, we concatenate the audio with frames and finally generate a video.
This is how the video is generated from FFMpeg:

- FFmpeg takes each image in sequence and converts it into a video frame.
- It then takes the corresponding audio clip and adds it to the video frame.
- This process is repeated for each image/audio pair in sequence.
- FFmpeg then concatenates all the resulting video frames into a single video file.
- Finally, it encodes the video in a desired format, such as MP4 or AVI.

5.9. Diagrams

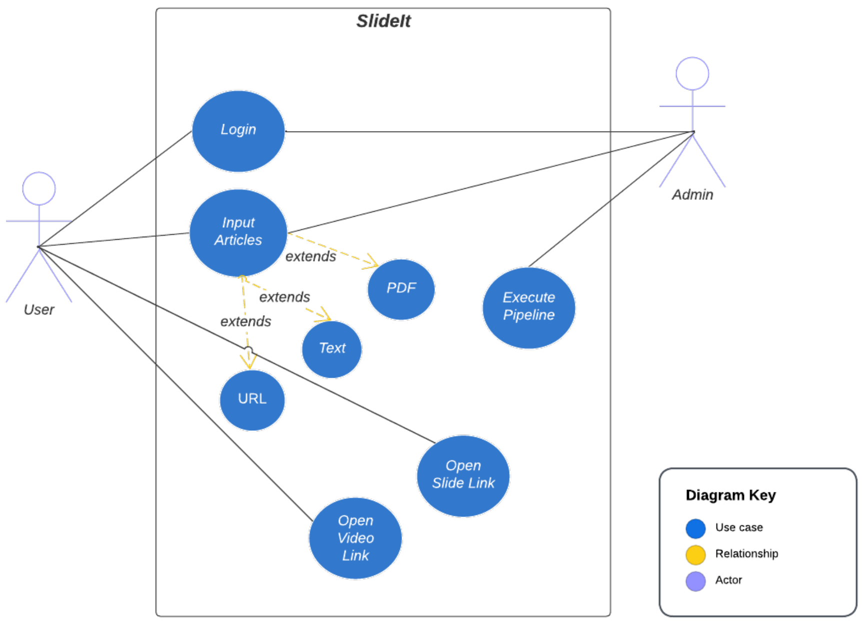


Figure 5.2: Use Case Diagram

The use case diagram gives us an overview of how we/user can interact with our web based application.The user can login first with their google accounts. They can feed text/paragraph, pdf or urls to the platform and get the link for slide and video generated.

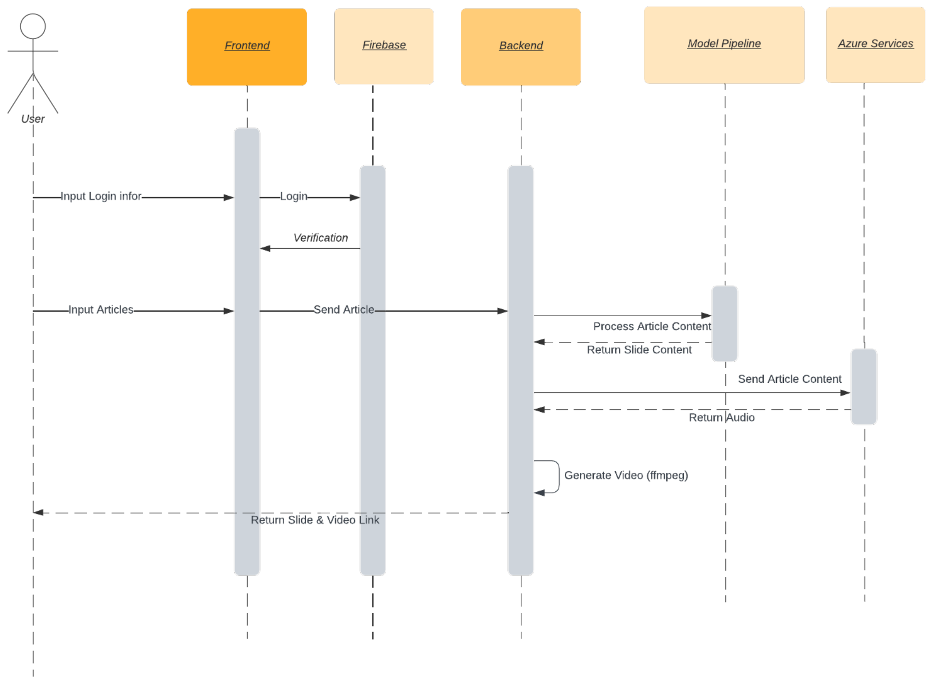


Figure 5.3: Sequence Diagram

The sequence diagram captures the interaction between objects in the context of collaboration. In this diagram ,we can see how the user interacts with front-end and backend side of the application. Whenever the user performs some operation, the objects interact with each other to give proper feedback.

5.10. Pipeline

We used Flask for the backend and React with Tailwind for the frontend of the web application. Flask is a Python web framework that allows us to build web applications quickly and easily. React, on the other hand, is a JavaScript library for building user interfaces. By using these tools, we were able to create a user-friendly web application that allows users to input raw data in the form of URLs, PDFs, and text.

The React web app sends the raw data to the backend for processing. This is where the whole pipeline, from preprocessing to final video generation, is performed. The pipeline processes everything and then sends the drive link to the frontend. The unique point of view of the project is the pipeline rather than the model and the dataset. This means that the pipeline is the main focus of the project, as it is what makes everything work together seamlessly.

To build the total pipeline, we have used various Python libraries, such as newspaper3k and pdfparser, for parsing the raw data. Parsing involves extracting relevant information such as author, date, and images from the input text. We also used PyTorch [17] for deep learning, which is a popular open-source machine learning library used for developing and training deep neural networks. We used the transformers library for our summarization model, which is a state-of-the-art language modeling library for natural language processing. Additionally, we used numpy for matrices, sklearn for machine learning algorithms like clustering and classification, azure-cognitive services for speech synthesis, pdf2images library for converting PDFs to image frames, and other third-party software such as marp for slide generation from markdown syntax and ffmpeg for video generation.

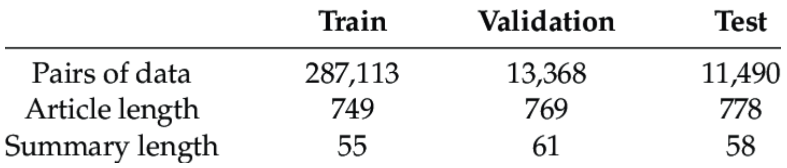
The pipeline is the heart of the project, and it consists of several steps. First, we perform text parsing from the raw data, extracting relevant information such as author, date, and images. Then, we extract sentence-level feature embeddings from the text using BART model fine-tuned on the CNN/DM dataset. Next, we cluster the sentence features and embeddings with K Medoids clustering algorithm into 3 clusters. Then, we perform on each cluster ranking with KNN algorithm, and we employ some heuristics to chop off some dissimilar sentences. After that, we perform abstractive summarization on each cluster using BART model finetuned on CNN/DM. The generated sentences are written on points with 3 different sections. Finally, we create slides from the presentation using markdown syntax and Marp and use Azure Cognitive Services for speech synthesis on the abstractive text. We then convert the slides PDF to image frames and concatenate the audio with frames using FFMPEG to generate the final video.

6. RESULTS & DISCUSSION

6.1. Dataset Quality

We used the CNN/DM dataset to fine-tune the BART model for summarization. The dataset is widely used for training summarization tasks on similar type of text i.e, Articles.
Datasets contains:

- 286,817 training pairs
- 13,368 validation pairs
- 11,487 test pairs.



	Train	Validation	Test
Pairs of data	287,113	13,368	11,490
Article length	749	769	778
Summary length	55	61	58

Figure 6.1: Dataset Summary

The article for the dataset is written in inverse pyramid style as discussed in the methodology. The quality of data is subjective, and the dataset is prepared by world class journalists, authors, reporters from CNN and Daily Mail.

Table 2: CNN/DM Details through words overlap metrics

Unigram	Bigram	Trigram	4-grams
17.00	53.91	71.98	80.29

Note that the overlap metrics are shown as percentages (%).

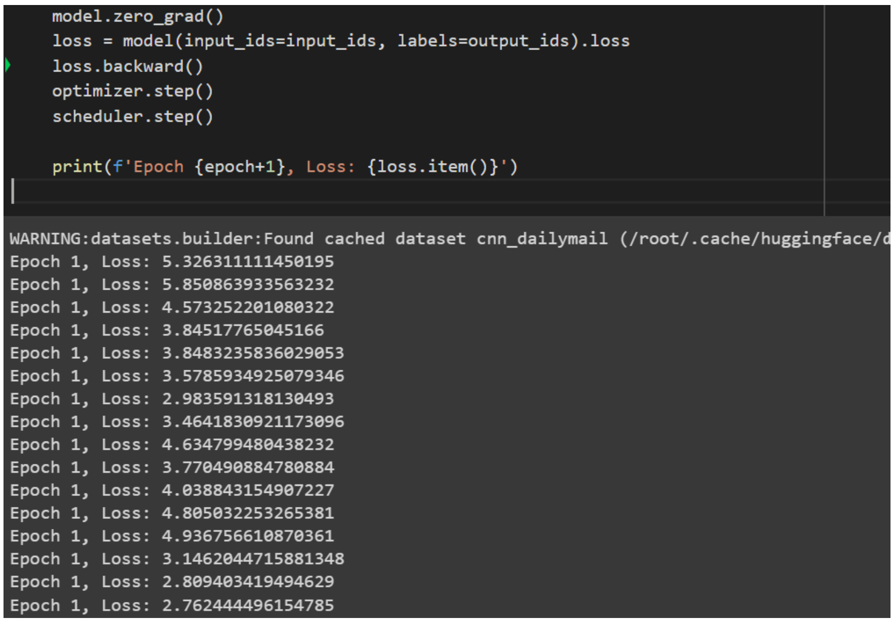
6.2. Finetuning objective

The model was initialized with pre-trained weights on the bart-base (139m params) as well bart-large (409m params) checkpoint. We used the latter model for final implementation as though it wasn't finetuned by us and was available at huggingface [18]. Only the BART base was tested but we have compared the result with BART large.

Here we perform tests to compare to BART-base and BART-large . The tokenizer was used to encode the input and target texts, and to pad or truncate them to a fixed length. We used the following hyperparameters:

- Batch Size: 2
- Learning Rate: 3e-05
- Maximum Sequence Length: 512
- Number of Epochs: 4
- Warmup Steps: 1000
- Adam Epsilon: 1e-08
- Gradient Accumulation Steps: 16
- Weight Decay: 0.01

Note that the maximum sequence length of 1024 is higher than the standard maximum sequence length of 512 for BART base, but is feasible with BART large due to its larger memory capacity. That's why we would use it in our project.



```
    model.zero_grad()
    loss = model(input_ids=input_ids, labels=output_ids).loss
    loss.backward()
    optimizer.step()
    scheduler.step()

    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

```
WARNING:datasets.builder:Found cached dataset cnn_dailymail (/root/.cache/huggingface/d
Epoch 1, Loss: 5.326311111450195
Epoch 1, Loss: 5.850863933563232
Epoch 1, Loss: 4.573252201080322
Epoch 1, Loss: 3.84517765045166
Epoch 1, Loss: 3.8483235836029053
Epoch 1, Loss: 3.5785934925079346
Epoch 1, Loss: 2.983591318130493
Epoch 1, Loss: 3.4641830921173096
Epoch 1, Loss: 4.634799480438232
Epoch 1, Loss: 3.770490884780884
Epoch 1, Loss: 4.038843154907227
Epoch 1, Loss: 4.805032253265381
Epoch 1, Loss: 4.936756610870361
Epoch 1, Loss: 3.1462044715881348
Epoch 1, Loss: 2.809403419494629
Epoch 1, Loss: 2.762444496154785
```

Figure 6.2: Training instance on BART-base with CNN/DM

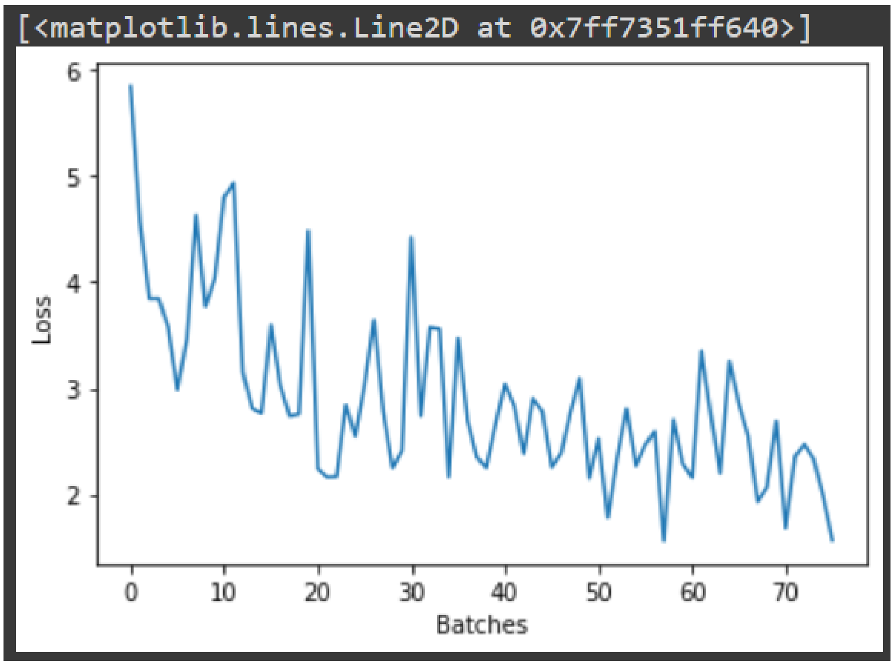

```
[<matplotlib.lines.Line2D at 0x7ff7351ff640>]
```

Figure 6.3: Sample Loss function for small batches (not complete)

6.3. Model Evaluation

We evaluated the performance of the BART large and BART base models on the CN-N/DM dataset. The BART-base was evaluated according to the training parameter mentioned in methodology. The BART-large was trained and evaluated by Facebook(now Meta) corporation.

Table 3: Comparison of ROUGE scores for BART and its variants on CNN/Daily Mail.

Model	ROUGE-1	ROUGE-2	ROUGE-L
BART base	44.30	21.94	41.05
BART large	45.35	22.27	41.59

ROUGE-1 measures unigram overlap, ROUGE-2 measures bigram overlap, and ROUGE-L measures longest common subsequence overlap. All scores are reported in percentage.

Based on this table, we can draw the following disucssions:

Reasons to use BART base over BART large

- BART base is a smaller model, so it may be more efficient to use in certain scenarios where computational resources are limited.

- BART base still performs well on the CNN/DM dataset, achieving ROUGE scores of 44.30, 21.94, and 41.05 for ROUGE-1, ROUGE-2, and ROUGE-L, respectively.
- BART base may be a better option for fine-tuning on smaller datasets, where the larger BART model may overfit.

Reasons to use BART large over BART base

- BART large generally performs better than BART base across all ROUGE metrics on the CNN/DM dataset.
- BART large may be a better option for fine-tuning on larger datasets, where the additional parameters in the model may be beneficial for achieving higher performance.
- BART large may be a better option for downstream tasks that require more complex language understanding and generation, such as question answering or dialogue systems.

6.4. Ranking Evaluation

For ranking (this process is performed before summarization task), we first carry out KMediods clustering which is an unsupervised machine learning algorithm therefore there is no reliable metric to evaluate the clusters. And for selecting the sentences we use, KNN. The evaluation of the ranking pipeline can be made with examples and cases.

Input Paragraph:

The future of space exploration requires big ideas, and NASA has no objection to considering some of the biggest ideas out there. The space agency's Innovative Advanced Concepts (NIAC) program exists for this very purpose, and it has chosen the next crop of concepts worthy of an initial study. The latest round of NIAC grants were awarded to 14 research teams, each receiving $175,000 to further develop their concepts, NASA announced yesterday. Of the 14, 10 are first-time NIAC recipients. These are all preliminary Phase I studies, which need to be completed within nine months. One of the country's most prominent academic centers that purports to "advance sustainability in animal agriculture" is almost entirely funded by industrial agriculture interests, new documents show.

Output Sentences (Using KMediods & KNN):

- The latest round of NIAC grants were awarded to 14 research teams, each receiving $175,000 to further develop their concepts, NASA announced yesterday.
- The future of space exploration requires big ideas, and NASA has no objection to considering some of the biggest ideas out there.
- One of the country's most prominent academic centers that purports to "advance sustainability in animal agriculture" is almost entirely funded by industrial agriculture interests, new documents show.

Output Sentences (Using KMeans & KNN):

- These are all preliminary Phase I studies, which need to be completed within nine months.
- The latest round of NIAC grants were awarded to 14 research teams, each receiving $175,000 to further develop their concepts, NASA announced yesterday.
- The future of space exploration requires big ideas, and NASA has no objection to considering some of the biggest ideas out there.

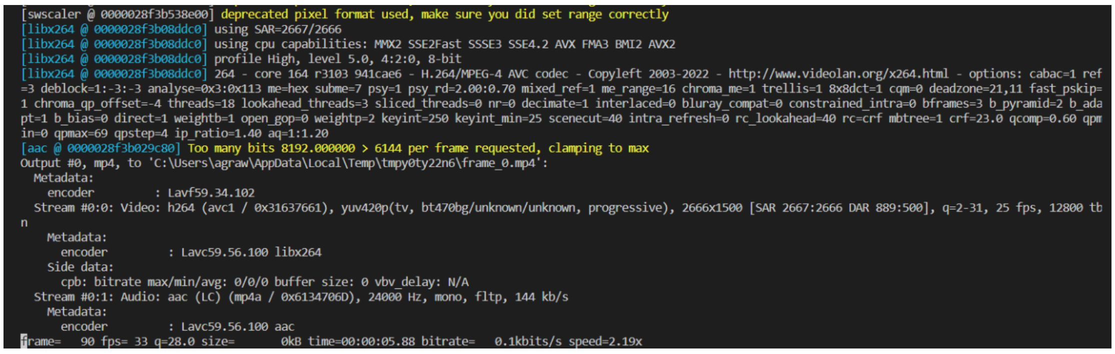
6.5. Azure Neural Speech Synthesis

Text-to-speech quality is measured by Mean Opinion Score (MOS), a widely-recognized scoring method for speech quality evaluation.

- For MOS studies, participants rate speech characteristics such as sound quality, pronunciation, speaking rate, and articulation on a 5-point scale.
- According to several MOS tests azure speech services have done (n>50 for each study), the average MOS score for the 15 new Neural TTS voices is above 4.1, about +0.5 higher than the scores for standard (non-neural) voices.

6.6. Generated Video Evaluation

The concatenation of the audio with the image sequences could be evaluated based on the synchronization of the audio with the visual content and the overall impact of the video in conveying the intended message. Concatenation is performed with FFMpeg.

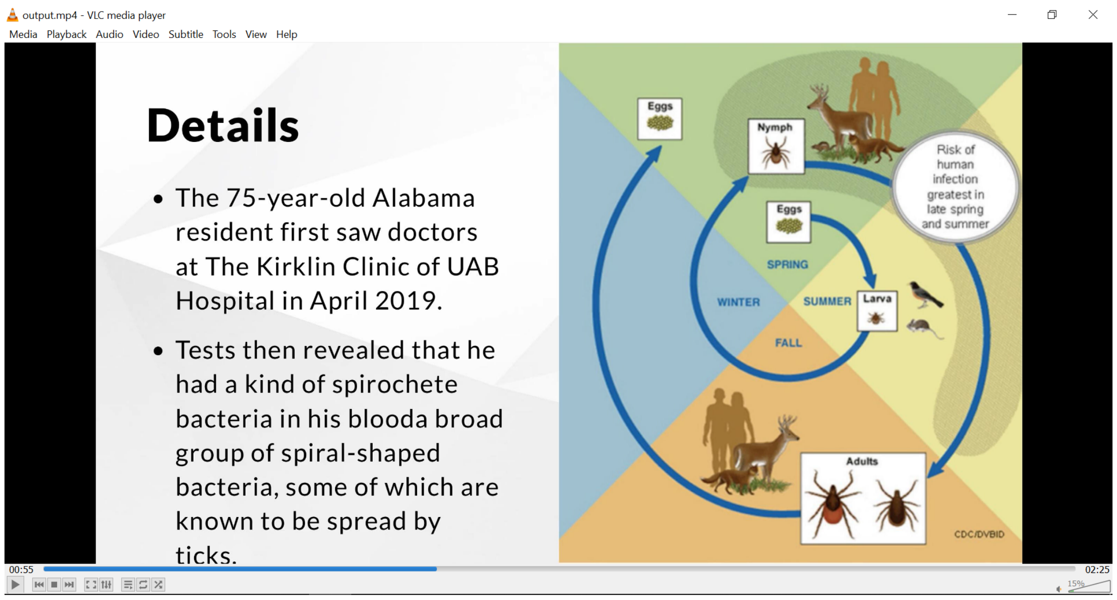


```
[swscaler @ 0000028f3b538e00] deprecated pixel format used, make sure you did set range correctly
[libx264 @ 0000028f3b08ddc0] using SAR=2667/2666
[libx264 @ 0000028f3b08ddc0] using cpu capabilities: MMX2 SSE2Fast SSSE3 SSE4.2 AVX FMA3 BMI2 AVX2
[libx264 @ 0000028f3b08ddc0] profile High, level 5.0, 4:2:0, 8-bit
[libx264 @ 0000028f3b08ddc0] 264 - core 164 r3103 941cae6 - H.264/MPEG-4 AVC codec - Copyleft 2003-2022 - http://www.videolan.org/x264.html - options: cabac=1 ref
=3 deblock=1:-3:-3 analyse=0x3:0x113 me=hex subme=7 psy=1 psy_rd=2.00:0.70 mixed_ref=1 me_range=16 chroma_me=1 trellis=1 8x8dct=1 cqm=0 deadzone=21,11 fast_pskip=
1 chroma_qp_offset=-4 threads=18 lookahead_threads=3 sliced_threads=0 nr=0 decimate=1 interlaced=0 bluray_compat=0 constrained_intra=0 bframes=3 b_pyramid=2 b_ada
pt=1 b_bias=0 direct=1 weightb=1 open_gop=0 weightp=2 keyint=250 keyint_min=25 scenecut=40 intra_refresh=0 rc_lookahead=40 rc=crf mbtree=1 crf=23.0 qcomp=0.60 qpm
in=0 qpmax=69 qpstep=4 ip_ratio=1.40 aq=1:1.20
[aac @ 0000028f3b029c80] Too many bits 8192.000000 > 6144 per frame requested, clamping to max
Output #0, mp4, to 'C:\Users\agraw\AppData\Local\Temp\tmpy0ty22n6\frame_0.mp4':
  Metadata:
    encoder         : Lavf59.34.102
  Stream #0:0: Video: h264 (avc1 / 0x31637661), yuv420p(tv, bt470bg/unknown/unknown, progressive), 2666x1500 [SAR 2667:2666 DAR 889:500], q=2-31, 25 fps, 12800 tb
n
    Metadata:
      encoder         : Lavc59.56.100 libx264
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: N/A
  Stream #0:1: Audio: aac (LC) (mp4a / 0x6134706D), 24000 Hz, mono, fltp, 144 kb/s
    Metadata:
      encoder         : Lavc59.56.100 aac
frame=   90 fps= 33 q=28.0 size=       0kB time=00:00:05.88 bitrate=   0.1kbits/s speed=2.19x
```

Figure 6.4: Frame Generation for each image sequence with concatenation with audio

The duration of the frame of video depends on the audio duration which is what we want. The frame change is possible only after the audio completes to generate the overall video. Around 3 minutes is used for speech/audio synthesis and video generation.

The video output is produced in mp4 format and can be previewed through drive link or media player as below :


output.mp4 - VLC media player

Media Playback Audio Video Subtitle Tools View Help

Details

- The 75-year-old Alabama resident first saw doctors at The Kirklin Clinic of UAB Hospital in April 2019.
- Tests then revealed that he had a kind of spirochete bacteria in his blooda broad group of spiral-shaped bacteria, some of which are known to be spread by ticks.

Figure 6.5: Output video generated

7. CONCLUSION

In conclusion, this project proposes an efficient and automated method for generating video slide presentations from input articles. The method involves text parsing and feature extraction from input articles, clustering and ranking algorithms to group and rank the extracted features, abstractive summarization using the BART model, slide creation using Markdown syntax and MARP, audio generation using Azure Cognitive Speech Services, and video concatenation using FFMPEG. Our evaluation on the dataset showed that the BART large model outperformed the BART base model in terms of rouge scores for summarization.

Although our system showed promising results, there are still areas for further exploration and improvement, such as evaluating the effectiveness of the system on different types of input articles, investigating different clustering and summarization algorithms to improve accuracy and quality, image positioning, postprocessing, deployment, and incorporating user feedback and preferences for a more personalized and user-friendly experience. Integrating the system with social media platforms or news aggregators could also make the summarization and presentation of news and information more accessible and convenient for users. Overall, our proposed method can significantly reduce the time and effort needed to create video slide presentations, and has the potential to make video content creation more accessible and efficient for a wider range of users.

8. LIMITATIONS AND FURTHER WORKS

8.1. Limitations

- Topic classification is basic utilizing the fact that article writing style is consistent i.e, inverse pyramid writing style.
- The images in the generated slides might not be properly position to the artilce.
- BART's performance can be affected by the quality of the training data and the fine-tuning process.
- The article length is limited to 1024 words, and can be extended to 3072 words. But the article with length greater than 3072 is not accepted by the sytem
- MARP has limited customization options for slide design and layout.
- FFMPEG can be slow when generating videos with a large number of frames.

8.2. Further Works

There are several areas for further exploration and improvement in our project. One area is to evaluate the effectiveness of our system on different types of input articles, such as academic papers or technical reports.

- Employ topic classification models to allow user to provide topic to which slides can be generated
- Expanding the domain of the articles so that any kind of writing style and publication can be converted to slides
- To enhance image positioning while parsing articles to generate slides with proper images.
- Incorporating user feedback and preferences into the system could lead to a more personalized and user-friendly experience.

REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on p. 1).

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: `1810.04805 [cs.CL]` (cit. on p. 5).

[3] Mike Lewis et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *CoRR* abs/1910.13461 (2019). arXiv: `1910.13461`. URL: `http://arxiv.org/abs/1910.13461` (cit. on p. 5).

[4] Anonymous Author. *BART Explained*. URL: `https://www.projectpro.io/article/transformers-bart-model-explained/553#mcetoc_1fq07mh0qe` (cit. on p. 5).

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on p. 9).

[6] Ashish Vaswani et al. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017) (cit. on pp. 11, 20).

[7] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: `2005.14165 [cs.CL]` (cit. on pp. 11, 17).

[8] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: `1409.3215`. URL: `http://arxiv.org/abs/1409.3215` (cit. on p. 20).

[9] L Kaufman and PJ Rousseeuw. "Clustering by means of medoids". In: *Statistical data analysis based on the L1-norm and related methods* 4.2 (1987), pp. 405–416 (cit. on p. 32).

[10] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: `10.1109/TIT.1967.1053964` (cit. on p. 35).

[11] Microsoft Azure. *Azure Text To Speech.* URL: `https://azure.microsoft.com/en-us/services/cognitive-services/text-to-speech/#overview` (cit. on pp. 46, 57).

[12] Michael Rubinstein, Ian Goodfellow, and David Iida. "Cross-entropy loss improves generative model". In: *arXiv preprint arXiv:1801.06146* (2018) (cit. on p. 49).

[13] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Internal Representations by Error Propagation". In: *Parallel Distrib. Processing: Explorations Microstructures* (1986), pp. 318–362 (cit. on p. 49).

[14] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *ArXiv* abs/1412.6980 (2015) (cit. on p. 50).

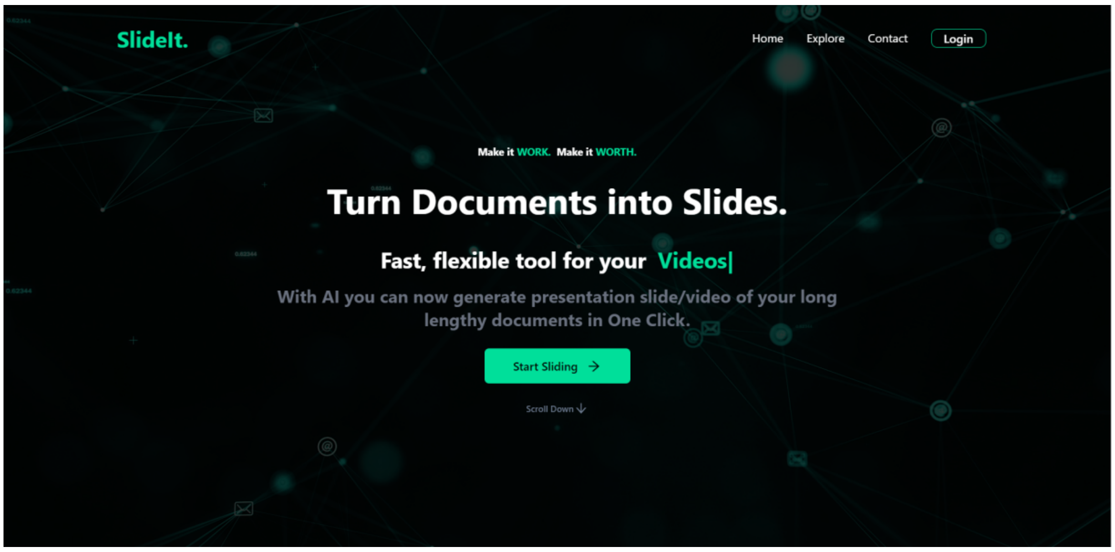
[15] Chin-Yew Lin. "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Workshop. Barcelona, Spain, 2004, pp. 74–81. URL: `https://www.aclweb.org/anthology/W04-1013` (cit. on p. 51).

[16] Sshliefer. *BART Documentation.* URL: `https://huggingface.co/transformers/v3.2.0/model_doc/bart.html` (cit. on p. 56).

[17] Sean Robertson. *NLP from scratch: Translation with Sequence to Sequence network and attention.* URL: `https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html` (cit. on p. 60).

[18] Sean Robertson. *Huggingface BART-CNN.* URL: `https:https://huggingface.co/facebook/bart-large-cnn` (cit. on p. 62).

9. APPENDIX



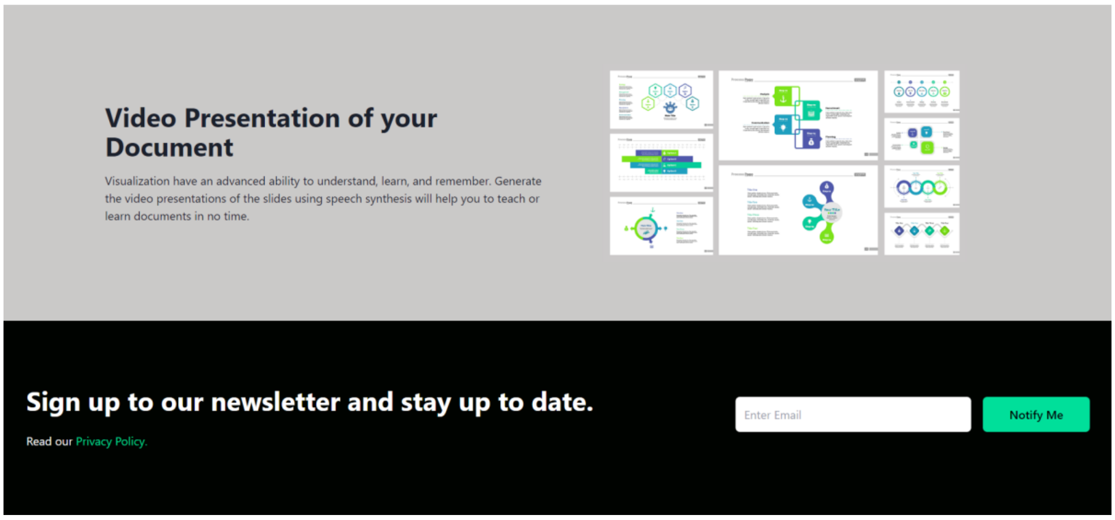
Make it WORK. Make it WORTH.

Turn Documents into Slides.

Fast, flexible tool for your Videos|

With AI you can now generate presentation slide/video of your long lengthy documents in One Click.

Figure 9.1: Landing page of the application



Video Presentation of your Document

Visualization have an advanced ability to understand, learn, and remember. Generate the video presentations of the slides using speech synthesis will help you to teach or learn documents in no time.

Sign up to our newsletter and stay up to date.

Read our Privacy Policy.

Figure 9.2: About page of the application

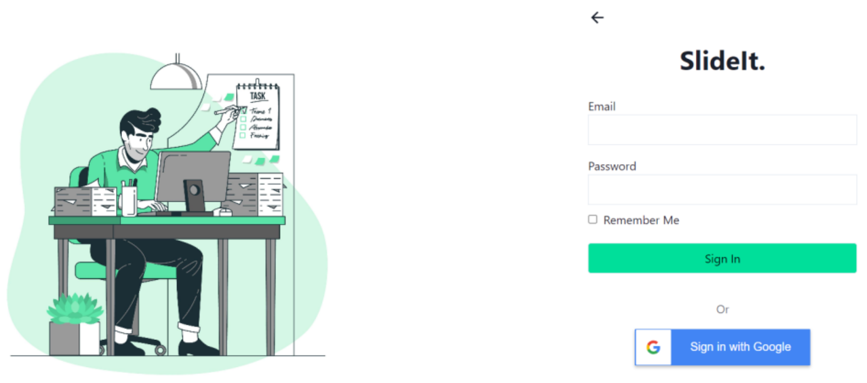


Figure 9.3: Login page of application

The user can login to the application using the Google SignIn Method

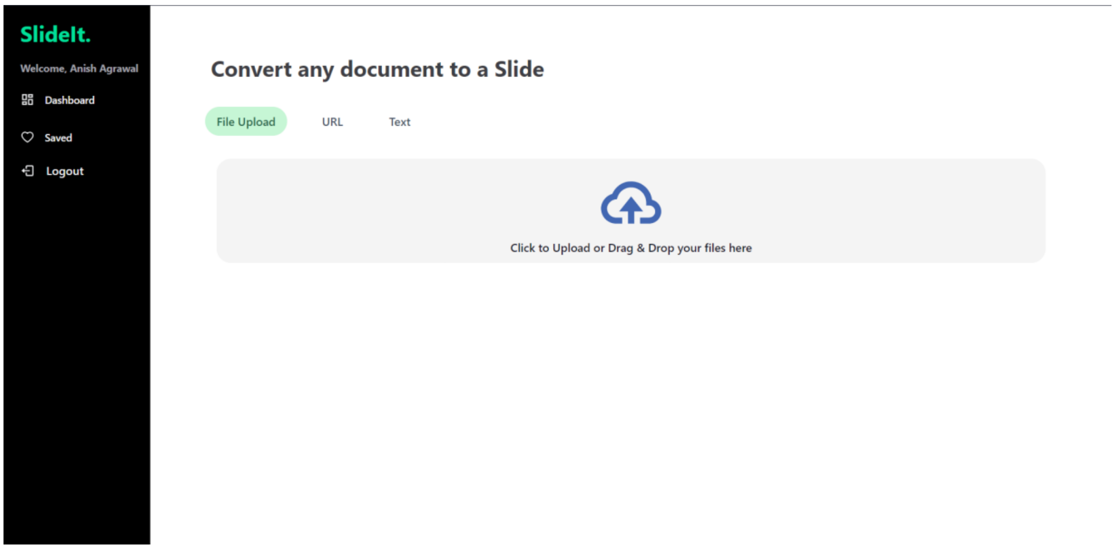

SlideIt.
Welcome, Anish Agrawal
Dashboard
Saved
Logout
Convert any document to a Slide
File Upload
URL
Text
Click to Upload or Drag & Drop your files here

Figure 9.4: Interface for input

The user can input articles in the form of PDF file upload, URL and Text

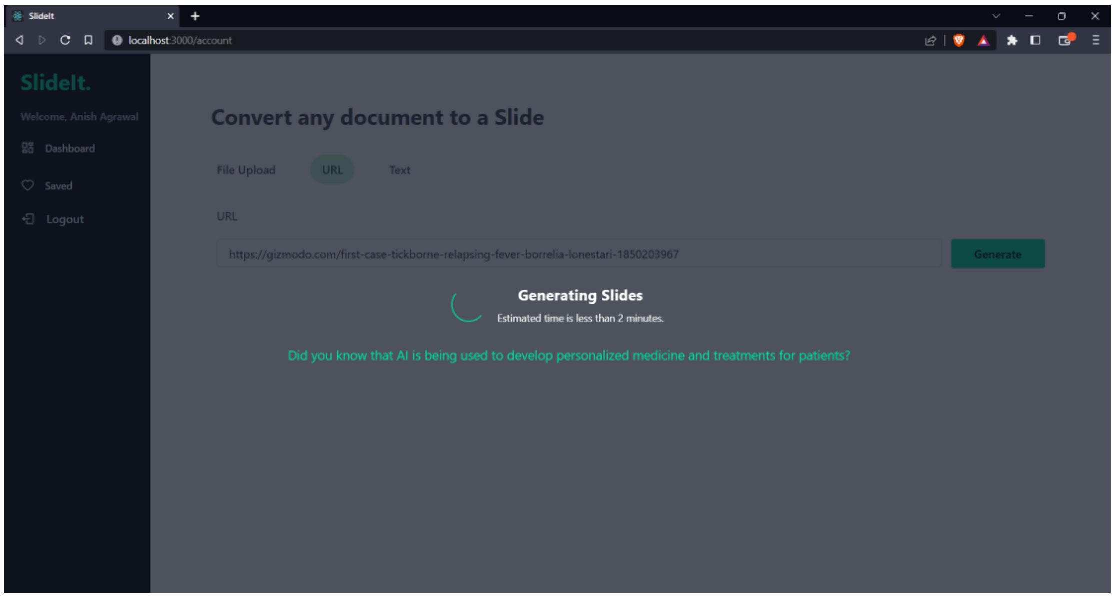

SlideIt
localhost:3000/account
SlideIt.
Welcome, Anish Agrawal
Dashboard
Saved
Logout
Convert any document to a Slide
File Upload
URL
Text
URL
https://gizmodo.com/first-case-tickborne-relapsing-fever-borrelia-lonestari-1850203967
Generate
Generating Slides
Estimated time is less than 2 minutes.
Did you know that AI is being used to develop personalized medicine and treatments for patients?

Figure 9.5: Output Generation Loading Screen

Once the Input article is submitted, the generating loading screen is displayed to the user and slide/video generation is done by the model pipeline in backend.

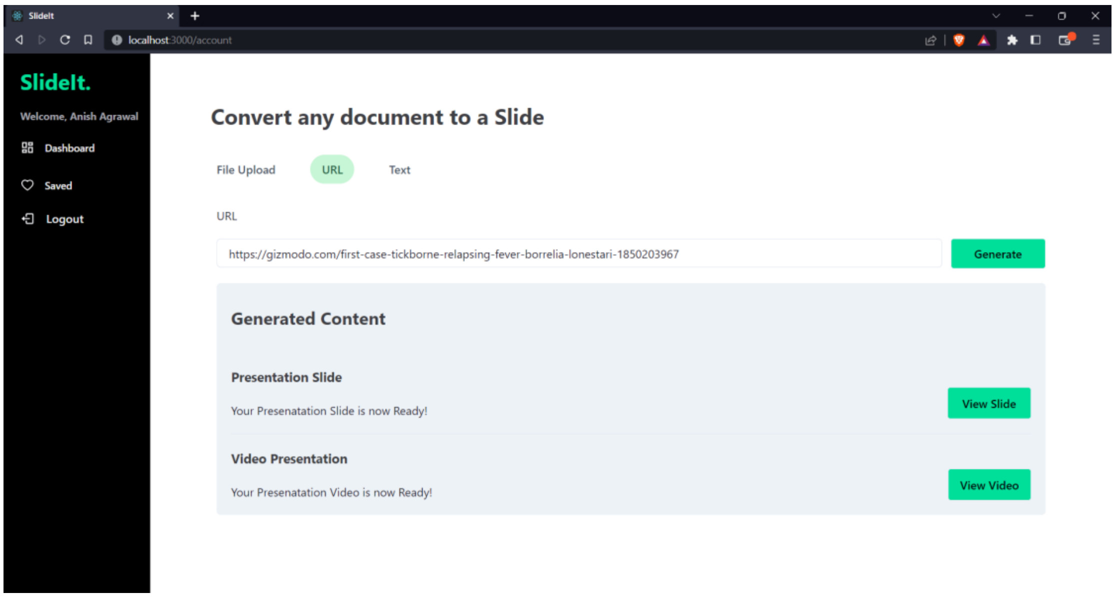

SlideIt
localhost:3000/account
SlideIt.
Welcome, Anish Agrawal
Dashboard
Saved
Logout
Convert any document to a Slide
File Upload
URL
Text
URL
https://gizmodo.com/first-case-tickborne-relapsing-fever-borrelia-lonestari-1850203967
Generate
Generated Content
Presentation Slide
Your Presenatation Slide is now Ready!
View Slide
Video Presentation
Your Presenatation Video is now Ready!
View Video

Figure 9.6: Generated Content of Slide and Video Link