UNIVERSITY OF BIRMINGHAM

EVENT-BASED MULTI-DOCUMENT TEXT SUMMARISATION OF NEWS ARTICLES

by

VIJAY JAWALI

A thesis submitted to the University of Birmingham for the degree of
M.SC. DATA SCIENCE

Supervisor: Dr. Michael Oakes
School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
September 2023

I confirm that the work was solely undertaken by myself and that no help was provided from any other sources than those permitted. All sections of the thesis that use quotes or describe an argument or concept developed by another author have been referenced, including all secondary literature used, to show that this material has been adopted to support my work.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BART Bidirectional Auto-Regressive Transformers. 2, 3, 7, 8, 10, 11, 24, 28, 29, 40, 42, 48, 50, 52–54

BERT Bidirectional Encoder Representations from Transformers. 7, 24

BLEU BiLingual Evaluation Understudy. 9, 26

CNN Convolutional Neural Network. 1

GloVe Global Vectors for Word Representation. 33

GPT Generative Pretrained Transformer. 8, 24

GRU Gated Recurrent Unit. 7

HAT Hierarchical Attention Transformer-based architecture. 7, 8

Hie-BART Hierarchical BART. 7

IDF Inverse Document Frequency. 14, 15, 31

KIGN Key Information Guide Network. 7

LDA Latent Dirichlet Allocation. 13, 14, 45

LLAMA Large Language Model Meta AI. 3, 9–11, 24, 25, 29, 37–39, 42, 49–54, 56

LoRA Low-Rank Adaptation. 8, 25, 38, 39, 56

LSA Latent Semantic Analysis. 3, 5, 9–11, 16, 17, 29, 32, 33, 42, 48–52

LSTM Long Short Term Memory. 1, 6, 17–20, 33, 34

MDS Multi-document Summarisation. 1, 2, 5, 10, 29

METEOR Metric for Evaluation of Translation with Explicit Ordering. 9, 11, 25, 26, 44, 51, 52, 55

NER Named Entity Recognition. 14, 27, 47

NNLM Neural Network Language Model. 6

PaLM Pathways Language Model. 8

PEFT Parameter Efficient Fine-Tuning. 38, 39

PEGASUS Pre-training with Extracted Gap-sentences for Abstractive SUmmarization Sequence-to-sequence models. 2, 7, 8, 54

ReLU Rectified Linear Unit. 23, 24

RNN Recurrent Neural Network. 1, 6, 7

ROUGE Recall-Oriented Understudy for Gisting Evaluation. 3, 6, 8, 9, 11, 25, 26, 29, 31–33, 35, 37, 39, 42, 44, 49–51, 55

SDS Single-document Summarisation. 1, 2, 10, 28

SMTP Simple Mail Transfer Protocol. 41

SPICE Semantic Propositional Image Caption Evaluation. 9

SVD Singular Value Decomposition. 3, 5, 16, 17, 32, 33

T5 Text-to-Text Transfer Transformer. 3, 7, 8, 10, 11, 21–24, 29, 36, 42, 48–54

TF Term Frequency. 14, 15, 31

TF-IDF Term Frequency - Inverse Document Frequency. 3, 4, 6, 10, 11, 14, 15, 17, 29, 31, 32, 42, 48–53

UniLM Unified Language Model. 7

Abstract

The spread of misinformation and disinformation poses a challenge for readers to efficiently obtain accurate overviews of news events. This project aims to develop an automated multi-document summarisation pipeline to address this issue. The system leverages credible news articles from established publishers and condenses the key information from multiple sources covering an event into a short, coherent summary. The project implemented and evaluated diverse extractive and abstractive text summarisation techniques, including statistical methods like Term Frequency-Inverse Document Frequency and Latent Semantic Analysis, graph-based algorithms such as TextRank, Encoder-Decoder neural networks with Pointer-Generator modules and Large Pre-trained Language Models like T5, BART and LLAMA-2. Both off-the-shelf models and variants fine-tuned on news data were tested. The models were systematically compared using the CNN/DailyMail news corpus as a benchmark and assessed on both quantitative metrics like ROUGE and METEOR, and human evaluations. The experimental results demonstrated that fine-tuned neural abstractive summarisation models significantly outperformed traditional extraction-based methods in generating coherent, relevant summaries. Adaptation of Large Language Models through supervised fine-tuning led to considerable gains over pre-trained versions. The project also created an innovative web dashboard with interactive features for Event, Topic and Custom text summarisation. The automated summarisation pipeline and comparative assessments effectively addressed challenges in producing accurate overviews of news events from credible multi-document sources.

Keywords: Multi-document Summarisation, Extractive Summarisation, Abstractive Summarisation, Transformers, LLAMA-2, Fine-tuning

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Michael Oakes, for his invaluable guidance, feedback, and support throughout my research. His immense knowledge and plentiful experience provided me direction and inspiration that greatly enriched my project experience.

I am grateful to all the professors in the Computer Science department for imparting their extensive subject matter expertise through thought-provoking coursework. The strong theoretical foundation equipped me for carrying out this research endeavor.

I greatly appreciate the participants who volunteered their time to contribute to my user studies. Their participation made an important part of my research possible.

I am forever indebted to my family for their unwavering love, patience, and belief in me. Their tremendous understanding and encouragement inspired me throughout my graduate studies. I could not have completed this thesis without their moral support.

Chapter 1

Introduction

Text summarisation is the task of generating a shorter version of a text document while preserving the critical information content and overall meaning. Automatic text summarisation is the process of creating a concise summary of a longer text document using computer software. The objective is to produce a concise summary highlighting the original text's main points and most important details. These methods utilise computational techniques such as machine learning, statistical modelling and natural language processing. Algorithms identify vital topics within the document and score sentences based on relevance to those topics. They analyse redundancy to remove repetitive information and simplify the language in the summary.

Potential applications for text summarisation include providing concise overviews of news articles, research papers, legal documents, online content and other lengthy texts. Summarisation allows for more efficient information processing by distilling key facts and conclusions without reading entire documents. Summarisation also has value for education by reducing learning materials for students, enhancing customer service through digests of consumer feedback and increasing web browsing speed through summarised web pages.

As automatic summarisation techniques continue to mature, the technology can potentially be adopted widely to help combat information overload across many real-world settings. Mainstream adoption will depend on progress in natural language processing to approximate human-level aptitude for identifying salient information.

1.1 Background

With the exponential growth of online content, it has become challenging for users to find and consume relevant news information efficiently. Multi-document Summarisation (MDS) addresses this issue by generating concise overviews of topics by extracting essential information from multiple documents. It combines and condenses textual data from multiple sources to produce more informative, thorough and less redundant summaries than Single-document Summarisation (SDS). Users can leverage automatically generated summaries to grasp the key facts and developments of news stories quickly.

The goal of automatically summarising text to distil key information dates to the early days of computer science research in the 1950s. One of the first published works was Luhn's 1958 paper "The Automatic Creation of Literature Abstracts" (Luhn, 1958), which proposed statistical methods for identifying significant words in a document and selecting key sentences.

In the 1990s, advances in natural language processing enabled more complex text analysis, spurring new summarisation approaches. Efforts were made to develop abstractive methods that could interpret and rephrase content. The SUMMARIST (Hovy et al., 1998) system in the late 1990s could condense technical papers using shallow semantic analysis. More advanced statistical approaches emerged during the same period, including Latent Semantic Analysis (Foltz, 1996) to capture semantic information. However, computational power was minimal. Most research focused on extractive summarisation based on the surface features of sentences.

From 2000 to 2010, statistical machine learning became dominant in summarisation research. Lexical features such as word frequency, sentence position, cue words and title overlap were commonly used. MDS, which condenses information across multiple sources, also gained interest. The coverage of topics and reduction of redundancy became important areas of focus.

In the 2010s, neural network methods like Recurrent Neural Network (RNN) encoder-decoders with attention revolutionised abstractive summarisation capabilities. Early adoption of sequence-to-sequence models with Long Short Term Memory (LSTM) and Convolutional Neural Network (CNN) demonstrated promising improvements in fluency and abstraction.

The "Attention is All You Need" paper (Vaswani et al., 2017), published in 2017 by researchers at Google, introduced the Transformer architecture for sequence modelling. This architecture is based entirely on self-attention

mechanisms rather than recurrent or convolutional neural networks. Modern abstractive summarisation models like BART and PEGASUS, which achieve state-of-the-art results, are based on the Transformer.

Over decades of research, text summarisation has evolved remarkably from simple statistical methods to complex neural network architectures. Driven by advances in natural language processing and deep learning, modern abstractive summarisation models can generate fluent, coherent summaries with paraphrasing capabilities. While recent progress is remarkable, challenges remain in accurately capturing semantics, reasoning and factual consistency. Continued research on novel architectures and training techniques will further advance summarisation capabilities to match human-level language understanding.

1.2 Problem Statement

Most significant news events are reported across multiple articles published at different times and from various media sources. Each article provides a subset of details and perspectives about the unfolding event. Readers interested in fully understanding an event need to thoroughly read through many long articles containing redundant and complementary information. This manual process of synthesising events from fragments across articles is extremely laborious and makes it difficult for readers to obtain a quick and comprehensive overview of the event.

Therefore, there is a clear need for an intelligent system that can automatically analyse collections of news articles reporting on the same event and produce a concise summarised description that integrates the most salient information from the various documents. Such summarisation capability would help readers efficiently grasp the key facts, timeline and themes about rapidly developing news events without laboriously reading multiple full-length articles. It can aid journalists and analysts in better tracking stories at scale as new reports continuously emerge over time around major events.

While existing algorithms for SDS can produce summaries of individual news articles, they do not provide a comprehensive understanding of a complex topic covered from various complementary perspectives across multiple document sources. This project aims to solve this challenge by merging the most critical information from diverse sources covering an event to generate summaries that are more thorough, holistic and integrated than any individual source alone.

1.3 Objective

As misinformation and fake news become more prevalent online, objectively understanding a topic or event from credible journalistic sources has become a critical need. Established news publishers, due to their long-standing history and reputation for accuracy can provide insightful coverage of events without bias or partisan agendas. However, with the 24-hour endless news cycle and multiple outlets, getting a concise summary of an event from these credible sources poses a challenge.

This project aims to develop an automated MDS pipeline to address this challenge. The system will leverage credible news articles from publishers and efficiently condense the key information from multiple sources covering the same event into a short, coherent summary. MDS will allow readers to quickly get an overview of the event from reliable, objective news coverage.

The summarisation pipeline will utilise neural network architectures and unsupervised learning techniques to identify and extract the salient information across news articles. The project will assess various methods for effectively generating single document summaries as modules within the pipeline. The approaches will cover a spectrum ranging from fundamental statistical features to state-of-the-art neural networks. The primary goal is not to create brand-new deep learning models but to utilise the most fitting and appropriate existing techniques.

The envisioned system will automatically generate a summary given a corpus of news articles discussing an event. The summary will synthesise the key details in a brief non-repetitive overview. This event-based Multi-document Summarisation of credible journalism aims to combat disinformation and provide readers with accurate summaries. The modular pipeline structure will allow the evaluation of different techniques for summarisation tasks.

1.4 Scope

The project focuses on extractive and abstractive summarisation of English-language news articles. Both statistical machine-learning approaches and neural network models will be explored. Users will have access to a dashboard application allowing them to select events and view MDS related to those events based on the chosen model.

1.4.1 Extractive Summarisation Methods

Extractive summarisation involves selecting and copying essential sentences or phrases from the original text to create a summary. This project will implement and evaluate several extractive summarisation techniques on the CNN/DailyMail news dataset. The first extractive method is Term Frequency - Inverse Document Frequency (TF-IDF), which scores sentences based on words with high term frequency and inverse document frequency in the corpus. Another technique is TextRank, a graph-based algorithm that computes sentence importance based on a similarity graph and iterates until convergence, like PageRank. Latent Semantic Analysis (LSA) is a third approach that will be explored, it uses Singular Value Decomposition (SVD) to uncover latent semantic relationships between words and sentences.

These unsupervised techniques rely on statistical and graph features to identify key sentences without training data. The methods will be implemented in Python and assessed using the standard ROUGE metrics to compare against reference summaries. Human evaluation will also be conducted to judge cohesion, redundancy and information coverage. These extractive approaches provide a simple baseline before exploring more advanced abstractive techniques.

1.4.2 Abstractive Summarisation Methods

Abstractive summarisation involves generating new sentences that capture the main meaning of the original text. Several abstractive summarisation models will be evaluated on the CNN/DailyMail dataset. A sequence-to-sequence architecture with a pointer generator network will be implemented to generate summaries with copying and paraphrasing. Pretrained Transformer encoder-decoder models like T5, BART and LLAMA will be leveraged for abstractive summarisation using both off-the-shelf and with fine-tuning.

For instance, the T5 model pre-trained on a large text corpus will be directly applied to generate summaries, then the pre-trained model will be fine-tuned on news articles and summary pairs from CNN/DailyMail to adapt it specifically for news summarisation. Similarly, LLAMA-2 model will be evaluated before and after CNN/DailyMail dataset fine-tuning. The models will be implemented in PyTorch and TensorFlow and evaluated using the ROUGE metric. Human assessment will also judge fluency, coherence and factual consistency.

1.4.3 Dashboard Application

A web dashboard application will be developed to showcase the summarisation techniques to users. The dashboard will feature an intuitive interface allowing users to browse news articles grouped into topics. Users can select a topic and view multi-document summaries of the latest articles generated by different methods. The application will also allow users to enter custom news articles to get summaries. Users can also compare summaries generated from various methods to evaluate quality. Analytics will be provided to compare precision, recall and f1 scores for ROUGE-1, ROUGE-2 and ROUGE-L metrics. Users will also have the capability to receive emails on a specific topic.

1.5 Structure of Dissertation

The remainder of the paper is structured as follows:

Chapter 2 provides a Literature Review of prior research in text summarisation. It explores early statistical, heuristic and graph-based approaches, recurrent neural network methods, and recent transformer models. Key findings on model architectures, training techniques and evaluation metrics are discussed. Current limitations and open challenges are identified.

Chapter 3 details the Methodology adopted in this project. It describes the dataset and pre-processing steps, exploratory data analysis, implementation of extractive models and abstractive models, evaluation metrics and the dashboard application design.

Chapter 4 documents the Implementation of the various models and techniques outlined in the methodology. It provides specifics on the libraries, frameworks, model architectures, training procedures, inference workflows and application development.

Chapter 5 presents the Results and Analysis. Both quantitative metrics and qualitative human evaluations are discussed. Comparisons are made between extractive and abstractive models as well as pre-trained versus fine-tuned versions.

Chapter 6 provides further Discussions and Interpretation of the results obtained. Relationships to prior work are highlighted. Current limitations and open challenges are acknowledged.

Finally, Chapter 7 concludes with a summary of key achievements, limitations, and potential future work to build on this research.

Chapter 2

Literature Review

Text summarisation can be approached using two main methods: extractive summarisation and abstractive summarisation. Extractive summarisation approach relies on identifying and extracting the most relevant information from the document or set of documents without rephrasing. On the other hand, Abstractive summarisation goes beyond simply extracting sentences and aims to create a summary that may not exist in the original text. Extractive summarisation has the advantage of preserving the accuracy of the original text, as it relies on verbatim extraction. However, it may result in a summary that lacks readability, since the extracted sentences are not rephrased. Abstractive summarisation, on the other hand, can generate more fluent and coherent summaries. However, it can be more challenging to ensure that the generated sentences accurately capture the central theme and meaning of the original text. Abstractive summarisation also requires additional skills such as paraphrasing, generalisation and assimilation of real-world knowledge to generate high-quality summaries. Text summarisation methods can also be categorised based on specific techniques used.

One such common technique used is statistical analysis (Moratanch et al., 2017). This approach involves using statistical and linguistic features to identify the most important sentences or phrases in the original text. Another approach is natural language processing, which involves using computational methods to analyse and understand the meaning of the text. This approach allows for more sophisticated analysis and representation of the text, generating summaries closer to human-style summarisation. Other techniques used in text summarisation include machine learning and deep learning. These methods utilise algorithms and models to analyse the text and generate summaries based on patterns and representations learned from large amounts of data. This literature review explores different approaches to text summarisation and key findings from previously published research materials.

2.1 Early Approaches

2.1.1 Statistical Methods

One of the earlier statistical approaches is Term Frequency - Inverse Document Frequency (Hans et al., 2016). This technique weights the importance of words in a sentence based on how often they appear in that sentence compared to the document. Words with high TF-IDF scores are considered informative for a document. The sentences containing more high-scoring words are selected for the summary.

The TF-IDF concept was first proposed by Karen Spärck Jones in a paper published in 1972. She introduced the notions of term specificity and term invariance to retrieve relevant documents (Sparck Jones, 1972). The goal was to automatically extract keywords that are descriptive of a document's content and could serve as a document surrogate for information retrieval. A decade later, the TF-IDF scheme was formalised and refined by Gerard Salton and Christopher Buckley (Salton et al., 1988). They utilised term frequency normalisation and inverse document frequency weighting to select keywords and index documents more effectively. This allowed them to design the Information Retrieval System using advanced vector space modelling and cosine similarity measures.

The summarisation task works by computing TF-IDF scores for words and assigning these weights to sentences containing those words, summaries can rank and extract the most salient phrases and sentences that reflect the major concepts in the original text (Jones, 1998; Radev, Jing, Styś, et al., 2004). This approach allows summarisation algorithms to focus the summary on core content rather than generic passages.

The limitation with TF-IDF is exposed when words with high frequency in a document get an overly large weight, even if they may be unimportant semantically. Many modifications and extensions to the TF-IDF scheme have been proposed to handle these limitations, such as incorporating normalisation factors, smoothing parameters, n-grams, concept hierarchies and semantic analysis. Despite some shortcomings, standard TF-IDF remains a dominant approach due to its efficiency, scalability and good empirical performance across many tasks.

Latent Semantic Analysis (Ozsoy et al., 2011) utilises Singular Value Decomposition (SVD), a matrix factorisation technique, to uncover latent semantic relationships between terms in a corpus. A term-document matrix is constructed containing word counts per document. Matrix decomposition derives a lower-dimensional approximation identifying associations between words and sentences. The system then automatically identifies related sentences for the summary.

The foundation of LSA is based on the distributional hypothesis of language, which states that words with similar meanings tend to occur in similar contexts (Harris, 1954). A seminal paper by Gong et al. (2001) first presented an LSA-based summarisation system that used SVD to reveal latent semantic structure from a document corpus. The authors showed that LSA outperformed baseline methods like term frequency heuristics on standardised news corpora.

A major strength of LSA for extractive summarisation is its unsupervised learning of semantic structure from the text. Without human annotation or external knowledge bases, LSA can leverage statistical patterns to model conceptual meaning and topic relationships from corpora alone (Evangelopoulos et al., 2012). This allows it to deal with synonymy and polysemy when identifying salient content to include in the summary. LSA also benefits from the dimensionality reduction of SVD, which filters out noise and reveals latent semantic dimensions efficiently with matrix operations (Martin et al., 2007).

A key limitation of typical LSA implementations is that they ignore word order and syntax, simply modelling word co-occurrence frequencies. This can limit understanding of semantic composition in phrases and sentences. Additionally, reducing dimensionality can potentially discard meaningful semantic relationships along with redundancies. Optimal tuning of low-dimensional subspace parameters (k) can be difficult. There are also criticisms that LSA semantic spaces rely more on term usage conventions rather than deep understanding (Landauer, 2007).

Despite these challenges, LSA's continued role as an insightful, unsupervised semantic analysis method means that it remains a valuable technique to study and understand within the field of natural language processing. The core principles and modelling approach of LSA are still relevant today.

2.1.2 Heuristic-based Methods

Early work on automatic text summarisation also focused on heuristic methods that relied on rules and algorithms rather than statistical training. These techniques use basic positional and lexical features of sentences to identify salient content. For example, the "Identifying Topics by Position system" by Lin and Hovy (1997) selected sentences occurring at the beginning of paragraphs and sections, which often express vital themes.

Edmundson (1969) proposed utilising additional indicators of salience beyond high-frequency words by including clue words, structural features like sentence location and overlap with document titles. The study also aimed to simplify and shorten the running time of the computer program by avoiding frequency-counting the entire text, resulting in a more efficient extracting system. This paper introduced a multifaceted view of sentence relevance for extraction.

Radev, Jing, and Budzikowska (2000) introduced MEAD, a Multi-document Summarisation system employing cluster centroids produced by topic detection to capture central themes across documents. Sentence selection using centroids helped reduce redundancy compared to single document extraction. They also proposed new utility-based evaluation metrics to assess responsiveness, coverage and readability.

Later heuristic models incorporated more semantic knowledge to improve coherence and cohesion. Barzilay et al. (1997) proposed a content-ordering approach that involved three main steps: segmenting the original text, constructing lexical chains using knowledge sources like WordNet, and extracting significant sentences. Lexical chains were built by selecting candidate words, finding related words based on semantic criteria from WordNet, and inserting them into chains. Chain strength was determined by factors like repetition, density, and length. The lexical chains helped identify the most salient content to extract into the final summary.

Saggion et al. (2002) built SumUM, a domain-independent heuristic summarizer that parsed text into semantic classes like concepts, instances, attributes to filter sentences conveying major topics. The system works by instanting indicative and informative templates representing important information. Indicative selection identifies topics using term distribution and matches between titles and templates. Informative selection expands topics based on patterns containing topic words and informative markers. Summaries are generated by sorting templates, merging information, and reformulating content from templates into abstract style. This approach aimed to identify salient content through shallow analysis and regenerate it into summary form.

These early data-driven approaches were still limited in generating high-quality summaries compared to modern neural abstractive techniques. Despite the rise of advanced models, simple heuristics utilising sentence location and length remain relevant for baseline benchmarks and as supplementary heuristics in cutting-edge summarisation systems. The fundamentals still have utility alongside complex neural approaches.

2.1.3 Graph-based Methods

Beyond statistical and heuristic approaches, representing the text as a graph network and modelling the connections between words and sentences has shown to be a valuable approach for extractive summarisation. Graph-based algorithms treat the text as nodes of words and sentences linked by edges that capture similarity relationships. Once this graph is constructed, graph centrality measures can be applied to identify the most prominent, interconnected nodes representing salient content to extract for the summary. Using network representations and relationships, rather than just isolated features, provides a holistic way to assess the key topics and themes within the text for summarisation.

LexRank method was proposed by Erkan et al. (2004) as an unsupervised, extractive approach. It computes sentence importance based on eigenvector centrality on a similarity graph. Cosine similarity between sentence vectors defines the graph edges. It then calculates eigenvector centrality using power iteration to iteratively update sentence importance scores based on the scores of their neighbouring nodes. This process computes a continuous LexRank score between 0 and 1 for each sentence, with higher scores indicating higher salience. The top-ranked sentences up to the desired summary length are extracted for the summary.

LexRank showed strong results on DUC-2004 data[1], outperforming other systems. While LexRank remains widely used, it has some inherent limitations as a purely extractive method. The graph representation does not explicitly model topics or higher-level document structure. Research shows LexRank performs well on short documents but can struggle with redundancy for longer documents.

TextRank (Mihalcea et al., 2004) is a graph-based ranking algorithm used for ranking text. Words or sentences are represented as nodes in a graph, with edges connecting similar nodes. PageRank is then applied to this graph to update relevance scores for each node until convergence. The top-scoring nodes are extracted for the summary. Modelling relationships between sentences helps identify central topics and themes. Due to its generality, TextRank has also been widely used in real-world applications across domains like news, legal cases, meetings, and social media. Independent comparative assessments show TextRank matching or exceeding the performance of other extractive methods across standard datasets based on both automatic ROUGE metrics and human evaluations. The algorithm's graph representation of textual units allows important sentences to be selected based on recursive global importance scores on the graph.

LexRank relies solely on sentence connectivity within the graph, computed using cosine similarity between sentence vectors. In contrast, TextRank also incorporates additional metrics like word co-occurrence and TF-IDF weighting through the edge weights in its graphs. A main weakness of LexRank is that it does not scale well to longer documents. As sentences get further apart in a large document graph, their connectivity decreases, making LexRank less effective. TextRank mitigates this issue by using sentence-level graphs, word relationships persist even between distant sentences.

2.2 Neural Network Approaches

With the rise of deep learning since the 2010s, neural network architectures have driven rapid progress in abstractive text summarisation capabilities. Sequence-to-sequence models marked an important breakthrough, leveraging encoder-decoder architectures with attention mechanisms to generate summaries. Sequence-to-sequence neural networks transform sequences from one domain to another using encoder-decoder architecture. The encoder reads the input sequence and encodes it into a fixed-length vector representation. The decoder then uses this vector to generate the output sequence one token at a time.

2.2.1 RNN-based Sequence-to-sequence Models

Early seq2seq models were based on recurrent architectures, particularly RNN and LSTM networks. Recurrent Neural Networks are a type of artificial neural network well-suited for processing sequential data. RNNs have cyclic connections that allow information to persist across time steps in the form of hidden states. This enables modelling temporal dynamic behaviour for sequence tasks and are suitable for processing text data. LSTM were introduced by Hochreiter et al. (1997), these networks overcame issues with vanishing gradients in standard RNNs through the use of memory cells and gating units, allowing them to be modelled for long-range dependencies critical for summarisation.

A pioneering seq2seq model for abstractive summarisation was proposed by Rush et al. (2015). Their architecture comprised a Bag of Words encoder to read the source text and an NNLM (Bengio et al., 2003)

[1]DUC stands for Document Understanding Conference, an annual summarisation evaluation workshop organized by (https://www.nist.gov/). DUC 2004 dataset was used in the summarisation task of the DUC 2004 workshop. It contains 50 document clusters, each containing 10 news articles on a common topic.

decoder to generate summary sentences. Attention mechanisms were incorporated to allow the decoder to focus on relevant parts of the encoder's hidden states while generating each word. This data-driven end-to-end approach with neural attention achieved promising single-sentence summarisation results, outperforming extractive methods with a ROUGE-1 score of 29.78 on the DUC-2004 dataset.

Nallapati, Zhou, et al. (2016) built on these initial seq2seq efforts using attentional RNNs, pointer-generator networks (Vinyals et al., 2015) and separate abstraction and extraction phases. Their models could create concise summaries with paraphrasing and generalisation while retaining key information from the original text. The authors address critical problems in summarisation that are not adequately modelled by the basic architecture such as modelling keywords, capturing the hierarchy of sentence-to-word structure and omitting rare or unseen words at training time. On the CNN/DailyMail dataset, the pointer-generator model achieved a 35.46 ROUGE-1 F1 score showcasing the progress in abstractive methods.

Chopra et al. (2016) incorporated conditional RNN, which generates a summary of an input sentence by focusing on the appropriate input words using a convolutional attention-based encoder. This improved coherence and relevance compared to prior attempts. Their model produced highly abstractive summaries competitive with extractive baselines on the DUC-2004 dataset, attaining a ROUGE-1 score of 28.97. The paper's main contribution is the novel convolutional attention-based conditional RNN model for abstractive sentence summarisation.

Nallapati, Zhai, et al. (2017) proposed SummaRuNNer, a two layer bi-directional GRU-RNN based sequence model that took a hybrid extractive-abstractive approach. A convolutional sentence extractor first identified salient sentences represented by RNNs, eliminating the need for handcrafted features. At the decoder stage, an RNN with attention generated an abstractive summary focusing solely on those sentences identified at the encoder stage. This approach achieved a 39.60 ROUGE-1 F1 score, showing state-of-the-art results on the CNN/DailyMail corpus.

Beyond RNN architectures, Li, Xu, et al. (2018) employed a hybrid convolutional and recurrent model with keywords encoding. The extractive method is first used to obtain keywords from the text, which are then encoded into key information representation by the Key Information Guide Network (KIGN). The convolutional encoder learned local features and dependencies. The RNN decoder with attention, then generated the abstractive summary. The KIGN is integrated into the abstractive model to guide the generation process, specifically through the attention and pointer mechanisms. This convolutional component improved capturing relationships within sentences. The overall model could produce informative yet concise one-sentence summaries. The experimental results on the CNN/Daily Mail dataset demonstrate significant improvements achieved by the proposed model with a ROUGE-1 score of 38.95 on the CNN/DailyMail dataset.

In a few years, neural seq2seq models drove rapid progress in fluent abstractive summarisation. RNN encoders, decoders and attention mechanisms proved to be critical innovations that enabled going beyond purely extractive methods. Later works have been built upon these fundamental architectures to address challenges like repetition, irrelevance and factual consistency.

2.2.2 Transformer-based Sequence-to-sequence Models

Bidirectional Encoder Representations from Transformers (BERT) revolutionised natural language processing through its masked language model pretraining objective (Devlin et al., 2019). For summarisation, BERT is used as an encoder to generate contextual sentence embeddings of the input text. These rich representations better capture semantics useful for identifying salient information. BERT encoders can be paired with simple classifiers or regressors to perform extractive summarisation, achieving solid results like 44.41 ROUGE-1 F1 score on the CNN/DailyMail dataset (Liu et al., 2019). Models like UniLM (Dong et al., 2019) fine-tune BERT for summarisation by adding prediction layers. They outperform earlier models by leveraging BERT's bidirectional context.

The rapid progress in model scale has led to a new paradigm of utilising gigantic neural networks with billions of parameters for natural language processing tasks. Leveraging such vast model capacity has shown promising results for text summarisation. Full encoder-decoder architectures like BART (Lewis et al., 2020), T5 (Raffel et al., 2020) and PEGASUS (Zhang et al., 2020) achieve state-of-the-art performance on abstractive summarisation benchmarks. Pretraining on large corpora provides critical capabilities for summarisation like paraphrasing, text generation and modelling long-range dependencies. The Transformer foundation has been key to recent summarisation advances.

BigBird-Pegasus (Zaheer et al., 2021) extends the PEGASUS approach by incorporating sparse attention. It replaces the full Transformer self-attention with a sparse variant (BigBird) to handle longer sequences. The encoder and decoder are still pre-trained with gap sentences. BigBird-Pegasus achieved similar ROUGE scores to PEGASUS on CNN/DailyMail summarisation but with faster training and inference. The sparse attention reduced complexity from quadratic to linear, allowing summarisation of documents with length eight times longer. More recent models like HAT-BART (Rohde et al., 2021) and Hie-BART (Akiyama et al., 2021) extend this

with hierarchical Transformer architectures to handle long documents. HAT achieved state-of-the-art results on scientific paper summarisation of PubMed dataset, outperforming previous models. The hierarchical attention ensures key information is retained from long documents.

Recent massive language models like Generative Pretrained Transformer (GPT) models excel at natural language generation through autoregressive decoding. Formulating summarisation as a text-infilling task, GPT can generate abstractive summaries conditioning only on the context (Radford et al., 2019). Fine-tuning further improves coherence and relevant content selection. Large GPT variants like GPT-3 (Brown et al., 2020), TuringNLG (Rosset, 2020) and PaLM (Chowdhery et al., 2022) show few-shot summarisation[2] abilities. However, summaries still lack coherence and factual consistency compared to state-of-the-art supervised models. Task-specific fine-tuning on labelled summarisation datasets can provide significant performance improvements.

2.2.3 Fine-tuning Large Language Models

While pre-trained abilities are essential, large language model performance can be substantially enhanced through supervised fine-tuning on in-domain labelled summarisation data. Task-specific adaptation remains crucial even for models with hundreds of billions of parameters.

PEGASUS model proposed by Zhang et al. (2020) introduces a new self-supervised pretraining objective tailored for abstractive summarisation. The model is pre-trained on massive text corpora by generating summary-like gap sentences from documents with certain sentences removed or masked. When fine-tuned on downstream summarisation datasets, PEGASUS achieves state-of-the-art ROUGE scores across 12 diverse domains including news, science, stories, emails and bills with CNN/DailyMail dataset in particular showing a ROUGE score of 44.17. The model also shows strong low-resource summarisation capabilities with only 1000 examples. PEGASUS demonstrates the effectiveness of pretraining objectives explicitly designed for summarisation instead of only general language modelling.

Full fine-tuning becomes infeasible as pre-trained language models scale to hundreds of billions of parameters. A recent method called Low-Rank Adaptation (LoRA) (Hu et al., 2021) is a parameter-efficient tuning approach that adapts only a tiny fraction of model weights for downstream tasks. LoRA injects low-rank decomposition matrices into each layer of a Transformer model, keeping base weights frozen. Only these lightweight adapter modules are trained on the end task. LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by three times compared to full fine-tuning.

Despite having fewer trainable parameters, LoRA performs on par or better than fine-tuning in model quality on various language models. It provides tunable control over the tradeoff between efficiency and quality - smaller rank adapters are faster to train but may impact accuracy. Parameter-efficient tuning techniques will be critical to leverage ever-growing model sizes and demonstrates that full-weight adaptation is unnecessary; instead, small injected modules can steer pre-trained knowledge towards downstream tasks effectively.

2.3 Recent Advances and Trends

Recent advancements in deep learning techniques and the availability of extensive corpora have significantly enhanced the performance of abstractive text summarisation (Ma et al., 2022; Hou et al., 2021). Sequence-to-sequence models comprising an encoder-decoder LSTM architecture with attention have become a dominant approach. The encoder reads and builds representations of the input text and the decoder generates the summary sequence. Attention enables focusing on the most relevant encoder outputs.

Pointer-generator networks (See et al., 2017) enhance seq2seq models by allowing dynamic copying of words from the source text via pointing and generating words from a fixed vocabulary. This improves the accuracy and handling of out-of-vocabulary words. Large pre-trained language models like BERT have also been applied to summarisation, achieving strong performance when fine-tuned on domain datasets.

T5 (Raffel et al., 2020) and BART (Lewis et al., 2020) are two such pre-trained models that have delivered state-of-the-art results on text summarisation benchmarks. T5 consists of an encoder-decoder transformer architecture pre-trained on a large text corpus using a self-supervised objective. For summarisation, T5 is further fine-tuned on CNN/DailyMail (See et al., 2017) datasets which include document-summary pairs. Fine-tuning adapts T5 to generate summaries given new text input.

BART (Lewis et al., 2020) also employs sequence-to-sequence pre-training using a denoising autoencoder objective. BART corrupts text with masking, token deletion and sentence permutation and learns to reconstruct the original. Fine-tuning on summarisation outperforms previous models, showing the advantage of bidirectional pre-training. Without task-specific fine-tuning, the models struggle to produce coherent summarisations.

[2]Few-shot summarisation involves generating summaries of documents by fine-tuning a pre-trained model on only a small number of examples

LLAMA (Touvron, Lavril, et al., 2023) models leverage knowledge-enhanced pre-training on an external commonsense knowledge graph and unlabelled text corpora. For instance, LLAMA-2 (Touvron, Martin, et al., 2023) incorporates contextualised knowledge graph[3] embeddings learned using transformer encoders. It can produce relatively good summaries without fine-tuning, indicating useful world knowledge incorporation. However, fine-tuning LLAMA-2 on domain summarisation data can provide dramatic performance improvements, showing the importance of adaptation even for knowledge-enhanced models.

2.4 Automatic Evaluation Metrics

Evaluating text summarisation systems is challenging as multiple valid reference summaries can exist for a given input. A range of automatic metrics have been developed to compare system outputs to human references.

BiLingual Evaluation Understudy (BLEU) was originally proposed by Papineni et al. (2002) for the automated evaluation of machine translation output. BLEU operates by comparing the n-gram overlaps between a candidate translation and human reference translations. It computes a precision score for n-grams of varying sizes up to 4-grams, checking how many word sequences in the candidate translation match sequences in the references. BLEU gained widespread early adoption for its simplicity and speed. However, its applicability for evaluating text summarisation systems has limitations. This is because BLEU only considers precision - the fraction of matching n-grams out of the total in the candidate summary. It does not account for recall, which measures how much of the content from the references is covered by the candidate. This imbalance between precision and recall makes BLEU less suited for assessing summarisation, where both coverage of salient information and concision are crucial. While initially popular, BLEU has gradually seen a decrease in adoption as an evaluation metric for text summarisation, overcome by metrics like ROUGE and METEOR which address some of its limitations.

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) (Lin, 2004) is the most widely adopted automatic evaluation metric for summarisation. To gauge quality, it measures the overlap of common sequences between system and reference summaries. ROUGE measures lexical overlap between system and reference summaries by comparing n-grams, word sequences and word pairs. Common variations are ROUGE-1 (unigrams), ROUGE-2 (bigrams) and ROUGE-L (longest common subsequence).

Semantic Propositional Image Caption Evaluation (SPICE) was originally proposed by Anderson et al. (2016) for evaluating the quality of automatically generated image captions. While initially designed for this purpose, some researchers have explored adapting SPICE for evaluating text summaries as well. The key idea behind SPICE is that it compares semantic graphs constructed from the candidate summary and reference summaries. These graphs represent the objects, attributes and relationships between entities contained in the summaries. By matching the semantic propositions in the graphs, SPICE aims to assess the factual and logical correctness of a summary beyond simply matching n-grams. This is a potential advantage over lexical overlap focused metrics like ROUGE and BLEU. However, significant challenges arise in accurately constructing graphs from free-form textual summaries. This requires complex natural language processing and technical solutions that pose practical difficulties. Errors or missing elements in automatically constructed graphs from summaries can adversely impact the evaluation. While the theoretical appeal of matching semantic propositions is promising, the feasibility of generating high-quality scene graphs from text remains a hurdle for wider adoption.

AutoSummENG is an automatic evaluation metric for summarisation proposed by Giannakopoulos et al. (2011). It utilises LSA to compare the semantic similarity between a candidate summary and reference summaries. AutoSummENG uses LSA to convert the candidate summary and reference summaries into vector representations. It then measures the similarity between these vector representations by computing the cosine similarity between them. A higher cosine similarity score indicates greater semantic overlap and similarity between the summaries. An advantage of AutoSummENG is that it assesses semantic content matching and not just lexical or syntactic similarity, which metrics like ROUGE focus on. While useful for assessing semantic overlap, AutoSummENG has limitations arising from its ignorance of other linguistic qualities that make a good summary.

Metric for Evaluation of Translation with Explicit Ordering (METEOR) (Banerjee et al., 2005) was proposed as an automatic evaluation metric to improve correlation with human judgments over metrics like BLEU and ROUGE. METEOR compares system and reference summaries by aligning unigrams based on exact, stemmed and paraphrase matches using WordNet synonyms. It accounts for semantic similarity lacking in n-gram overlap metrics. Early work found METEOR correlates better with human assessments for machine translation (Lavie et al., 2007).

[3] A knowledge graph in the context of large language models refers to an external structured knowledge source that enhances the knowledge and reasoning capabilities of LLMs (Pan et al., 2023).

2.5 Gaps in Current Research

The landscape of text summarisation techniques has rapidly evolved from relatively simple statistical approaches developed initially to sophisticated deep learning models today.

Current state-of-the-art neural abstractive models still struggle with core capabilities needed for high-quality multi-document summarisation. This includes properly contextualising information, minimising repetition and generating novel phrases through abstraction while preserving semantic accuracy. Additionally, Multi-document Summarisation poses unique challenges compared to Single-document Summarisation. It requires identifying and consolidating the key information across multiple source texts covering the same topic. This is more difficult than summarising a single document due to increased information redundancy and the need to synthesise content from diverse documents.

This project seeks to advance the current research by comprehensively evaluating these diverse set of cutting-edge and traditional models using a standardised benchmark on news datasets and creating a practical application that can address the shortcomings and harness the potential of multi-document summarisation, thereby contributing to a more effective information comprehension and decision-making process.

The scope of this project focuses on news articles given limited research on multi-document summarisation of long-form content. The project builds directly on key findings from the literature review.

For extractive methods, TF-IDF, TextRank and LSA are included because the literature review showed that extractive summarisation techniques based on statistical, graph and matrix factorisation approaches were some of the effective summarisation techniques. As unsupervised methods, they provide a strong baseline and contrast to compare against more advanced abstractive techniques. TF-IDF leverages word frequencies, TextRank uses graph centrality and LSA employs latent semantic analysis to identify salient sentences, demonstrating distinct approaches to summarisation.

For abstractive summarisation, sequence-to-sequence models with pointer generator networks were a breakthrough that enabled the generating of novel summaries by copying words from the source text. Pre-trained Transformer encoder-decoder models like T5, BART and LLAMA-2 represent the current state-of-the-art in abstractive summarisation on large datasets based on the literature. T5 uses a self-supervised pre-training objective, while BART uses denoising autoencoding. LLAMA-2 incorporates external knowledge through pre-training on a commonsense graph.

These models are chosen because the literature review showed their strengths for abstractive summarisation. T5 and BART have achieved impressive results across many datasets and domains. LLAMA-2 model is chosen because it is the latest Large Language Model released incorporating real-world knowledge through pre-training on an external commonsense knowledge graph. This linkage to structured factual knowledge will demonstrate novel capabilities for generating abstractive summaries grounded in real-world facts. Fine-tuning them on a news dataset will further provide insights into their capabilities for multi-document summarisation of long-form articles, which remains an open challenge.

Chapter 3

Methodology

This chapter delves into the systematic processes adopted to achieve effective text summarisation of news articles. Beginning with the step of Data Collection, we address the sources from which data is gathered and the pre-processing techniques used to prepare the data for further analysis. An in-depth Exploratory Data Analysis is then conducted, containing sentiment analysis, topic modelling and named entity recognition to get deeper insights into the dataset's attributes. 'Extractive Summarisation' is explored involving three main techniques: TF-IDF, TextRank and Latent Semantic Analysis, each of them provide a unique approach to extracting summary information. Transitioning to 'Abstractive Summarisation', we explore contemporary techniques such as Seq2seq, T5, BART and the recently published LLAMA-2, detailing their architectures, training processes and the significance of fine-tuning. The 'Evaluation Methods' section explains quantitative metrics like ROUGE and METEOR and qualitative analysis procedures to gauge summarisation efficacy. Finally, the 'Dashboard Application' section illustrates the framework used to present results and the multitude of features it offers.

3.1 Data Collection

3.1.1 Source

The dataset utilised in this research is sourced from the CNN/DailyMail dataset, accessible through the Hugging Face dataset repository[1]. This vast repository of English-language content contains a collection exceeding 300,000 news articles generated by journalists associated with CNN and the Daily Mail between April 2007 and April 2015.

The structure of this dataset, as described in Table 3.1 is composed of discrete instances, wherein each instance contains an article, its corresponding highlights and a distinctive identifier. In the context of text summarisation, statistics regarding token usage are provided, articles have an average of 781 tokens, while highlights contain an average of 56 tokens. The dataset has been partitioned into three principal subsets: the training subset, the validation subset and the test subset, with quantities of 287,113, 13,368 and 11,490 instances respectively.

Table 3.1: Data Instance Example

Field	**Content**
id	0054d6d30dbcad772e20b22771153a2a9cbeaf62
article	(CNN) – An American woman died aboard a cruise ship that docked at Rio de Janeiro on Tuesday, the same ship on which 86 passengers previously fell ill, ... said. The Veendam left New York 36 days ago for a South America tour.
highlights	The elderly woman suffered from diabetes and hypertension, ship's doctors say. Previously, 86 passengers had fallen ill on the ship, Agencia Brasil says.

For the scope of this study, Version 3.0.0 is selected due to its adaptability to facilitate summarisation tasks, diverging from earlier versions, which initially centred around machine reading and comprehension. This version offers data without anonymisation, a change from preceding versions where named entities were substituted with identifier labels during the pre-processing phase.

3.1.2 Pre-processing

Text data in its raw form contains many irregularities such as inconsistent casing, punctuation, formatting and encoding. There can also be redundant or irrelevant content. Pre-processing steps like tokenization, cleaning and

[1] https://huggingface.co/datasets/cnn_dailymail

filtering are needed to convert the text into a standardised form. This involves breaking text into words, sentences and documents, converting to lowercase, removing extra whitespace, filtering noise and replacing abbreviations.

Below are the descriptions of the techniques used for pre-processing:

3.1.2.1 Lower-casing

In pre-processing, the initial step involves converting all characters to lowercase. Although seemingly insignificant, this operation is crucial in reducing complexity when analysing text and maintaining uniformity among characters. Applying lowercase transformation ensures uniform treatment of words with varying letter cases, preventing the creation of duplicate tokens.

3.1.2.2 Punctuation Removal

The next step is to remove punctuation marks. These characters have limited meaning in NLP tasks and can be safely removed without affecting the main message. Removing punctuation also helps simplify tokenization and reduce vocabulary size for further analysis.

3.1.2.3 Special Character Removal

This step involves removing special characters from the text using regular expressions. Special characters include symbols, emojis or non-alphanumeric characters that do not contribute meaning to the summarisation process and can introduce noise.

3.1.2.4 Alphanumeric and Whitespace Preservation

Following the removal of punctuation and special characters, this step ensures that alphabetic characters and spaces are retained in the text. Any non-alphabetic characters are removed and only words and spaces remain in the text.

3.1.2.5 Tokenization

Tokenization involves splitting the pre-processed text into individual words or tokens. This is an important step as it provides a structured representation of the text, allowing the summarisation model to understand the input as a sequence of discrete units.

3.1.2.6 Stop-word Removal

Stop-words are common words (e.g., "the," "and," "is") that often do not carry significant meaning in the context of text summarisation. Removing stop-words helps reduce the noise in the text and focuses the model's attention on the more content-rich words. This step is essential for improving the efficiency of summarisation and enhancing the quality of generated summaries.

3.1.2.7 Lemmatization

Lemmatization involves reducing words to their base or root forms. For example, "running" is lemmatised to "run," and "better" is lemmatized to "good". This step helps standardise the text by grouping different word forms. Lemmatization ensures that different forms of the same word are treated as one, which can improve the model's ability to recognise word relationships and meanings.

3.1.2.8 Contraction Expansion

Contractions like "won't," "isn't," and "can't" are expanded to their equivalent full forms like "will not", "is not", "cannot". This is important because contractions can lead to variations in text representation and expanding them ensures consistency. Consistent text representation aids the summarisation model in recognising words correctly and reduces ambiguity.

These pre-processed text inputs serve as the foundation for the modelling stage, where algorithms generate concise and coherent text summaries by focusing on the essential information contained in the input text.

3.2 Exploratory Data Analysis

Exploratory data analysis is the first step in gaining insights from the CNN/Daily Mail dataset. This section analyses article and summary lengths, word distributions, sentiment, topics and named entities.

3.2.1 Length Distribution Analysis

The analysis of article lengths provides insights into the distribution of news content. Trends in news coverage can be identified by analyzing the distribution of article lengths using a histogram. Lengthy articles may contain in-depth analyses while shorter ones focus on concise news updates. This understanding is invaluable for text summarisation as it helps determine how much information a summary should encapsulate. Additionally, variations in article lengths may influence summarisation algorithms to adjust their generation strategy, ensuring that the resulting summaries are accurate and informative.

Similarly, analysing summary lengths also offers insight into the summarisation process's efficacy. The histogram of summary lengths showcases the distribution of condensed content, guiding whether the summarisation algorithm tends to generate concise summaries or retains more details. This analysis assists in optimising models, ensuring that summaries align with user's expectations for brevity or comprehensiveness.

3.2.2 Vocabulary Analysis

This analysis aims to get insights into the frequency distribution and characteristics of words within the dataset.

The initial analysis involves splitting the articles into words, leading to the creation of a word dataset. This dataset is then utilised to compute the frequency of each unique word present in the corpus. This frequency information provides insights into the significance and popularity of words across the dataset after the stop words have been removed at the pre-processing stage. We identify the words that dominate the content by sorting and selecting the most common words, potentially highlighting key themes or topics. Similarly, the analysis extends to the summaries associated with the articles.

To provide deeper insights into the data, the project also investigates the distribution of word lengths in frequency. This distribution can reveal whether the language in the corpus tends to use shorter or longer words more frequently. Analysing and visualising these patterns not only helps in understanding linguistic tendencies but also assists in making informed decisions regarding text summarisation techniques and aids in choosing vocabulary libraries.

3.2.3 Sentiment Analysis

Sentiment analysis can identify whether a text carries a positive or negative inclination through its polarity assessment. This process facilitates the examination of sentiment fluctuations across numerous articles, offering insights into the prevailing bias. The polarity score spans from -1 to 1 and acts as a measure wherein -1 indicates highly negative sentiment and 1 signifies extremely positive sentiment.

This knowledge becomes valuable when orchestrating the prompting of large language models and fine-tuning their responses. After considering the sentiment nuances in the source articles, the entropy of prompts can be optimally set guiding the generation of summaries that capture factual information and the underlying emotional tone.

3.2.4 Topic Modeling

This exploratory data analysis leverages Latent Dirichlet Allocation (LDA) to extract underlying topics and themes from a text corpus. LDA (Blei et al., 2003) provides an unsupervised strategy to identify semantic themes and clusters directly from the text data. Through inferring distributions of words that commonly appear together in documents, LDA can extract insightful topics without manual supervision or rules encoding. This allows more objective discovery of themes that are latent in the corpus.

Mathematically, the LDA generative process is formalised as:

$$p(\theta, z, w|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta) p(w_n|z_n, \beta), \tag{3.1}$$

Where:

- $p(\theta|\alpha)$ is the prior probability of the topic distribution θ given parameter α

- $p(z_n|\theta)$ is the probability of topic z_n given the topic distribution θ
- $p(w_n|z_n, \beta)$ is the probability of word w_n given the topic z_n per topic word distribution β
- θ is the topic distribution for a document (random variable)
- z_n is the topic for the n^{th} word in a document (random variable)
- w_n is the n^{th} word in a document
- α is the Dirichlet prior parameter
- β is the word distribution per topic parameter

LDA models each document as a mixture of topics, where each word is assumed to be generated from one of the document's topics. The probabilities are defined hierarchically in equation 3.1, with θ and z_n as the latent random variables. After multiple iterations, the proportions of topic assignments to words and documents converge to reflect latent semantic themes.

3.2.5 Named Entity Recognition

Named Entity Recognition (NER) is a natural language processing technique that identifies and categorises important information entities such as names, organisations and locations from unstructured text data. NER tokenizes the text into sentences, tags parts of speech and chunks based on grammar relations, then extracts named entities and their categories from the parsed structures. Key entities like organisation names, person names and locations as illustrated in figure.3.1 are valuable for downstream text analysis.

Figure 3.1: Example Sentence with Named Entities. *Credit:* Inés Roldós

Implementing NER allows for structured extraction of entities from unstructured text, enabling scalable search functionality on the dashboard application.

3.3 Extractive Summarisation

3.3.1 Term Frequency-Inverse Document Frequency

Term Frequency - Inverse Document Frequency is a well-established statistical technique extensively used in natural language processing research for text modelling and information retrieval applications. The core premise behind TF-IDF is that it can identify keywords and phrases in a document that are highly specific to the document relative to the entire document corpus. Thereby, TF-IDF scores words and phrases in a document based on two metrics - how frequently they appear in the document and how rarely they appear across documents in the corpus.

Words that appear frequently in a document but rarely across the corpus receive high TF-IDF scores and can be indicative of the document's subject matter. Thus, TF-IDF provides a simple yet effective statistical signal to assess the importance of words and phrases within a document in the context of the entire document collection. This technique can, therefore be leveraged for extractive single-document summarisation by scoring sentences based on the TF-IDF values of the words contained within them. Sentences with cumulatively higher TF-IDF scores can be extracted as summaries.

The Term Frequency (TF) component measures how frequently a term occurs within a document. It is computed as the number of times the term appears in the document divided by the total number of terms in the document. TF quantifies the relevance of the term within the particular document.

$$TF = \left(\frac{\text{Number of occurrences of the term in the document}}{\text{Total number of terms in the document}} \right) \tag{3.2}$$

The Inverse Document Frequency (IDF) component measures how important a term is across the entire document corpus. It is computed as the logarithm of the ratio of the total number of documents to the number of documents containing the term. IDF quantifies the specificity of the term relative to the corpus.

$$IDF = \log \left(\frac{\text{Total number of documents}}{\text{Number of documents containing the term}} \right) \tag{3.3}$$

Words that appear frequently in a few documents but rarely across other documents will have high TF and high IDF scores, resulting in very high TF-IDF. Such words are highly relevant to those specific documents. On the other hand, commonly occurring words across the corpus like "the", "is" and "of" will have low IDF and hence low TF-IDF scores despite having high TF values.

$$TF - IDF = (TF * IDF) \tag{3.4}$$

This statistical signal is leveraged to score and rank sentences for extractive text summarisation. TF-IDF provides an efficient unsupervised framework as a baseline extractive summarisation technique for text documents. More advanced techniques can build on this statistical foundation with deeper natural language understanding capabilities.

3.3.2 TextRank

TextRank is an unsupervised graph-based algorithm for extractive text summarisation inspired by applications of network theory in the context of text data. The core intuition behind TextRank is that important sentences in a document are analogous to important web pages on the internet. Just as the PageRank algorithm ranks web pages based on external hyperlink connections, the TextRank algorithm ranks sentences in a document based on their inter-connections. TextRank adapts the PageRank framework to model text summarisation as a graph-based ranking problem.

The algorithm represents the text as a graph with sentences as vertices and edges denoting similarity relationships. The connectivity and position of a sentence in this graph determine its relative importance. Iteratively applying the PageRank formula, a stationary probability distribution over sentences is achieved which defines their salience scores. The top-ranked sentences are subsequently extracted into the summary.

3.3.2.1 Graph Construction

Given a text document, the first step is to tokenize it into sentences. Let these sentences be represented as $S = s_1, s_2, ..., s_n$ where n is the total number of sentences in the document.

A graph G(V, E) is constructed where each sentence s_i becomes a vertex v_i in the vertex set V. Two vertices v_i and v_j are connected by an undirected edge with weight w_{ij} that represents the similarity between sentences s_i and s_j. Thereby, the edges capture the pairwise relationships between sentences.

The similarity w_{ij} is computed using cosine similarity between vector representations of the sentences. The sentence vectors can be formed in various ways, a popular method is using TF-IDF weighted average of the word vectors.

Therefore, the graph (figure. 3.2) represents the sentences as vertices, connected by edges denoting similarity. The resulting adjacency matrix forms the basis for TextRank scoring.

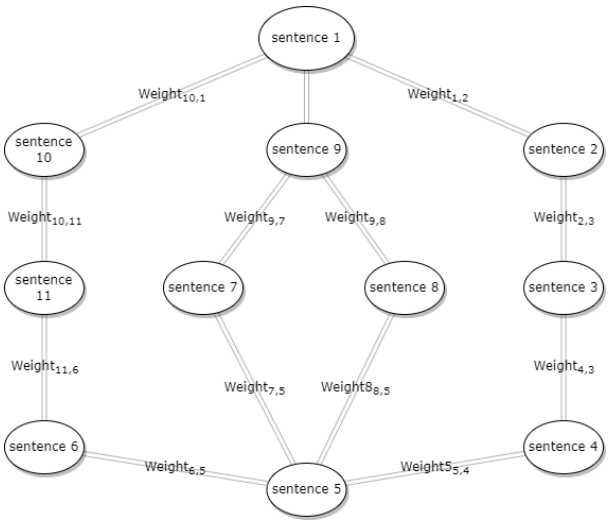


Figure 3.2: Sentence Graph.

3.3.2.2 TextRank Formulation

The TextRank rating of a sentence is defined recursively based on the ratings of its neighbouring sentences, similar to PageRank for webpages.

Mathematically, the sentence ranking generative process is formalised as:

$$Score(S_i) = (1 - d) + d * \sum_{j \in In(i)} \frac{w_{ji}}{Out(j)} * Score(S_j), \tag{3.5}$$

Where:
- $Score(S_i)$ = TextRank score of sentence S_i
- In(i) = Set of vertices pointing to S_i
- w_{ji} = Edge weight between S_j and S_i
- Out(j) = Outdegree of vertex S_j
- d = Damping factor

Initially, each sentence is assigned an equal TextRank score. The scores are recursively updated based on Formula 3.5 until convergence. The stationary score distribution achieved reflects the TextRank rating. TextRank then identifies important sentences based on their position in the similarity graph, determined by interconnectedness and relationships with other significant sentences.

Once TextRank scores for all sentences are obtained through the iterative algorithm, they are sorted in decreasing order of their importance. The top N sentences are then extracted from the original document according to the desired summary length. The extracted sentences form the final summary, retaining the most salient information from the document as quantified by the TextRank metric.

3.3.3 Latent Semantic Analysis

Latent Semantic Analysis is an unsupervised machine learning technique that utilises statistical computations to extract latent semantic information from a corpus of texts. The core premise is that words with similar meanings will occur in similar contexts or documents. Analysing patterns of word co-occurrence across documents, LSA can determine semantic similarity and extract the key themes and concepts.

The LSA methodology involves constructing a term-document matrix of words and their frequencies within each document. Dimensionality reduction via Singular Value Decomposition is then applied to this matrix to uncover the latent semantic structure and relationships between terms and documents. The sentences are then scored and ranked based on their representation of the dominant topics. The top-ranking sentences are extracted to form the summary.

3.3.3.1 Singular Value Decomposition

SVD transforms the high-dimensional vector space into orthogonal dimensions that expose the underlying latent semantics. Singular Value Decomposition is applied to the vectorised representation of sentences in the form of a weighted term-document matrix X to derive the latent semantic structure.

SVD factorises X into three matrices:

$$X = U_{m \times r} \Sigma_{r \times r} V^T_{r \times n} \tag{3.6}$$

Where:
- U contains the left singular vectors
- Σ is a diagonal matrix with singular values
- V^T contains the right singular vectors
- r is the rank of X

The columns of U represent relationships between terms, while the columns of V represent relationships between documents. Σ contains scaling values in descending order along the diagonal.

A low-rank approximation $\hat{X}$ can be obtained by zeroing out smaller singular values in Σ and reconstructing using only the top k singular vectors:

$$\hat{X} = U_{m \times k} \Sigma_{k \times k} V^T_{k \times n} \tag{3.7}$$

Setting $k << r$ filters noise and reconstructs semantic associations between terms and documents in a lower dimensional subspace. This surfaces the latent semantic variables that characterise the corpus.

3.3.3.2 Topic Modelling

The dimensions of the reduced SVD space represent the discovered latent topics. Terms with high magnitude coefficients in the same singular vectors are closely semantically related. Documents with similar vectors contain similar conceptual content.

The rows of $V^T_{k\times n}$ from equation 3.7 provide a new compact document representation in this semantic topic space. Dimensionality reduction via SVD extracts the key semantic information and filters out the noise that causes term usage variability.

3.3.3.3 Sentence Scoring

To determine sentence relevance for summarisation, the document sentences are projected into the semantic topic space.

Let s_i be the TF-IDF weighted term frequency vector for sentence i. The sentence vector ψ_i is computed as:

$$\psi_i = U_k^T s_i \tag{3.8}$$

ψ_i is the representation of sentence i in the latent semantic topic space. Sentences conveying key semantic concepts will have higher vector magnitudes and relevance scores.

3.3.3.4 Summary Extraction

The sentences are ranked by their relevance scores ψ_i (equation. 3.8) projected in topic space. The top n ranked sentences that fit within the desired summary length are extracted sequentially to form the final summary.

LSA selects content covering the main semantic themes without needing any topic keywords. The latent semantic variables capture conceptual similarity beyond surface word occurrences. This allows for generating informative summaries without supervised training.

3.4 Abstractive Summarisation

3.4.1 Sequence-to-sequence + Pointer Generator network

The initial approach in abstractive text summarisation involves utilising a neural sequence-to-sequence model with a pointer generator network for abstractive text summarisation. The model builds upon the encoder-decoder architecture commonly used in neural machine translation by augmenting it with a pointing mechanism to aid the accurate reproduction of information from the source text.

The overall pipeline consists of an encoder network that reads the input text and a decoder network that generates the output summary. Additionally, the pointer generator allows dynamic copying of words from the source text to handle out-of-vocabulary issues.

3.4.1.1 Encoder

The encoder maps the input text to a sequence of continuous representations called encodings. It captures the semantic meaning of the text that will be useful for the decoder to generate appropriate summaries.

In this project, LSTM cells are utilised as the encoder. LSTM networks, illustrated in Figure 3.3 are adept at learning long-range dependencies in sequential data like text.

The core components of LSTM are the memory cells, which can maintain information across many timesteps. Access to the cell state is controlled by three gates, namely the input gate, the forget gate and the output gate.

The first gate of the LSTM is the input gate i_t (equation 3.9) which decides what information is disregarded from the previous cell state. The sigmoid layer inspects h_{t-1} and x_t then outputs zero or one for each number c_{t-1}

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3.9}$$

In the forget gate of the LSTM, the now outdated c_{t-1} is updated to c_t (equation 3.10) as the new cell state. Consequently, f_t (equation 3.11) is multiplied by the previous c_{t-1} and then i_t. C_t is added, this is essentially the new values of state scaled by the update parameter.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3.10}$$

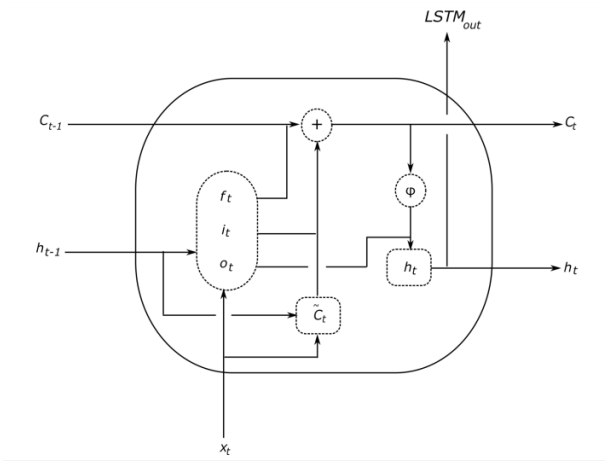


Figure 3.3: LSTM Architecture. *Credit:* Bandara et al. (2020)

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{3.11}$$

The last stage of the LSTM is the output, which will be based on a filter version of the cell state. It includes two sub-steps; running a sigmoid layer to decide output (equation 3.12) and putting the cell state through tanh (equation 3.13) to push the values between –1 and 1 and multiplying by the output of the sigmoid gate.

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{3.12}$$

$$h_t = o_t \odot \tanh(c_t) \tag{3.13}$$

Where,

- W_x is the weight matrix that projects the new input x_t to the hidden size h_{t-1}.
- W_h is the weight matrix applied on the previous hidden state h_{t-1}.
- W_c is the weight matrix applied on the previous cell state c_{t-1}.
- b_i is the bias term that controls the overall offset or activation of the input gate.
- σ is the sigmoid activation function defined in equation 3.14 .

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.14}$$

In simple terms, the input gate controls what to add, the forget gate controls what to remove and the output gate controls what to output from the memory cell.

The key aspect is the cell state c_t that can maintain information over long sequences, controlled by the input and forget gates. This helps LSTMs learn long-term dependencies critical in sequence tasks.

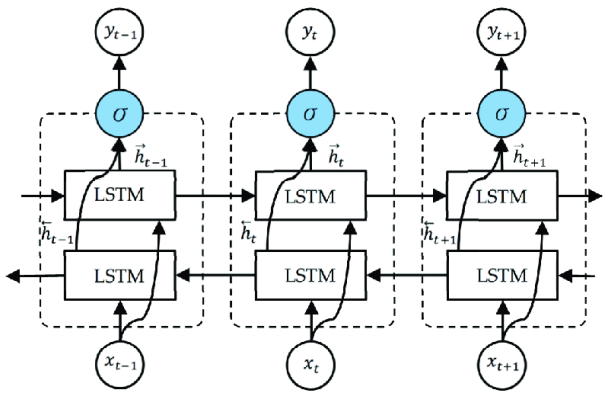


Figure 3.4: Bidirectional LSTM. *Credit:* Li, Harfiya, et al. (2020)

Although LSTM captures long-term dependencies, it can only capture uni-directional processing leveraging past context at each timestep. Future context is not available. A bidirectional LSTM encoder, as depicted in Figure 3.4 is used to get access to both past and future context at each timestep. The forward LSTM encodes the sequence from left to right, while the backward LSTM processes the sequence in reverse.

Given an input text sequence $X = (x_1, x_2, ..., x_n)$ of length n, the encoder LSTM processes it one token at a time and outputs a sequence of encoder hidden states $H = (h_1, h_2, ..., h_n)$.

At each timestep t, the hidden state h_t encodes information about the tokens x_1 to x_t seen up till that point. The final hidden state h_n thus captures a semantic summary of the entire input sequence.

$$h_t^{fwd} = LSTM_{fwd}(x_t, h_{t-1}^{fwd}) \tag{3.15}$$

$$h_t^{bwd} = LSTM_{bwd}(x_t, h_{t+1}^{bwd}) \tag{3.16}$$

The final encoder hidden state h_t is obtained by vector concatenation of the forward and backward states:

$$h_t = [h_t^{fwd}; h_t^{bwd}] \tag{3.17}$$

This provides each time-step access to contextual information from the entire input text. The encoder outputs H from the memory that the decoder will attend over to generate the summary.

3.4.1.2 Decoder

The decoder is a conditional language model that estimates the probability of a target summary sequence $Y = (y_1, y_2, ..., y_m)$ given the encoder outputs H. Using the chain rule of probability, the likelihood is decomposed as:

$$p(Y|H) = \prod_{t=1}^{m} p(y_t|y_{1,2,...,(t-1)}, H) \tag{3.18}$$

An LSTM network is used to model the conditional probability $p(y_t|y_{1,2,...,(t-1)}, H)$ at each timestep. It takes as input the previously generated tokens $y_{1,2,...,(t-1)}$ and the encoder outputs H. The initial hidden state of the decoder LSTM is initialised using the final state of the encoder. This promotes alignment between the encoder and decoder dynamics.

At each timestep t, the decoder hidden state s_t is updated as:

$$s_t = LSTM(y_{t-1}, s_{t-1}, c_t) \tag{3.19}$$

Where c_t is the context vector computed using attention over the encoder outputs H.

3.4.1.3 Attention Mechanism

The context vector c_t provides relevant parts of the encoder memory H that can aid the generation of the next token y_t. This is computed using an attention mechanism which acts as a soft search over H. First, alignment scores $e_{t,i}$ are calculated between the current decoder state s_t and each encoder hidden state h_i:

$$e_{t,i} = score(s_t, h_i) \tag{3.20}$$

Where $score$ is a feed-forward network scoring how well s_t and h_i match. Next, a softmax gives the normalised attention distribution α_t:

$$\alpha_{t,i} = \frac{exp(e_{t,i})}{\sum_{k=1}^{n} exp(e_{t,k})} \tag{3.21}$$

This decides which encoder words are most relevant for generating the next summary token.

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \tag{3.22}$$

The context vector is then computed as the weighted sum of encoder hidden states providing a dynamic representation of the most relevant source content.

3.4.1.4 Pointer Generator Network

A limitation of seq2seq models is out-of-vocabulary (OOV) words in the target sequence. To overcome this, a pointer generator network illustrated in Figure 3.5 is added, which allows dynamically copying of words from the source text.

The final probability distribution over the decoder vocabulary V and the input tokens X is computed as:

$$p(y_t|y_{<t}, H) = p_{vocab}(y_t|y_{<t}, H) + \sum_{i:x_i=y_t} p_{copy}(i|y_{<t}, H) \tag{3.23}$$

Where p_{vocab} is the standard softmax over V and p_{copy} is the attention distribution α_t over the inputs. The generation probability $p_{gen} \in [0, 1]$ controls the blend between generating from the vocabulary vs. copying from the input:

$$p_{gen} = \sigma(w_h^T h_t + w_s^T s_t + w_x^T x_t + b_{ptr}) \tag{3.24}$$

Where σ is the sigmoid function and w, b are learnable parameters.

If $p_{gen} = 1$, the model generates the next word from the vocabulary distribution $P_{vocab}(w)$

$$P(w) = p_{gen}.P_{vocab}(w) \tag{3.25}$$

If $p_{gen} = 0$, the model copies a word from the source by sampling the attention distribution α_t

$$P(w) = (1 - p_{gen}). \sum_{i:w_i=w} \alpha_t(i) \tag{3.26}$$

This allows dynamic switching between generating novel words and copying source text based on relevance.

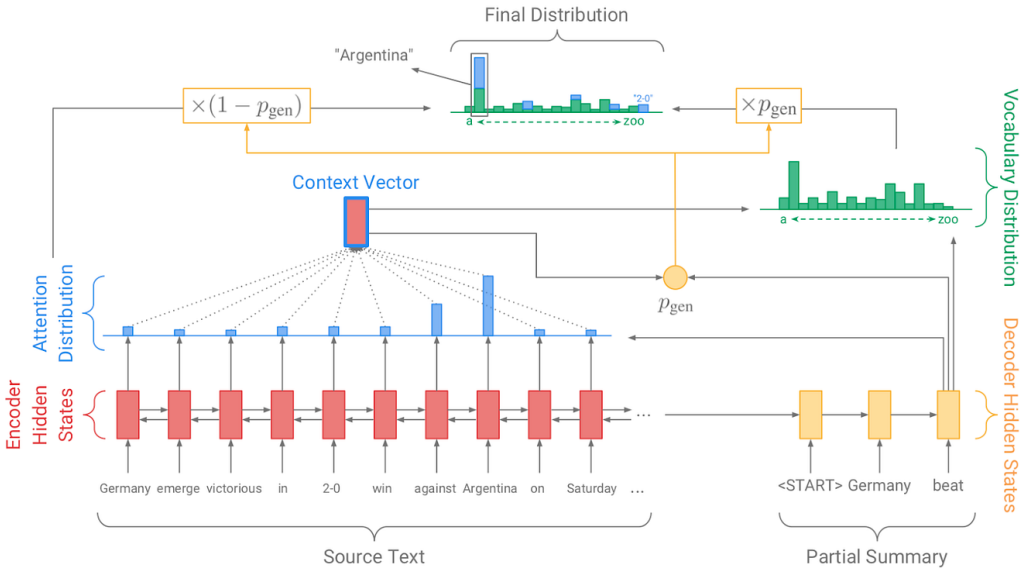


Figure 3.5: Pointer-Generator Seq2seq Architecture. *Credit:* See et al. (2017)

3.4.1.5 Training

The encoder and decoder LSTM weights, along with the attention and pointer generator parameters are trained jointly to maximise the summary log-likelihood:

$$\mathcal{L} = \sum_{i=1}^{N} \log p(Y^{(i)}|X^{(i)}, \theta) \tag{3.27}$$

Where $(X^{(i)}, Y^{(i)})$ are the i^{th} training sequence pairs and θ denotes all model parameters.

The objective is optimised over the training set using back-propagation and gradient-based optimisation like Adam. Teacher forcing is employed by using the ground-truth previous tokens y_{t-1} as input during training instead of the model's own predictions.

3.4.1.6 Inference

During inference, the encoder processes the input text to produce encodings H. The decoder initialised with encoder final states, attends over H and generates the summary token-by-token either from the vocabulary or by copying words from the source.

The model outputs probabilities over the decoder vocabulary and source tokens at each position. The token with the highest probability is appended to the current stream. This continues until the <END> token is reached, indicating summary completion.

The pointer-generator network combines the strengths of extractive and abstractive summarisation. The copying mechanism allows directly reproducing important keywords and details from the source article, while the vocabulary generation facilitates paraphrasing and abstraction.

3.4.2 T5

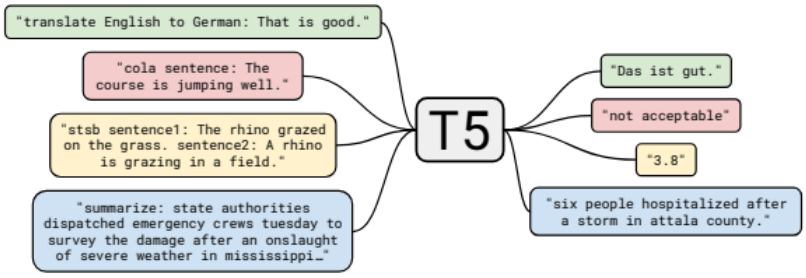


Figure 3.6: Text-to-Text Framework. *Credit:* Raffel et al. (2020)

Text-to-Text Transfer Transformer adopts a "text-to-text" framework illustrated in Figure 3.6, meaning that both input and output are treated as text sequences. This allows the same model architecture, objective function and inference procedure to be used across various natural language processing tasks including translation, summarisation and classification.

The T5 model applies a transformer-based encoder-decoder architecture illustrated in Figure 3.7 for abstractive text summarisation. The implementation of transformer architecture closely follows its originally proposed form described by Vaswani et al. (2017).

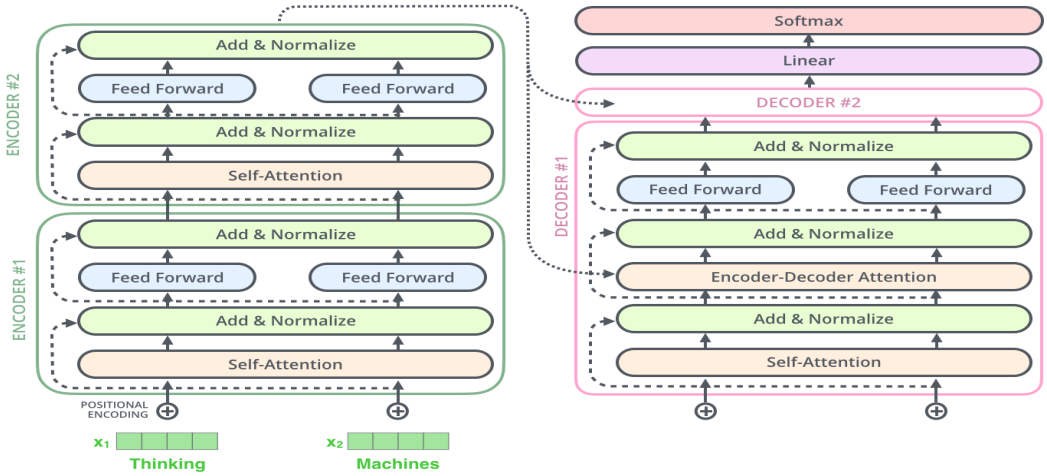


Figure 3.7: T5 Architecture. *Credit:* Jay Alammar.

3.4.2.1 Encoder

The encoder, illustrated in Figure 3.8 comprises stacked identical transformer blocks that use multi-headed self-attention and feed-forward neural network to model relationships between all input tokens.

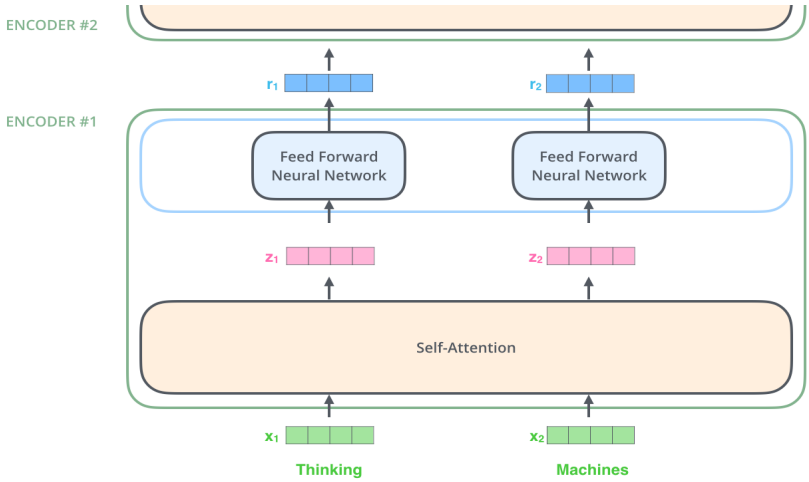


Figure 3.8: T5 Encoder Unit. *Credit:* Jay Alammar.

Multi-head Self-Attention: Self-attention, illustrated in Figure 3.9 allows relating different positions of the input text with each other. It computes attention scores representing dependencies between the different input tokens. The T5 encoder uses multi-head self-attention to perform this computation.

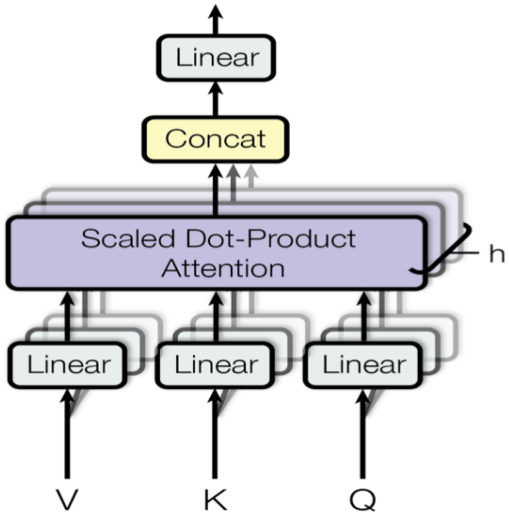


Figure 3.9: Multi-head Attention Module. *Credit:* Vaswani et al. (2017)

For a sequence of input embeddings $X = (x_1, ..., x_n)$ where $x_i \in \mathbb{R}^d$, multi-head self-attention first linearly projects them h times to get query, key and value vectors:

$$Q_i^j = XW_Q^j, K_i^j = XW_K^j, V_i^j = XW_V^j \tag{3.28}$$

where $Q_i^j, K_i^j, V_i^j \in \mathbb{R}^{d/h}$ and $W_Q^j, W_K^j, W_V^j \in \mathbb{R}^{d \times d/h}$ are parameter matrices for the j^{th} attention head.
It then computes scaled dot-product attention for each head:

$$\text{Attention}(Q^j, K^j, V^j) = \text{softmax}\left(\frac{Q^j K^{jT}}{\sqrt{d/h}}\right) V^j \tag{3.29}$$

The attention outputs of all heads are concatenated and linearly transformed to get the final multi-head self-attention output:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{3.30}$$

where $W^O \in \mathbb{R}^{d \times d}$.

This allows the model to jointly attend to information from different representation subspaces at different positions.

Feed-forward Neural Network: This sub-layer applies a position-wise fully connected feed-forward network to each position separately and identically. It consists of two linear transformations with a Rectified Linear Unit (ReLU) activation:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3.31}$$

where $W_1 \in \mathbb{R}^{d \times d_{ff}}, b_1 \in \mathbb{R}^{d_{ff}}, W_2 \in \mathbb{R}^{d_{ff} \times d}, b_2 \in \mathbb{R}^d$ are learnable parameters.

An encoder takes as input a sequence of vector representations. It processes this input sequence by first applying a self-attention mechanism, which relates different positions of the input to each other. The output of the self-attention layer is then passed to a feedforward neural network to perform non-linear transformation. The encoder output is propagated upwards to be used as input to the next encoder layer in the stack. Each encoder layer applies self-attention and feedforward processing to transform the representations, the final output is an encoded representation of the full input sequence.

3.4.2.2 Decoder

The decoder generates the summary autoregressively using its own transformer stack. Unlike traditional transformers, where separate models are used for encoding and decoding, T5 employs a single model for both tasks. It attends over the encoder outputs through cross-attention to focus on relevant parts of the source text. The attention distributions derive a context vector summarising key document content needed for the next summary token. Teacher forcing trains the model using ground truth tokens as inputs instead of its own predictions.

3.4.2.3 Softmax Layer

The decoder generates a vector of continuous representations as output. This vector is then passed through a linear transformation layer to project it into a much larger vector called logits, representing the score for each word. The softmax turns these into probabilities from which the model picks the most likely word.

3.4.2.4 Pre-training

Pre-training of T5 works by feeding the model massive amounts of text data and learning to predict masked-out words and sentences based on the surrounding context. Specifically, the input to the model during pre-training is corrupted text with some words or spans of text randomly masked out. The model is trained to reconstruct the original text by predicting the masked-out parts.

This is done using a text-to-text format where the input is text with masked tokens and the target is the original text. The objective is to maximise the likelihood of generating the correct tokens. T5 models are pre-trained in an unsupervised manner on a diverse corpus of datasets like Common Crawl web scrape data, books, Wikipedia, news articles and more. Billions of text input-target pairs are used to train the parameters of the model.

The pre-training objectives teach the model relationships between words, sentence structure, meaning, grammar and other linguistic concepts. This enables the model to develop a deep understanding of natural language that can then be transferred and fine-tuned for specialised NLP tasks like summarisation.

3.4.2.5 Fine-tuning

After pre-training comes supervised fine-tuning on document-summary pairs to specialise for summarisation. This adapts the general capabilities learned during pre-training to improve performance on the target dataset and task. Fine-tuning maximises the likelihood of generating reference summaries given source text by updating model parameters.

3.4.3 BART

Bidirectional Auto-Regressive Transformers is a denoising autoencoder that is pre-trained using a sequence-to-sequence model. BART utilises a standard Transformer-based neural machine translation architecture (Figure 3.10), which allows it to generalise BERT, GPT and other pretraining schemes. BART modifies the activation functions and parameter initialisation based on GPT. Specifically, the ReLU activations are changed to Gaussian error Linear Units (GeLUs). BART incorporates advantages from BERT's masking approach by implementing bidirectional encoder, GPT's left-to-right autoregressive decoder and sequence-to-sequence training.

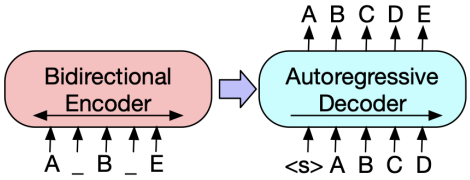


Figure 3.10: Schematic Representation of BART Architecture. *Credit:* Lewis et al. (2020)

The model is trained by corrupting text with a noising function and learning to reconstruct the original text similar to T5 model. The model employs various noising approaches, including randomly shuffling the order of sentences and using an in-filling scheme where spans of text are replaced with a single mask token.

The pre-trained model is then fine-tuned on document-summary pairs in a supervised manner for abstractive summarisation. The encoder encodes the document and the decoder generates the summary. The summaries also integrate information across the document and incorporate background knowledge. The model parameters are updated to maximise summary generation probability.

3.4.4 Llama-2

This research utilises the Large Language Model Meta AI-2 for abstractive text summarisation. LLAMA-2 (released July 18, 2023) was developed by Meta AI as a successor to LLAMA-1 released by Meta AI in February 2023 and incorporates knowledge graph embeddings to enhance the semantic representations learned during pre-training (Touvron, Martin, et al., 2023).

3.4.4.1 Architecture

LLAMA-2 is based on a transformer-based architecture commonly used for large auto-regressive language models. LLAMA-2 was pre-trained on a corpus of two trillion tokens with a context length of 4096 tokens. A key component is the tokenizer, which uses SentencePiece, a language-independent tokenizer and detokenizer designed by Google with 32,000-token vocabulary encodings to handle the input text. The released versions of LLAMA-2 range from 7 billion to 70 billion parameters, with depth varying from 32 to 53 layers depending on model size. The hidden dimension grows from 512 to 2048 for larger models. LLAMA-2's architectural choices provide strong language generation skills while efficiently training and adapting to varying levels of conciseness.

3.4.4.2 Pre-training

LLAMA-2 was pre-trained on a diverse mix of publicly available English text data totalling two trillion tokens. Factual data sources were upsampled in the mix to help reduce hallucinations in the model. The pretraining objectives were autoregressive language modelling to predict the next token and masked span prediction to reconstruct the partially hidden text. The training utilised an AdamW optimiser and a cosine decay schedule was used for the learning rate. Additional optimisations like weight decay, gradient clipping and a large batch size of four million tokens were employed to enable efficient scaling. The models were trained on Meta's internal clusters equipped with A100 GPUs.

3.4.4.3 Low-Rank Adaptation of Large Language Models

Large language models like LLAMA-2 contain billions of parameters, making them computationally expensive to fine-tune for new tasks. This challenge is particularly daunting for individuals with restricted computational capabilities. To address this hurdle, researchers have devised methods for efficient fine-tuning of parameters. These methods aim to attain optimal model performance while keeping computational demands minimal. Low-rank

adaptation is one such technique to efficiently adapt these huge models to new tasks with minimal computational overhead.

The fundamental concept behind Low-Rank Adaptation, illustrated in Figure 3.11 is that adapting the original model to a new task doesn't require updating all its parameters. Instead, only a minor portion of these parameters are selected. This is usually less than one percent of the parameters and the LoRA keeps updating only these parameter weights to improve downstream tasks. This approach is feasible due the fact that natural language models exhibit a low-rank structure. Modifying only a limited set of directions within the high-dimensional parameter space allows us to effectively capture most of the signal.

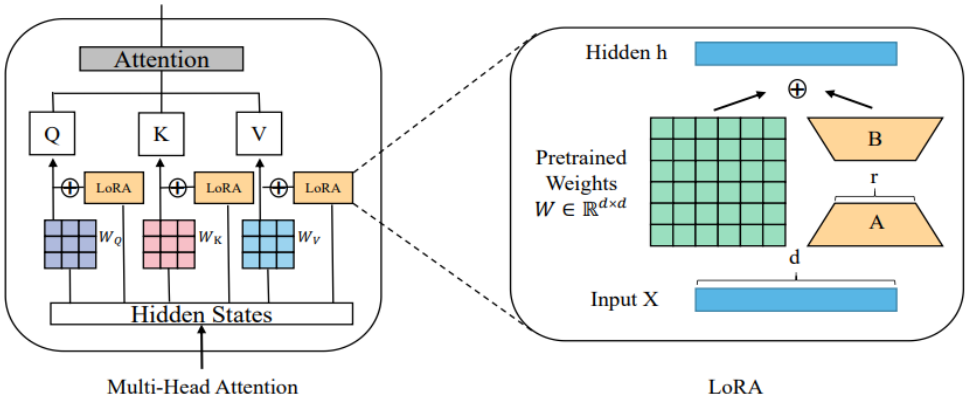


Figure 3.11: Transformer Architecture with LoRA. *Credit:* Wang et al. (2023)

The process of LoRA begins by selecting a pre-trained model like LLAMA-2 with parameters W. Then a small number of adaptation parameters A are selected, where A has fewer rows than W.
The adapted parameters are computed as:

$$W_{adapted} = W + A \tag{3.32}$$

Where A is a low-rank matrix that captures the perturbations to W.

To determine the adaptation parameters A, a small amount of task-specific training data is needed. This training data is used to learn the low-rank adaptation A that captures the essential changes needed to make the pre-trained W adapt to the new task.

Once the adaptation to a specific task is learned, the adapted model is constructed by adding low-rank adaptation matrix A to the original weights of the Large Language Model as described in equation 3.32. This adapted model retains most of the knowledge in the original W, but gains specialised capability on the new task.

This parameter-efficient adaptation with low-rank regularisation yields several advantages over full fine-tuning for abstractive summarization. The computational efficiency enables faster iteration and experimentation. The modular encapsulation of the summarisation adapters A allows flexible building on the ever-improving foundation of models like LLAMA-2 and practically injecting sophisticated summarisation capabilities into massive models within hours on a single GPU.

3.5 Evaluation Methods

3.5.1 Quantitative Evaluation

Quantitative evaluation of text summaries refers to numerically measuring the quality of summaries generated by automatic summarisation systems. This includes metrics like ROUGE and METEOR which compare the system summary to reference summaries and calculate similarity scores. The key advantage of quantitative evaluation is it provides concrete numerical scores that can be systematically compared across conditions and benchmarked over time.

3.5.1.1 ROUGE

The Recall-Oriented Understudy for Gisting Evaluation (ROUGE) scores represent the evaluation metrics for summarisation. ROUGE measures the similarity between the generated and reference summaries in terms of

overlapping n-grams.

ROUGE-1 measures the overlap of unigrams (single words) between the system and reference summaries and calculates recall, precision and F1. ROUGE-2 does the same for bigrams (two-word sequences). ROUGE-L identifies the longest common subsequence between the system and reference summaries and computes recall, precision and F1 based on this.

Precision: Precision measures the correctness of the generated summary. It is the ratio of the number of correctly identified relevant elements (such as words or n-grams) in the generated summary to the total number of identified elements. A higher precision indicates a lower rate of false positives.

Recall: Recall measures the comprehensiveness or completeness of the generated summary. It is the ratio of the number of correctly identified relevant elements in the generated summary to the total number of relevant elements present in the reference summary. A higher recall indicates a lower rate of false negatives.

F1 score: The F1 score is the harmonic mean of precision and recall. It provides a balanced measure that considers both precision and recall.

The F1 score is calculated using the formula:

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{3.33}$$

It ranges from 0 to 1, where a higher F1 score indicates a better balance between precision and recall.

3.5.1.2 METEOR

Metric for Evaluation of Translation with Explicit Ordering (METEOR) improves on metrics like BLEU and ROUGE by accounting for synonyms, stems and paraphrases when scoring a machine summary versus references.

The key innovation is using a harmonised alignment between the machine and reference summaries. This alignment uses modules like a synonym matcher and a paraphrase database to enable matches between semantically equivalent words and phrases. The harmonised alignment identifies semantic similarities that would be missed by simple string matching.

The aligned words are used to calculate precision, recall and fragmentation penalty to produce a composite METEOR score from 0 to 1. Higher scores indicate better semantic correspondence between the machine summary and references. METEOR improves on ROUGE by capturing semantic, not just lexical similarity.

3.5.2 Qualitative Analysis

Qualitative evaluation involves human judgement and manual inspection to assess summary quality holistically. This includes ratings of coherence, fluency, redundancy and coverage of salient information. Qualitative methods also include human checking for factual consistency and errors like hallucinations. The main benefit of qualitative techniques is capturing nuanced aspects of summary quality that automated metrics cannot fully measure.

Ideal evaluation combines both quantitative metrics and qualitative human checks to leverage the strengths of each other. Quantitative measures enable efficient comparison of different methods and conditions. Qualitative inspection provides an indispensable human perspective on the subtler aspects of summary quality that machines cannot yet fully assess. Using both types of evaluation together allows rigorous, comprehensive analysis to drive progress in summarisation research.

3.6 Dashboard Application

3.6.1 Framework

This project uses Plotly Dash libraries in Python to build a dashboard application. Plotly Dash is an open-source framework for building analytical web applications in Python built on top of Plotly.js, React and Flask, Dash abstracts away all of the technologies and protocols required to build an interactive web-based application and makes it simple to get a dashboard up and running with minimal code.

At the core of Dash is a set of reusable React components for creating analytical user interfaces like dropdowns, graphs, tables, buttons, sliders and more. Dash connects user interface elements such as graphs and dropdowns to

python analytical code by binding these components to Python functions and objects. This allows the user to build the logic in Python rather than JavaScript.

Dash apps run in the web browser. Data visualisation and processing happen in Python on the server and the results are sent to the browser where the React components update and render them. Callbacks define how components interact with the Python code. When input components like dropdowns are changed, the callback function is run, updating the output components like graphs. This creates dynamic and interactive applications.

3.6.2 System Architecture

The architecture, illustrated in Figure 3.12 consists of a Plotly Dash frontend communicating with a Flask backend running on Google Colab, with data stored on Google Drive.

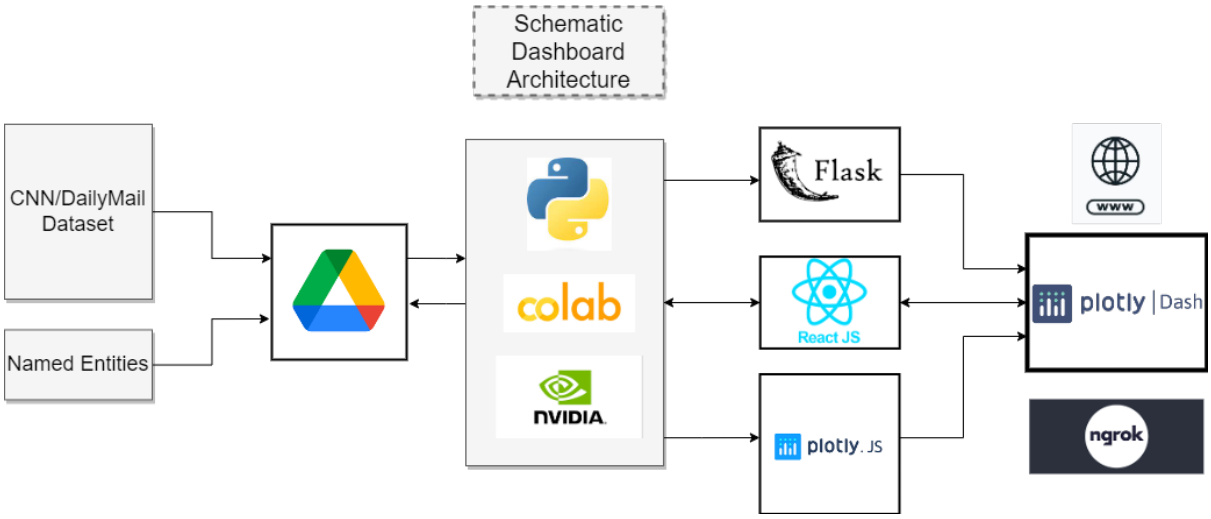


Figure 3.12: Dashboard Architecture.

On the frontend, Plotly Dash provides the core user interface components like graphs, dropdowns and tables using React.js under the hood. Callbacks wired to these components execute Python code on the backend to process data and update the User Interface. The backend runs on a Google Colab notebook instance, Nvidia A100 GPU is leveraged to speed up the usage of summarisation models. Flask runs on the Colab backend to serve the Plotly Dash app. It handles routing URLs to callback functions that execute the Python data processing and model inference logic. Flask also serves front-end assets like CSS and JavaScript to the browser.

The data sources consist of the CNN/DailyMail datasets stored on Google Drive from HuggingFace data repository. This includes the original text corpus as well as derived named entity and annotation files. The files are loaded and parsed in the Colab notebook kernels to extract the data needed for analysis. To deploy the app, the ngrok library opens a public URL to the Flask web server running on Colab. This tunnels traffic over HTTPS to allow external access to the Dash frontend.

3.6.3 Features and Functionalities

3.6.3.1 Entity Search

The entity search feature allows users to search for named entities like people, organisations and locations to find related news articles. This is powered by Named Entity Recognition technique in the backend. The application first uses a NER model to detect named entities in the news articles dataset and extract them. The extracted named entities are stored in a DataFrame with the article ID and entity name. This DataFrame is loaded into the application and provides the backend data for the entity search. When the user types into the search box, it looks for entity names that start with the typed prefix and shows autocomplete suggestions.

Once the user selects an entity, the application uses the DataFrame to lookup article IDs for that entity name. It then fetches the full articles for those IDs and summarises them using the preferred model. NER provides the ability to automatically detect and extract named entities from unstructured text data. Storing these entities in a structured form powers the search capability, allowing users to easily find articles relevant to a particular entity without manual tagging or metadata.

3.6.3.2 Event Summary

The event summary feature, illustrated in Figure 3.13 in this application works by generating multiple article summaries related to a particular entity or event and combining them into a consolidated summary.

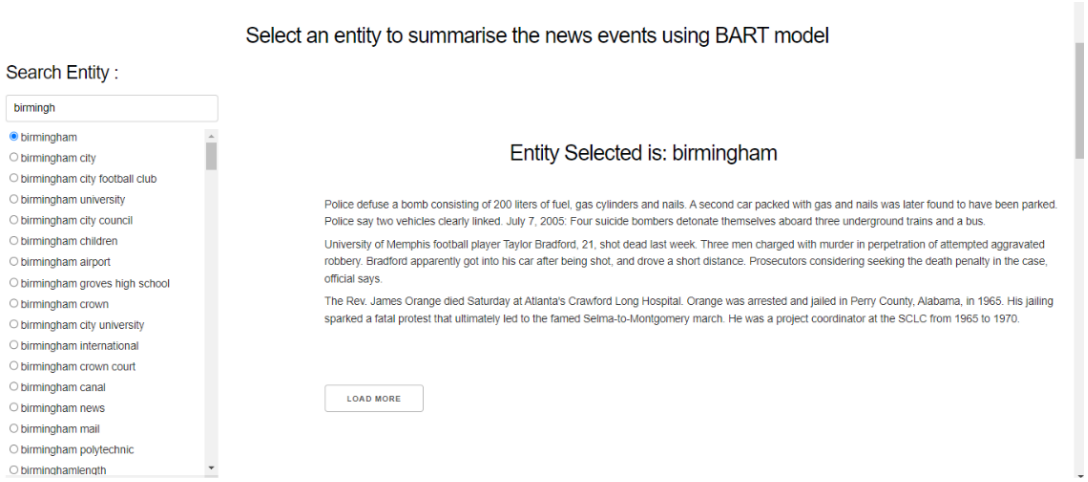


Figure 3.13: Event Summary.

When the user selects an entity, it retrieves a list of articles related to that entity using the extracted named entities as described earlier. It then generates a summary for each article using the BART summarisation model. BART is an abstractive model fine-tuned on news data to produce concise summaries. These per-article summaries are then concatenated together to provide a multi-document summary giving an overview of the key events related to the entity.

For efficient display of events related to an entity. The first batch shows three summaries and lets users click a "Load More" button to understand more about the specific event. When the "Load More" button is pressed, a callback function is executed and three more summaries are displayed, this process can be repeated till all the articles related to an event in a corpus are exhausted.

3.6.3.3 Topic Summary

The topic summary feature, illustrated in Figure 3.14 works similarly to the Event summary by generating summaries of multiple articles related to a chosen topic or entity using different underlying summarisation models. Additional features for topic summary include options to select specific summarisation models, control the length and conciseness of topic summary and ability to send email.

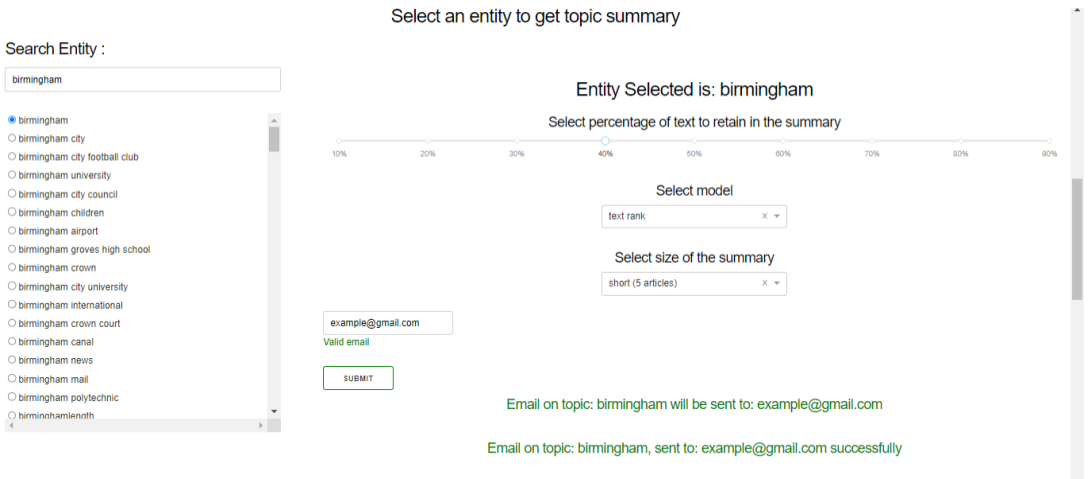


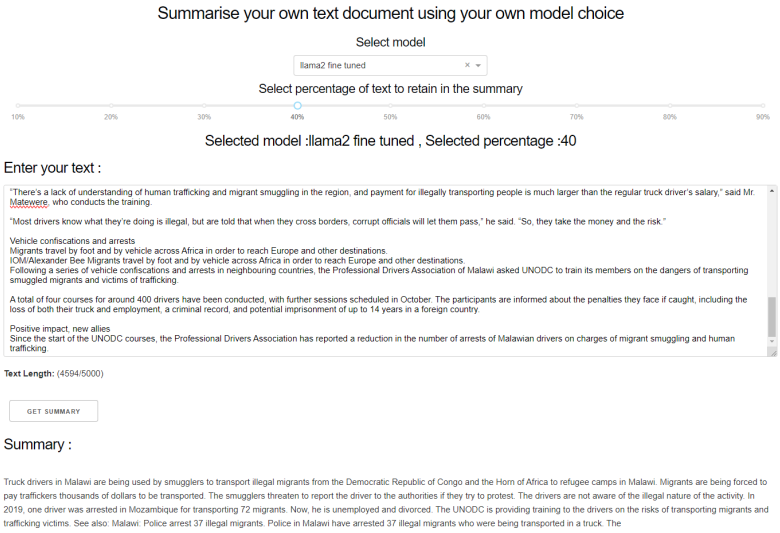
Figure 3.14: Topic Summary.

When the user selects an entity, it retrieves the most recent set of articles related to that entity. These articles are summarised independently using models specified by the user. The Single-document Summarisation outputs

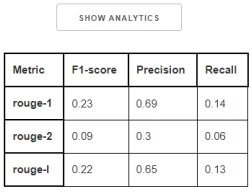
are then compared for relevance to the overall topic using cosine similarity on TF-IDF vectors. The most relevant summary per article is selected based on maximum cosine similarity to the topic. These selected summaries are concatenated based on the rank from the similarity score to create a Multi-document Summarisation outputs.

3.6.3.4 Article Summary

The custom article summarisation feature, illustrated in Figure 3.15a allows users to input their own text and generate a summary using different underlying models. The user enters or pastes their custom text into the text area provided in the application. They can choose a summarisation model from the dropdown. Using the slider, they select the summary length as a percentage of the original text. When the user clicks "Get Summary", the custom text is summarised using the selected model and length percentage.



(a) Summary Generation



Metric	F1-score	Precision	Recall
rouge-1	0.23	0.69	0.14
rouge-2	0.09	0.3	0.06
rouge-l	0.22	0.65	0.13

(b) ROUGE score

Figure 3.15: Custom Article Summary.

For extractive models like TF-IDF, LSA and TextRank, key sentences are extracted from the original text based on importance scores and combined to form the summary. For abstractive models like T5, BART and LLAMA-2, the full text is ingested and a completely new summary is generated based on learned patterns. The generated summary is displayed back to the user. Additionally, ROUGE evaluation metrics (Figure 3.15b) are calculated to quantify the quality of the summary against the original text.

This feature allows users to summarise any text of their choice using both extractive and abstractive models in a customisable way. The inclusion of ROUGE metrics provides analytical feedback on the quality of the generated summary.

The next chapter will present the system implementation details, including the data preprocessing pipelines, model architecture specifications, training procedures and inference workflows.

Chapter 4

Implementation

4.1 Environment Setup

This project utilised the python programming language for implementation, as it provides a robust ecosystem for data science and machine learning libraries. To balance computational demands, both local and cloud-based environments were leveraged. Initial statistical methods and prototyping were executed locally. Cloud execution on Google Colab was used for large-scale processing and training of deep neural models, providing them access to GPU acceleration. Colab was chosen because it's integration with Google Drive offered convenient high-capacity storage for caching large corpora and downloading sizable pre-trained models from HuggingFace.

4.1.1 Local Environment Setup

Python Environment: To leverage the benefits of dependency and version control, the python implementation was executed within an Anaconda virtual environment. A new Conda environment was created with Python 3.7 to provide a contained workspace. All required libraries like TensorFlow, PyTorch and scikit-learn were installed within this environment.

Spark Cluster Environment: In the Exploratory Data Analysis stage, Spark was used to compute on a large dataset and derive insights as it performs better than Pandas Dataframes at scale. Spark utilises a distributed architecture for large-scale data processing. It is an open-source framework built for handling large volumes of structured, semi-structured and unstructured data.

This clustered architecture provides a general engine for distributed data analytics applications, leveraging parallel execution. It uses data frames similar to pandas as the fundamental data storage mechanism to optimise the Spark process and big data computation. The use of this framework was necessitated due to the size of the corpus having more than 300K records.

4.1.2 Google Colab Setup

Cloud-based accelerated computing was used to enable large-batch training and inference of deep learning models for this project. Access to high-memory GPUs provided the necessary computational performance and capacity. A NVIDIA A100 GPU with 40GB of VRAM was utilised. Additionally, the cloud infrastructure provided an extended runtime of 24 hours for uninterrupted experimentation.

The blend of local and cloud environments provided the ideal balance of flexibility, scalability and integration with data storage needed at each phase of development and experimentation.

4.2 Data Ingestion

The dataset is accessed through the HuggingFace library using `load_dataset()` function from the datasets library. The dataset is fetched and stored locally through downloading and caching. Implementation of this method eliminates the need for manual downloading and unpacking of raw data files. This simplifies data ingestion and makes it easily reproducible.

The function call returns a dataset object that contains both the article texts and summary texts in separate columns. The metadata is described in Table 4.1 and has train, validation and test splits ready for use in modelling.

Table 4.1: Dataset Split and Metadata

Subset	Features	Number of Instances
Train	article, highlights, id	287,113
Validation	article, highlights, id	13,368
Test	article, highlights, id	11,490

4.3 Term Frequency-Inverse Document Frequency

The implementation pipeline described in Figure 4.1 begins by pre-processing the raw text data, including lowercasing, removing punctuation and stopwords, and lemmatizing. The CNN/DailyMail dataset is ingested and tokenized into words and sentences. The algorithm then loops through each sentence and calculates a TF-IDF score for every candidate word that is not a stopword. The Term Frequency is computed by counting the word's occurrences in the sentence divided by the total terms. The Inverse Document Frequency weighs down common words by taking the log of the sentence count divided by sentences containing that word.

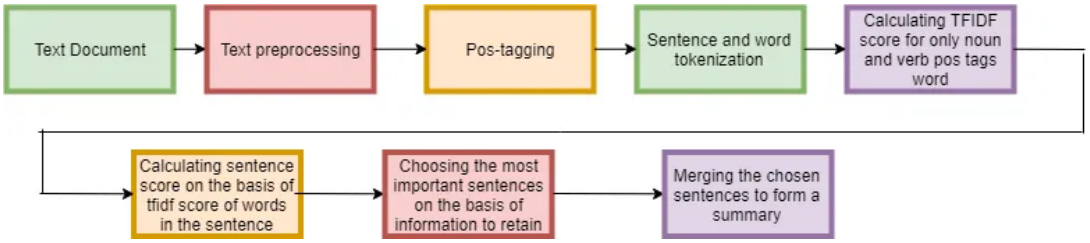


Figure 4.1: TF-IDF Model Pipeline.*Credit:* Ashna Jain

For a given sentence, the algorithm first initialises an accumulator variable to store its aggregated TF-IDF score. Next, NLTK's part-of-speech tagging is leveraged to identify informative nouns and verbs while filtering out stopwords and short words.

The pre-processed tokens are lemmatized to consolidate different forms of a word which helps better represent its semantic meaning. For each of these filtered, lemmatized words, their term frequency in the current sentence and inverse document frequency across all sentences are calculated. The TF and IDF statistics are multiplied to produce the TF-IDF value for that word, which represents its weighted relevance.

Looping through each word and accumulating its TF-IDF into a total sentence score, the system arrives at a representative metric indicating the significance of the overall sentence. The sentences are then ranked based on these scores and selected for inclusion in the summary. Sentences are ranked by their TF-IDF scores in descending order and the top portion are selected for the summary based on the number of top N sentences required to be selected as a summary sentence.

The number of sentences to retain in summary is set to 10 percent of the total number of sentences in an article, this value is selected based on the insights derived from Exploratory Data Analysis, which showed that the length of golden summaries in the whole corpus is 50 tokens, while the length of articles is 500 tokens on an average.

This process is repeated for each of the CNN/DailyMail test dataset containing 11,490 instances and quantitative scores are calculated including ROUGE precision, recall and F1 scores. To process each of the instances from a Pandas dataframe, a lambda function is used.

4.4 Text Rank

As outlined in the previous implementation, the first phase is pre-processing the input articles. The pre-processed tokens for each sentence are retained to construct the text graph.

With the sentences tokenized, the next phase constructs a vectorised version of sentences to build a similarity matrix. For the current algorithm, "TfidfVectorizer" from scikit-learn feature extraction library is used to vectorise sentences. To build a similarity matrix of the article, each sentence is treated as a vector using the TF-IDF weighted word representation, which assigns higher weights to rare and informative terms. Taking the cosine similarity of these TF-IDF vectors gives a semantic similarity score between 0 and 1 for any two sentences.

This results in a symmetric n x n similarity matrix described in Table 4.2b defining the relationships between all sentences in the document. A higher value indicates two sentences convey related information. This matrix transforms the unstructured text into a mathematical graph formulation.

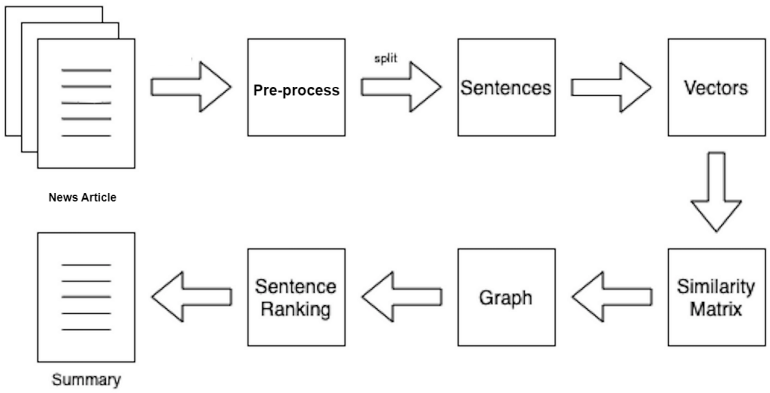


Figure 4.2: TextRank Model Pipeline. *Reproduced from:* Arda Cem Özmen

Table 4.2: Similarity Matrix Construction

(a) Example Sentences

#	Sentences
1	"I love to play soccer."
2	"Soccer is my favorite sport."
3	"I enjoy watching soccer matches."

(b) Sentence Similarity Matrix

	S1	**S2**	**S3**
S1	1.000	0.670	0.354
S2	0.670	1.000	0.537
S3	0.354	0.537	1.000

The numeric similarity matrix is converted into a networkx graph object, where each sentence is a node and weighted edges connect similar nodes (Figure 3.2). The edge weights are proportional to the cosine similarities. This network representation models the interconnectivity and relationships between the semantic content.

With the text graph formulated, TextRank assigns an importance score to each sentence based on its connections to other sentences. The score is calculated using the PageRank algorithm. This ranking scheme identifies globally important nodes in the graph based on recursive weighted aggregation from neighbouring nodes. The mechanism captures the notions of centrality in the text.

Once computed, the sentences are sorted by their TextRank score in descending order. The top N sentences composing the desired summary length of 10 percent of the original length are selected. Similarly, other article document summaries are generated from the test dataset and average ROUGE scores are calculated to quantify the quality of the generated summary.

4.5 Latent Semantic Analysis

The implemented Latent Semantic Analysis technique leverages linear algebra and dimensionality reduction to identify salient sentences.

As outlined in the methodology, the news articles are initially pre-processed using NLTK's tokenization, stopword removal and regularisation expressions. This cleans the text while retaining the original sentences. With the corpus tokenized, scikit-learn's "TfidfVectorizer" is applied to transform each sentence into a vector representation. This vectorisation described in Table 4.3 produces a document-term matrix encapsulating the semantics of the entire corpus in mathematical form. Each sentence vector represents the TF-IDF weighted word frequencies.

Table 4.3: Sentence Vectorization and TF-IDF Transformation

Sentence	TF-IDF Vector
"I love to play soccer."	[0.25, 0.25, 0, 0, 0.405, 0]
"Soccer is my favorite sport."	[0.25, 0.25, 0.405, 0.405, 0, 0.405]
"I enjoy watching soccer matches."	[0.25, 0, 0.405, 0, 0.405, 0.405]

Next, "TruncatedSVD" from scikit-learn performs dimensionality reduction on this matrix via Singular Value Decomposition. "TruncatedSVD" reconstructs a low-rank approximation of the original matrix by truncating to only the top low-dimensional subspace vectors (k). This retains only the most important latent semantic dimensions.

In practice, k is set to the desired number of sentences to extract, so the model focuses on the key semantic concepts. This compression removes noise and redundancy while preserving the semantic relationships between sentences.

The resulting reconstructed matrix after SVD truncation is the LSA representation with reduced dimensionality. To rank sentences, their LSA vector rows are summed to get scores (Table 4.4) indicating significance based on the latent semantic dimensions.

Table 4.4: Latent Semantic Analysis and Sentence Scores

Sentence	LSA Components	Sentence Score
"I love to play soccer."	[0.7, 0.2]	0.7 + 0.2 = 0.9
"Soccer is my favorite sport."	[0.5, 0.6]	0.5 + 0.6 = **1.1**
"I enjoy watching soccer matches."	[0.3, 0.7]	0.3 + 0.7 = **1.0**

The sentences are sorted by these scores and the top-ranked ones within the target summary length are selected. The implemented approach demonstrates how matrix factorisation can distil key semantic information without any labelled training data. The next step evaluates against reference summaries using ROUGE metrics on CNN/DailyMail dataset.

These unsupervised technique provides a strong baseline leveraging fundamental linear algebra and statistics.In the next sections, neural network methods like seq2seq models will be explored to produce abstractive summaries with paraphrasing and generalisation capabilities.

4.6 Sequence-to-sequence + Pointer Generator network

This section details the implementation of the sequence-to-sequence pointer generator network. The overall pipeline involves data pre-processing, defining the model architecture, training on the CNN Daily Mail dataset, generating and inferring summaries on a test set and evaluating with ROUGE metrics.

4.6.1 Data Pre-processing

Before the raw text data could be input into the neural network model, it required substantial pre-processing. Pre-processing steps included lowercasing all text, removing punctuation marks, expanding shortened words using the 'contractions' library, taking out non-alphabetic characters and tokenizing the text into individual words using NLTK's `word_tokenize()` function. Additional pre-processing involved removing common stopwords defined in NLTK's stopword list, lemmatizing words to their base form using "WordNetLemmatizer" and joining the processed word tokens back into full sentences.

The articles and summaries were truncated to maximum lengths of 500 and 50 tokens each in order to enable batch processing. Finally, the Keras Tokenizer class was used to map the pre-processed text to numeric word-integer sequences based on word frequency, building separate mappings for articles and summaries.

Start and end tokens were added to mark sequence boundaries. Padding was added on the right to maintain consistent length for batch processing. Vocabulary was limited to frequent words that appeared at least five times in the corpus to avoid rare words, this step was taken to prevent unnecessary increases in vocabulary size and minimise compute during model training.

4.6.2 Model Architecture

4.6.2.1 Encoder-Decoder Architecture

Encoder-decoder architecture, illustrated in Figure 4.3 is a neural network model consisting of two main components - an encoder network that reads and encodes the input sequence into a fixed-length vector representation and a decoder network that uses this vector to generate the output sequence.

The implementation described in Table 4.5 begins by embedding layer that maps the integer word indices to dense vector representations. Pre-trained GloVe embeddings were loaded to initialise the embedding weights. The embedding dimensionality was set to 300 based on the pre-trained vectors.

The embedded input is processed by encoder Bidirectional LSTM that outputs a sequence of encoder hidden state vectors. The final state of this LSTM is passed to the decoder as the context vector. This latent vector condenses the full input sequence down to the most relevant information.

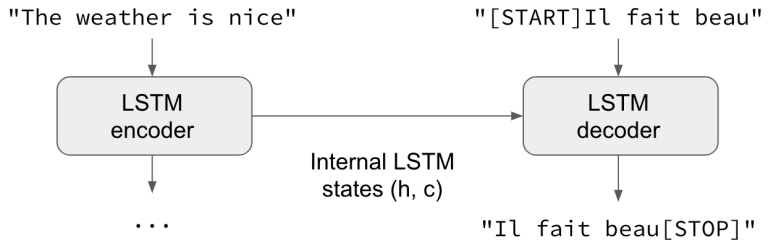


Figure 4.3: Seq2seq Encoder-Decoder Architecture. *Reproduced from:* Francois Chollet

Table 4.5: Encoder-Decoder Components

Layer	Description	Output Shape
encoder_inputs	Input layer for article text	(batch, 500)
encoder_embedding	Embedding layer for input text	(batch, 500, 300)
encoder_lstm1-3	3 stacked bi-LSTM layers	(batch, 500, 256)
encoder_states	Encoder state vectors	(256)
decoder_inputs	Input layer for target summary	(batch, None)
decoder_embedding	Embedding layer for target text	(batch, None, 300)
decoder_lstm	Unidirectional LSTM layer	(batch, None, 256)
decoder_outputs	Decoder output predictions	(batch, None, vocab)

At each timestep the decoder LSTM takes as input the embedding of the previous predicted word and the context vector from the encoder. It outputs predictions for the next word and updates its internal hidden state.

Teacher forcing was used during training, where the actual previous words were fed back into the decoder rather than its own predictions. This exposes the model to more accurate data during learning. The decoder outputs a probability distribution over all words at each timestep. The word with the highest predicted probability is chosen and added to the summary. This process repeats until an end token is predicted or the maximum summary length is reached.

To improve on the basic sequence-to-sequence approach, a pointer generator was added.

4.6.2.2 Pointer Generator Network

A limitation of the basic sequence-to-sequence approach is that it can only generate novel words when producing the output summary. It is often necessary to directly copy certain words from the input text, proper nouns like names and places generally should remain unchanged in the summary.

The pointer generator network, described in Table 4.6 augments the decoder by allowing it to directly copy words from the input in addition to predicting words from a fixed vocabulary. This improves accuracy by copying entities and details directly.

Table 4.6: Attention and Pointer Generator Components

Layer	Description	Output Shape
attention	Attention layer	(batch, None, 256)
context_vector	Attention context vector	(batch, None, 512)
pointer_logits	Pointer distribution logits	(batch, None, 1)
pointer_probs	Pointer distribution probabilities	(batch, None, 1)
output_gate	Generation probability gate	(batch, None, 1)
vocab_logits	Vocabulary distribution logits	(batch, None, vocab)
vocab_probs	Vocabulary distribution probabilities	(batch, None, vocab)
final_probs	Final word distribution	(batch, None, vocab)

4.6.3 Training

The model was trained using the parameters detailed in Table 4.7. It utilised the Adam optimiser and the sparse categorical cross-entropy loss function, which measures the difference between predicted and actual target word distributions. Early stopping was used to prevent overfitting by ending training when validation loss failed to decrease for several epochs.

Table 4.7: Training Parameters

Parameter	Description	Values
optimiser	Adam optimizer	
loss	Sparse categorical cross-entropy	
epochs	Number of training epochs	50 epochs
batch size	Batch size for training	128 sequences
early_stopping	Early stopping on validation loss	3 epochs patience
checkpoint period	Model checkpoint saving	5 epochs
teacher_forcing	Train decoder with ground truth	Enabled

The model was trained for 50 epochs with a batch size of 128 sequences. After each epoch, the training and validation losses (Figure 4.4) were saved to track convergence. The models were also periodically saved for every five epochs using Model checkpoints to restart training from the last saved checkpoint if needed. Teacher forcing was used during training, as described earlier.

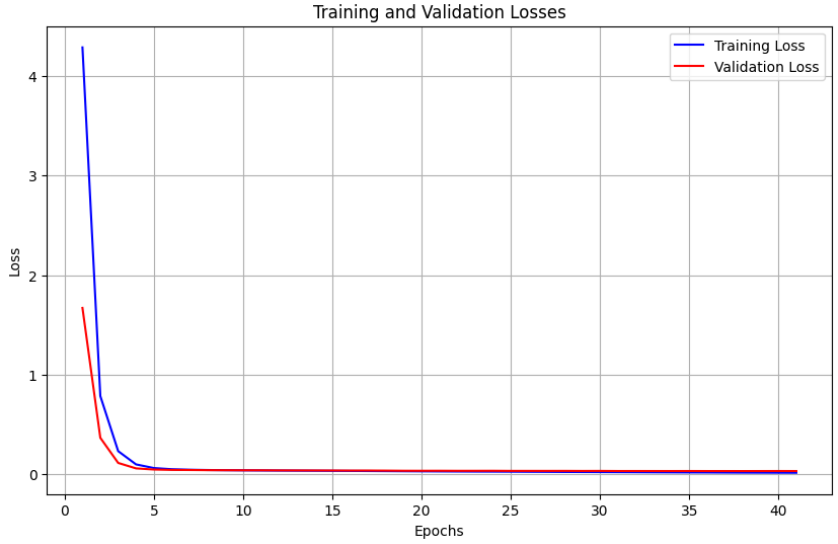


Figure 4.4: Seq2seq + Pointer Generator Loss Plot.

4.6.4 Inference

After training the encoder and decoder, the models could be applied to generate summaries for new articles. The trained encoder model encodes each input article into the fixed-length context vector. Then, the decoder conditioned on this context generates the output summary for each word until an end token is reached.

Teacher forcing is disabled at this stage and the decoder model recursively predicts the next word, given all previous predictions along with the encoder context. The trained decoder model outputs a probability distribution predicting the most likely next word. The word with the highest probability is selected as the next word in the summary.

This inference process is repeated for each article sequence to produce full summaries across the test dataset. To reduce memory requirements, the test set was split into batches and decoded iteratively. The summaries generated for each article were saved along with the original reference summaries. This complete set of predicted and actual summaries were then evaluated quantitatively on ROUGE metrics.

4.7 T5

Before the T5 model was fine-tuned on summary data, a pre-trained model was tested to serve as a benchmark to study the performance improvements. The TensorFlow Datasets API was used to load the CNN Daily Mail dataset containing news articles paired with multi-sentence summaries. The raw text was pre-processed by truncating summaries to roughly 10 percent of the article length. Articles were tokenized using the "T5Tokenizer" and fed as input IDs into the pre-trained "T5ForConditionalGeneration" model from Hugging Face Transformers. This model is based on a pre-trained Transformer encoder-decoder architecture.

The T5 model was instantiated with the "t5-base" checkpoint comprising 220 million parameters. At inference, each article was encoded into a high-dimensional representation using "T5Tokenizer". The decoder then autoregressively generated the output summary conditioned on this representation, one token at a time. Generated token IDs were decoded into text with the same "T5Tokenizer".

To enable efficient processing of the test set articles, batching was implemented when generating summaries. The complete test set was divided into batches of 100 examples each. This process of summarising articles in batches reduced memory overhead compared to processing all test set examples at once. After summarisation, the separate batch CSV files were concatenated to reconstruct the full test set with summaries.

4.8 Fine-tuned T5

The pre-trained T5 model from Hugging Face Transformers was instantiated as the starting point for transfer learning. Specifically, the 't5-base' model was loaded, which consists of a 12-layer Transformer encoder-decoder model. The raw text was tokenized and padded to maximum lengths of 500 and 50 tokens for articles and summaries respectively. This processed data was then converted into PyTorch tensors for model input.

TrainingArguments defined hyperparameters like batch size, number of epochs and logging frequency (Table 4.8) . The model and arguments were passed to a trainer instance that handled the fine-tuning loop. At each epoch, the loss was evaluated on the validation set to monitor convergence.

Table 4.8: T5 Fine-tuning Parameters

Parameter	**Value**
Model	*T5-base*
Article length	500 tokens
Summary length	50 tokens
Batch size	4 sequences
Epochs	3
Learning rate	Default (5e-5)
Optimisation	AdamW

After three epochs, the model checkpoint with the lowest validation loss was identified. Plots of the training losses showed a steady decrease (Figure 4.5), indicating the model was successfully adapting to the summarisation task through fine-tuning. After training, the final model was saved for inference.

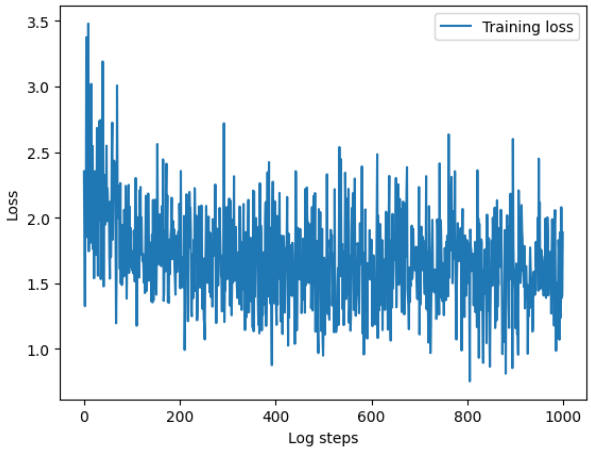


Figure 4.5: T5 Fine-tuning Loss Plot.

During testing, each article was truncated to its length and padded, the summary was limited to 10 percent article's length. The output summary is produced autoregressively, with the model generating the next token conditioned on all previous tokens. The resulting summary token IDs are converted back into text using the tokenizer. The ROUGE package was used to evaluate the model summaries against ground truth references and improvements over pre-trained model were studied.

4.9 BART

The "BartTokenizer" and "BartForConditionalGeneration" model from Hugging Face Transformers was instantiated using the pre-trained "facebook/bart-large-cnn" model with parameters described in Table 4.9.

Table 4.9: BART Summarisation Parameters

Parameter	Value
Model	*BART-large-cnn*
Input length	1024 tokens
Output length	10% of input
Beam size	4
Early stopping	True

The model loaded comprises 406M parameters and was pre-trained on CNN and Daily Mail articles. The articles were truncated to 1024 tokens before tokenization. The target summary lengths were set to 10 percent of the source length, capped between 56 and 1024 tokens.

During inference, the input article was tokenized and fed to the model to generate an output summary autoregressively using beam search. The generated token IDs were decoded back into text with the tokenizer. ROUGE evaluation metrics were calculated between the model summaries and reference summaries to quantify performance.

4.10 Llama-2

This section discusses the implementation of text summarisation using the LLAMA-2 large language model. LLAMA-2 is a 7 billion parameter autoregressive language model trained on large-scale real-world factual text data. While not fine-tuned on summarisation data, its broad language understanding capabilities allow for generating abstractive summaries when conditioned on a prompt.

A prompt-based approach described in Block 4.1 was used to leverage the pre-trained capabilities of LLAMA-2 for zero-shot summarisation [1]. A prompt template was defined containing instructions delimiting the input text and requesting a 10 percent length summary.

```
template = """ Write a concise summary of the following text delimited by triple backquotes.
              Return your response in paragraph form which covers the key points of the text.
              Add "<start>" at the beginning of the summary and "<end>" at the end of summary.
              Keep the length of summary 10 percent of the original article
              ```{text}```
           """
prompt = PromptTemplate(template=template, input_variables=["text"])

llm_chain = LLMChain(prompt=prompt, llm=llm)
```

Listing 4.1: Prompting LLM

To prompt the model, Langchain library was leveraged with parameters described in Table 4.10 to provide a clean interface for accessing the model via Hugging Face Pipelines and creating an LLMChain wrapping the prompt logic. At inference, the input article was substituted into the template and fed to the model to generate an abstractive summary.

To enable efficient summarisation of the full test set, the data was processed in batches of 100 articles. The LLMChain method allowed summarising the individual prompt by passing the article text and returning the generated summary. These were converted to a data frame and saved in a zipped CSV for each batch. The outputs were combined by concatenating the CSV files into a complete data frame containing articles, reference summaries and model-generated summaries.

[1] zero-shot summarisation is the technique of generating summaries from pre-trained model without explicitly training them on summarisation task, models are provided with only instructions, generally in the form of prompts.
```

Table 4.10: Llama-2 Summarisation Configuration

Parameter	Description	Value
Model	The pre-trained language model used.	*Llama-2-7b-chat-hf*
Input length	The maximum input token length allowed.	1024 tokens
Temperature	Controls the creativity of the text generated.	0
Top-k	The top-k value for token selection.	10
Num return sequence	The number of generated sequences.	1
Summary length	The length of summary relative to input.	10% of input

An important advantage of this approach is the ability to rapidly prototype summarisation without expensive training. However, performance can lag behind fine-tuned models. The prompt engineering provides some ability to condition the model's behaviour but lacks the precision of gradient-based optimisation.

4.11 Fine-tuned Llama-2

This section provides an overview of implementing text summarisation by fine-tuning the LLAMA-2 language model using Parameter Efficient Fine-Tuning (PEFT) based on Low-Rank Adaptation (LoRA) described in Figure 4.6. PEFT allows rapidly adapting models like LLAMA-2 to new datasets through low-rank updates to the weight matrices.

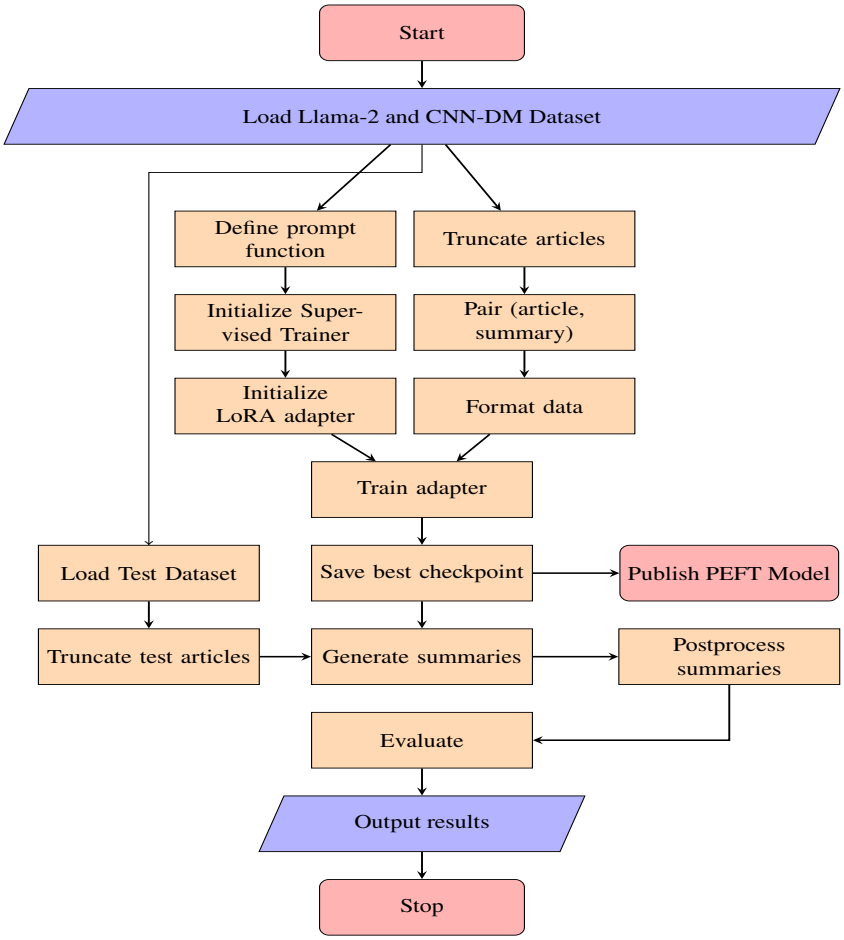


Figure 4.6: Llama-2 Supervised Fine-tuning Workflow.

The core technique in PEFT is LoRA, which appends low-rank adapter modules to the pre-trained model. These adapters contain parameter matrices initialised randomly. When multiplied with the original weights, they update the weights tuned to the downstream dataset. Only the low-rank matrices are trained on the new data, keeping the original model weights mostly constant.

To implement this approach, PEFT library provided model classes to integrate LoRA adapters into LLAMA-2. "PeftConfig" defines hyperparameters like adapter size, regularisation and placement described in 4.11. The "TrainingArguments" are defined by hyperparameters like batch size, learning rate and number of epochs described in Table 4.12.

Table 4.11: Low-Rank Adapter Arguments

Parameter	Description	Value
`lora_alpha`	Controls the regularisation strength on low-rank adapter	16
`lora_dropout`	LORA Dropout	0.1
`r`	Specifies the rank of adapter matrices.	64
`bias`	Disables bias	"none"
`task_type`	Specifies model to generate text autoregressively.	"CAUSAL_LM"

Table 4.12: Training Arguments

Parameter	Description	Value
`num_train_epochs`	Number of training epochs	1
`optim`	Optimiser	"paged_adamw_32bit"
`save_steps`	Save steps	10
`logging_steps`	Logging steps	10
`learning_rate`	Learning rate	0.001
`weight_decay`	Weight decay	0.001
`max_grad_norm`	Maximum gradient norm	0.3
`max_steps`	Maximum training steps	-1
`warmup_ratio`	Warmup ratio	0.03
`lr_scheduler_type`	LR scheduler type	"linear"

The CNN Daily Mail articles and summaries were formatted into text pairs representing the input and target output. The train and validation sets were kept at 1,000 and 100 examples, respectively for prototyping. The full test set was used for the final evaluation. The decision to use less training and validation data is based on the fact that parameter-efficient models can perform tasks with very limited data, referred to as few-shot learning [2]. This is possible because the model has learned useful abstractions during pre-training.

A prompt function formatted the input-target demonstrations for training the LoRA adapter parameters. The low-rank matrices were initialised to trainable values. The "SupervisedTrainer" handled optimisations from training arguments, the parameters are described in 4.13.

Table 4.13: Supervised Fine-tuning Parameters

Parameter	Description	Value
`model`	Model architecture	*Llama-2-7b-chat-hf*
`dataset_text_field`	Text field in the dataset	"article"
`max_seq_length`	Maximum sequence length	500
`args`	Training arguments	Table 4.12
`dataset_batch_size`	Dataset batch size	64

The best checkpoint with minimum validation loss was saved. This adapted model could then generate summaries by conditioning on an encoded passage and decoding autoregressively. After fine-tuning the model, the optimised adapter parameters were published to share the adapted model[3]. This allowed the dissemination of a specialised summarisation model for usage in the test phase when loaded along with the pre-trained model.

The test articles were truncated to the max length to elimiate trailing spaces for inference. Each article was fed to the adapted LLAMA-2 summarisation model to generate a summary. The generated summaries were post-processed to clean up output formatting. ROUGE metrics were calculated between the system and reference summaries to quantify summarisation accuracy.

[2]Few-Shot Learning is the process of adapting an LLM for new categories of data using only a small number of labelled data.
[3]https://huggingface.co/vijayjawali/llama2_query_tuned

4.12 Dashboard Application

This section provides a comprehensive overview of implementing an interactive dashboard for multi-document summarisation of news articles related to user-specified entities. The dashboard enables selecting entities to summarise related news, generate summaries on custom text and email topic summaries.

The implementation involved integrating multiple summarisation models, building interactive User Interface elements for user control, optimisation for performance and final deployment. The code structure followed a component-based architecture using the Dash framework for Python.

4.12.1 Features and Functionalities

4.12.1.1 Event Summarisation

The Event summary feature workflow is described in Figure 4.7. It begins with "Select an entity" section that allows searching for entities present in the news dataset. As the user types into the input, suggested entities are dynamically filtered from the unique entities list based on substring matching.

The entity chosen from the radio button suggestions summarises news articles related to that entity present in the corpus. When the user selects an entity, the dataset is filtered to retrieve the article IDs containing the selected entity. Using these article IDs, the full article texts are then obtained from the corpus.

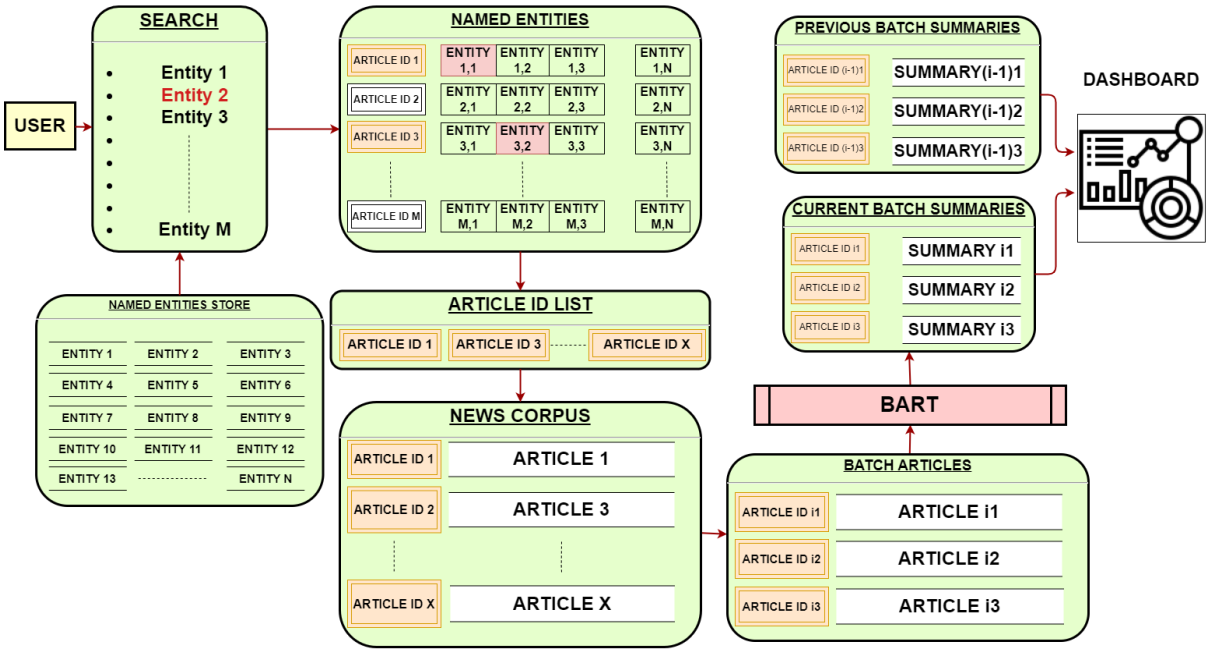


Figure 4.7: Event Summary Workflow.

The retrieved articles related to the selected entity are divided into batches containing three articles each. This creates multiple batches of articles for further processing. The first batch consisting of the first three articles is then summarised using the BART neural abstractive summarisation model. Each article is fed as input text into the "bart-large-cnn" model. BART encodes the article and then decodes it by each token to generate a summary. This process is applied to all three articles in the first batch. The generated summaries are then rendered on the dashboard to show the user.

After generating summaries for the initial batch of three articles using the BART model, a "Load More" button is displayed which allows incrementally loading additional content to the user. The user can click "Load More" to fetch another batch of articles once they have consumed the initial set.

After the "Load More" button is clicked, newly generated summaries are combined with the previously displayed content before rendering the updated page. This creates an aggregated event summary containing summaries from multiple batches.

As more batches are fetched via pagination, summaries from additional related articles are appended to the existing summary. So the event summary grows incrementally with each click rather than fully regenerating the

page. This pagination approach balances presenting a comprehensive multi-document summary while maintaining a digestible experience by expanding the content progressively based on user actions.

4.12.1.2 Topic Summarisation

The Topic summary feature workflow is described in Figure 4.7. This section provides customisable multi-document summarisation across top articles related to an entity. Suggestions are again filtered dynamically based on the entered topic. Once an entity is selected, the user can customise the summary percentage, model and length in a number of articles. The summary length ranges from short (5 articles) to long (15 articles).

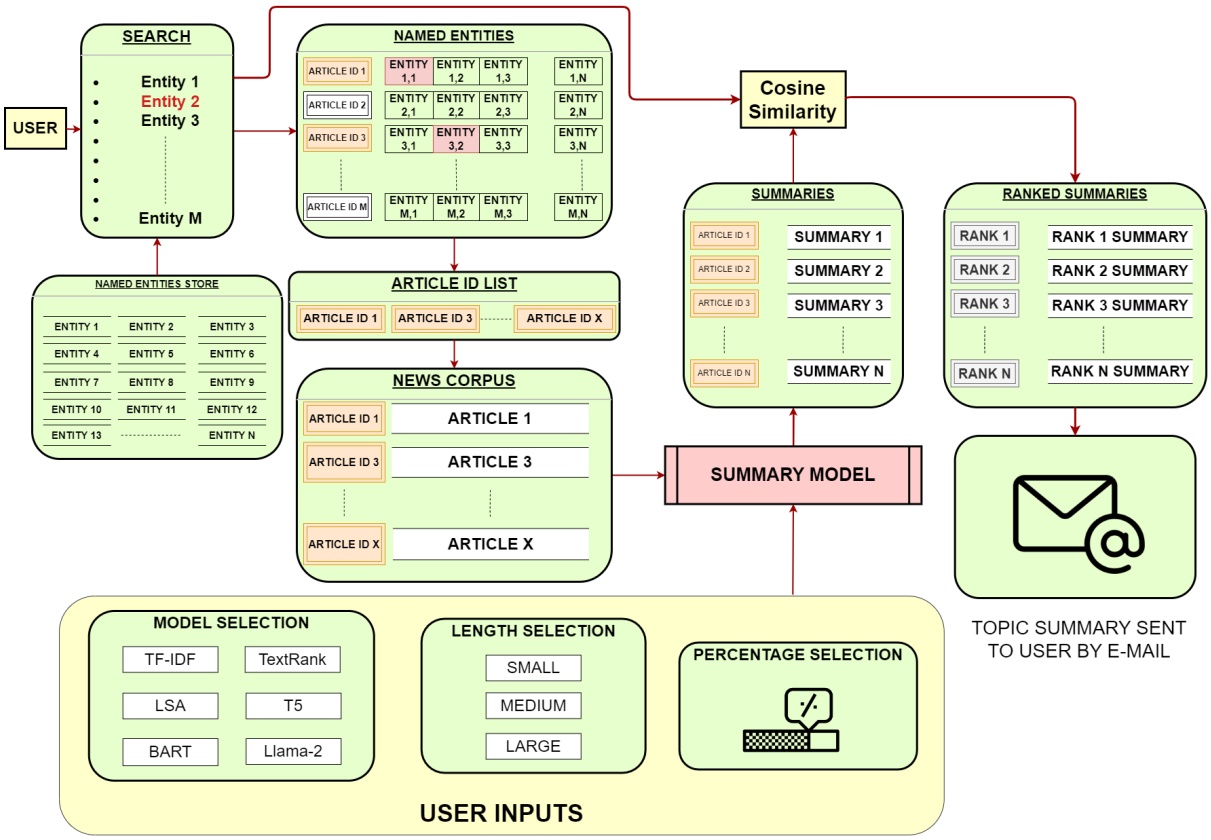


Figure 4.8: Topic Summary Workflow.

After entering a valid email, submitting will trigger an asynchronous call to generate the summaries. The selected model constructs a single-document summary by extracting sentences from the relevant articles.

Once summaries are generated for the selected articles related to the chosen entity, they are ranked based on relevance to the specified topic. The relevance ranking is performed by computing cosine similarity between the topic entity and each article summary. The summaries are then sorted in descending order of their similarity scores.

This places the most relevant summary with the highest match to the topic first, followed by other summaries in decreasing order of relevance. Ranking the generated multi-document summaries this way ensures the most relevant summary is shown at the top. It provides a logical structure to the summary email by prioritising content most strongly related to the user's specified topic of interest. The summaries are then emailed to the provided address after constructing the message with the SMTP protocol. The email content is rendered only after successful validation of custom selections, preventing unnecessary generation.

4.12.1.3 Custom Text Summarisation

In this section of the application, users can input arbitrary text content and generate a summary with their preferred percentage and model. A textarea input field accepts the content, which can be up to 5000 characters in length for performance reasons. The word count indicator warns if the limit is exceeded.

Dropdown menus allow picking summarisation models and the summary percentage from 10% to 90%. The selected parameters are displayed for clarity. When the user clicks the "Summarise" button, the application first

checks that all required inputs like text and model selection are provided. If any necessary inputs are missing, warnings are displayed to the user indicating what needs to be added. This client-side validation saves unnecessary calls to the server to generate summaries when prerequisites are not met. Only once all required information is present will the summarise request be sent to the backend.

The generated summary is rendered directly once available. A "Show Analytics" button displays ROUGE evaluation metrics between the summary and original text in a table for transparency.

4.12.2 Model Integration

The Python server backend integrated diverse text summarisation models based on both abstractive and extractive techniques.

For extractive methods like TF-IDF, TextRank and LSA, the text is preprocessed, scored and salient sentences extracted. Sentences with top relevance to the overall meaning were concatenated into extractive summaries. The diversity enables holistic evaluation of different techniques.

The summarisation dashboard also incorporates various neural network models for abstractive text summarisation, allowing users to compare their capabilities. The integrated models include pre-trained models like T5, BART and LLAMA-2 and models trained specifically for summarisation task like Seq2Seq Pointer Generator Network, Fine-tuned T5 and Fine-tuned LLAMA-2. These models generate summaries by encoding the input text and decoding an abstractive summary autoregressively.

4.12.3 Interactive Elements

Dash provides interactive widgets like dropdowns, text boxes and sliders to accept user inputs. These are rendered as HTML and CSS components in the front end. Callbacks in Python react to events like text entry or button clicks to update the application state.

For entity search, suggestions are filtered dynamically from the unique named entity list as the user types. This provides a responsive search experience. Under the hood, this is implemented using a callback that filters the unique entities list against the search text typed by the user. As the user types each character, the callback is executed to update the suggestion list. This provides a smooth and responsive search experience without page reloads. When the user selects an entity, the application generates summaries of related news articles containing that entity using the BART model.

Pagination allows incrementally loading more content. This feature allows users to reveal content progressively, avoiding large blocks of text and providing control over event understanding. The pagination is achieved by tracking paragraphs shown and hidden using component state variables that get updated when the button is clicked.

For topic summarisation, the component implemented is a similar search bar but focused on topic-based summarisation. It provides the same auto-complete suggestion experience, but when an entity is chosen, options appear for generating a customised summary emailed to the user. This includes selecting the model, length of summary and percentage of text to retain. The user can enter their email address which gets validated before submitting the request. Email validation happens through a callback that checks the text against a regex pattern. This improves robustness by only allowing valid emails.

The next dynamic component allows summarising custom text entered by the user. Options are provided to pick the model and percentage of text to retain. The choice of model and percentage are persisted in component state variables. When the user clicks the generate summary button, a callback uses these options to produce the appropriate summary of the custom text. A word count indicator shows the length of input text, warning if it exceeds the maximum characters allowed.

The analytics button component works in sync with the custom summarisation feature, when clicked shows ROUGE evaluation scores comparing the original text and summary. This helps users assess the quality and relevance of the generated summary quantitatively.

These interactive elements provide an intuitive flow for users to find summaries tailored to their interests. Client-side rendering along with Python callbacks enables building intricate features with reactive inputs.

4.12.4 Optimisation and Performance

For web application, optimising performance was crucial for responsive behaviour at scale. Various optimisations were implemented to ensure fast response times for users:

Asynchronous Design: The app uses asynchronous callback functions to avoid blocking the event loop. Summarisation occurs in background server threads while showing loading indicators.

Caching and Persistence: Model checkpoints and tokenizers are persisted locally to avoid re-downloading. Datasets are cached to prevent repeat pre-processing. Batch outputs in the event summary are saved to cache, rendering repeated runs efficiently.

Batch Processing: Input data is divided into batches to summarise chunks in parallel. This limits memory usage while maximising compute resource utilisation for event summaries.

Mixed Precision: Converting tensors to lower precisions like float16 from float32 reduces the memory footprint. This enables larger batch sizes for further acceleration.

Together, these allow the app to efficiently handle multiple users interacting smoothly. These optimisations also allow the dashboard to scale across the corpus of articles and entities while maintaining interactivity.

4.12.5 Deployment

The deployment of the interactive Plotly Dash application involves installing dependencies, exposing the application through a tunnelling service, mounting required files and executing the Dash application python code.

First, the Python dependencies are installed in deployment environment using the pip package manager. This includes core packages like Dash, Plotly and others required for data processing and machine learning models.

Next, a tunnelling service called ngrok is configured to expose the local Dash application to a public URL. The ngrok authentication token is set up and a tunnel is initiated to the port the Dash app will use. For deployment in a cloud environment from Google Colab, the required files and folders are mounted from Google Drive.

With the dependencies and environment set up, the Dash application code is executed. This launches the web app on localhost with a dynamically assigned port. The ngrok tunnel exposing this local port can then be used to access the Dash app publicly.

Chapter 5

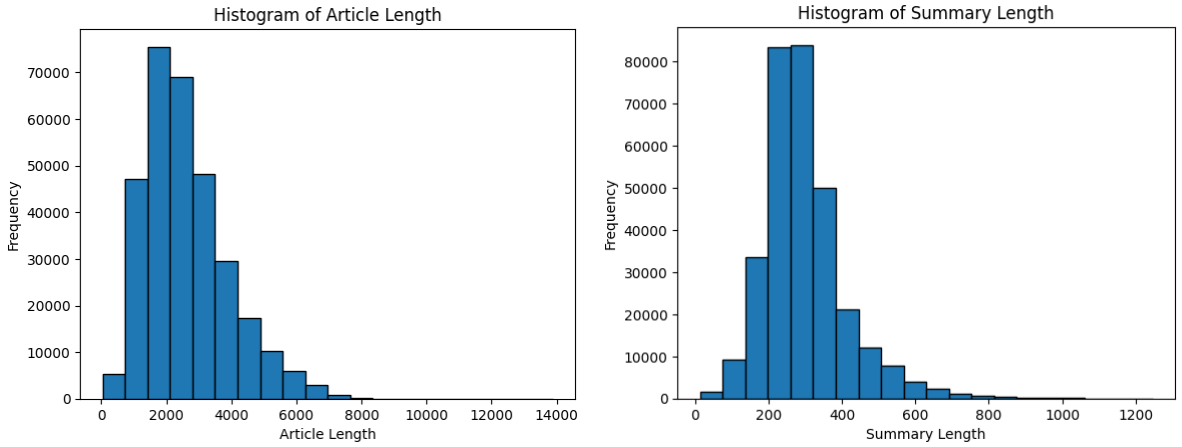
Results

This chapter evaluates various summarisation models on a news dataset using quantitative metrics and qualitative analysis. Both extractive methods and recent abstractive models are tested. Pre-trained versions are compared against fine-tuned variants to assess the impact of adaptation. The automatic ROUGE and METEOR metrics are examined to quantify lexical and semantic overlaps with reference summaries. Additionally, human evaluations provide perspective into coherence, fluency, redundancy and coverage of salient content. By holistically evaluating the models from both algorithmic and subjective viewpoints, the analysis aims to uncover relative strengths and weaknesses to guide appropriate model selection.

5.1 Exploratory Insights from News Corpus

5.1.1 Length Distribution Analysis

Performing statistical analysis on the dataset is a first step before developing a machine learning model for text summarisation. Examining basic length statistics provides insights into data distribution to set hyperparameters like target decoding lengths for model architecture and establishes compression ratio for the summarisation task. In this case, it was found that most of the articles in the dataset were less than 4500 characters long, while the provided summaries were under 400 characters (Figure 5.1). These values helped set the compression ratio to 10%.



(a) Article Length Distribution

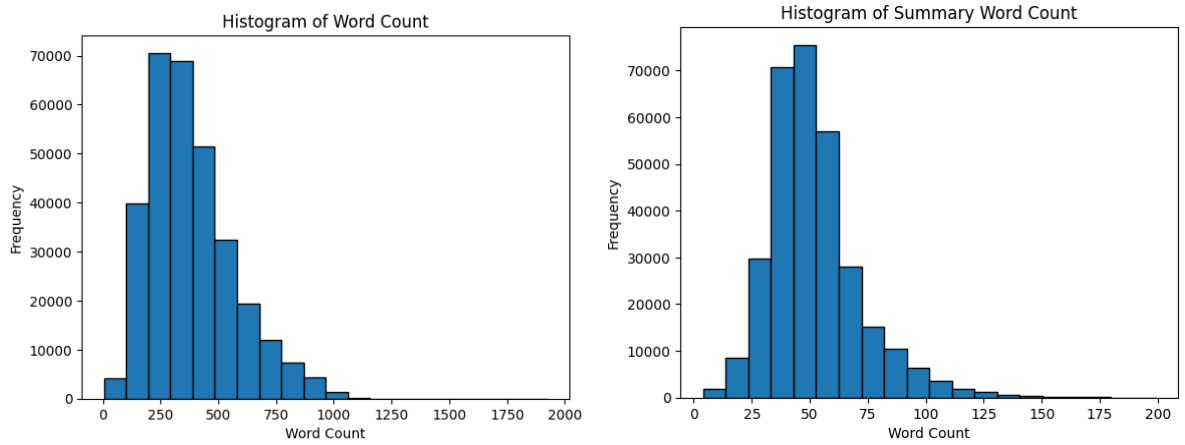
(b) Summary Length Distribution

Figure 5.1: Histogram of Text Length.

5.1.2 Vocabulary Analysis

In addition to studying the overall length distribution of articles and summaries, it was important to analyse the vocabulary and lexical characteristics before developing text summarisation models. As part of the preliminary statistical analysis, the distribution of word counts or tokens in the articles and summaries was examined. This involved plotting a histogram of the number of tokens to understand the variation.

The analysis found that most articles contained around 500 tokens, while the summaries were restricted to less than 50 tokens (Figure 5.2). This indicated a compression ratio of approximately 10% from the input text to the output summary in terms of vocabulary.

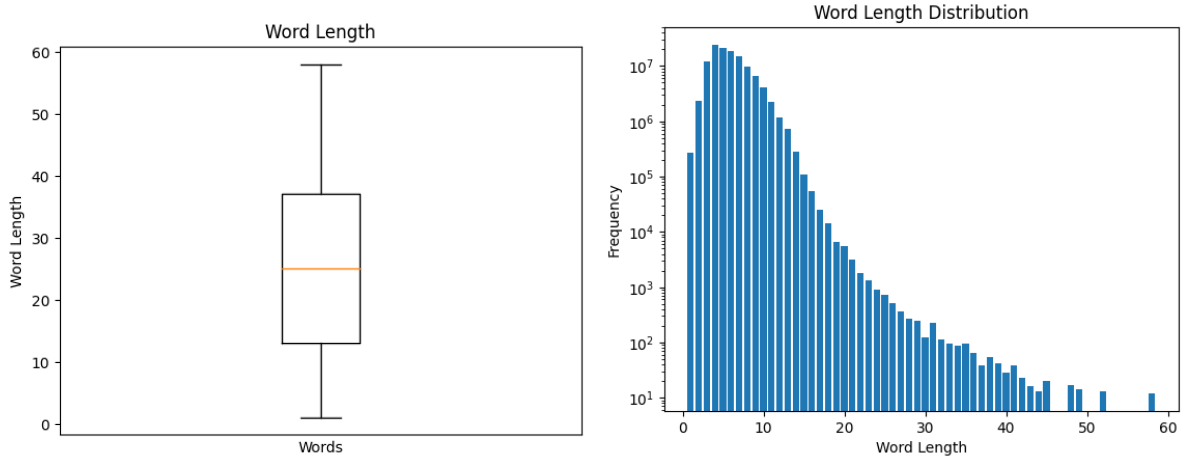


(a) Article Word Length Distribution

(b) Summary Word Length Distribution

Figure 5.2: Histogram of Text Word Length.

The box plot visualisation of the word length distribution provided additional insights into the lexical characteristics of the corpus (Figure 5.3a). The box plot along with the histogram (Figure 5.3b) highlights some key statistics. The minimum word length was one character, the maximum length observed was 58 characters and the median length was 25 characters. This indicates that most words were relatively short, with 50% of the vocabulary being 25 characters or less in length.



(a) Box plot of Word Length

(b) CNN/DailyMAil Word Length Distribution

Figure 5.3: News Corpus Word Length Distribution.

5.1.2.1 Topic Modelling

Performing topic modelling provides insights into the thematic structure of a text dataset before developing summarisation models. For this corpus, LDA was applied. It is an unsupervised machine learning technique that extracts abstract "topics" within a collection of documents.

The model identified major topics along with the most representative words for each one as shown in Table 5.1. The topics capture logical themes like politics, sports and entertainment. Understanding these topics helps ensure the dataset has sufficient diversity to train generalised models or guides us to select models pre-trained on specific topics.

Table 5.1: LDA Topics and Topic Names

Topic Name	**Top Words**
Topic 0: Daily	use,find,year,also,per,make, people,could,new, cent,one,food,study,dr
Topic 1: Transport	car,water,one,take,area,around,build,flight, see,two,park,year,passenger,crash
Topic 2: Legal	police,court,tell,officer,find,charge,death,two,home, yearold,man,arrest,report,family
Topic 3: Think	not,get,go,would,tell,like,think,one,i, child,take,time,is,know
Topic 4: Politics	mr,would,claim,labour,year,minister,make, last,people,party,pay,government,also,work
Topic 5: Government	state,new,per,president,cent,year,house,would, obama,million,not,people,clinton
Topic 6: Sports	game,league,player,play,season,club,team,win, goal,last,unite,england,score,first,leave
Topic 7: Entertainment	year,show,new,one,star,also, picture,film,make,include,world,look,first,take,london
Topic 8: Conflict	attack,group,kill,force,isi,fight,people, war,state,islamic,military,syria,one
Topic 9: International	country,china,world,president,russian,international,state, government,russia,official,south,africa,foreign

5.1.2.2 Sentiment Analysis

In addition to analysing article lengths and vocabulary, sentiment analysis was also performed on the dataset as part of our preliminary statistical review. Sentiment analysis involves computationally identifying and categorising the subjective tone of the text, it expresses positive, negative or neutral sentiment.

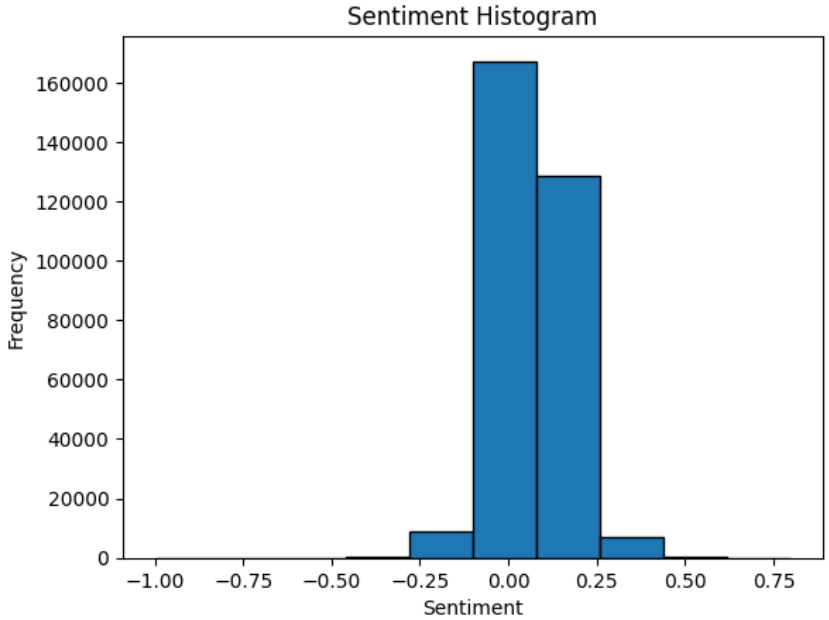
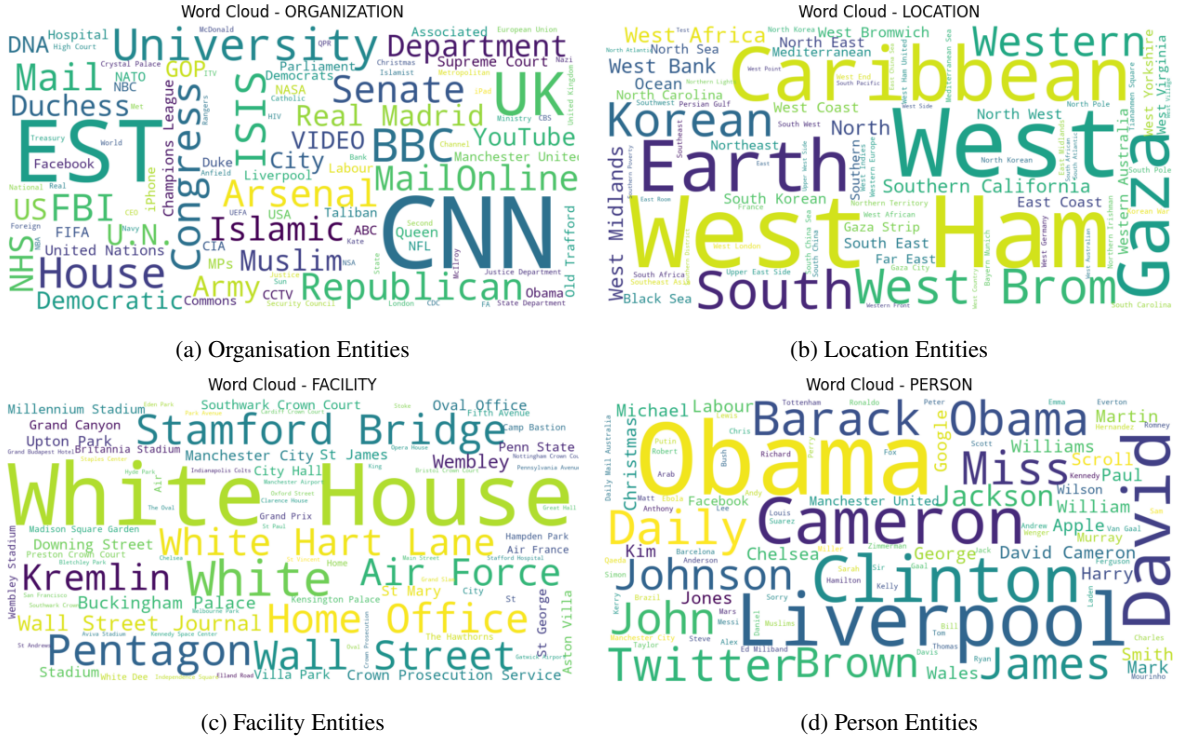


Figure 5.4: Articles Sentiment Polarity.

The analysis found a relatively balanced sentiment distribution, indicating no significant overall bias (Figure 5.4). Most articles expressed neutral sentiment, while positive and negative sentiment articles were present in equal measure.

5.1.2.3 Named Entities

Performing NER discovers the distribution of entities like organisations, persons, locations and facilities in a data-driven manner without manual annotations. The extracted entities illustrated in Figure 5.5 power the interactive search in the dashboard app, letting users filter summaries by entities of interest.



(a) Organisation Entities

(b) Location Entities

(c) Facility Entities

(d) Person Entities

Figure 5.5: Named Entities.

5.2 Qualitative Analysis

A qualitative study evaluates and compares several text summarisation models on their performance in summarising a news article. The original input article is long, detailing a narrative story. A short two-sentence golden summary was created to highlight the key details of the article. This section analyses the summarisation capabilities of each model on this specific news article described in 5.2, the detailed version is available in Appendix B.1. The strengths and weaknesses of each summary are discussed in relation to the golden summary and original text

Table 5.2: Original Article and Golden summary

Original Article
(CNN)He's a blue chip college basketball recruit. She's a high school freshman with Down syndrome. At first glance Trey Moses and Ellie Meredith couldn't be more different. But all that changed Thursday when Trey asked Ellie to be his prom date. Trey – a star on Eastern High School's basketball team in Louisville, Kentucky, who's headed to play college ball next year at Ball State – was originally going to take his girlfriend to Eastern's prom ... "It's exciting to bring awareness to a good cause," she said. "Trey has worked pretty hard, and he's a good son." Both Trey and Ellie have a lot of planning to do. Trey is looking to take up special education as a college major, in addition to playing basketball in the fall. As for Ellie, she can't stop thinking about prom. "Ellie can't wait to go dress shopping" her mother said. "Because I've only told about a million people!" Ellie interjected.
Golden Summary
College-bound basketball star asks girl with Down syndrome to high school prom . Pictures of the two during the "prom-posal" have gone viral .

The extractive summaries described in Table 5.3 are able to produce fluent summaries by extracting sentences directly from the well-written input article. However, they fail to effectively summarise the key narrative events and topics discussed in the article. The TF-IDF summary lacks coherence as it simply strings together extracted sentences without condensing details or conveying the overall narrative arc. TextRank builds an incoherent summary containing repetitive phrasing about tangential details rather than core topics. The LSA summary exhibits a disorganised sentence structure and inadequate topical coverage despite extracting some relevant information. There are also grammatical issues with missing connectors between sentences.

Table 5.3: Comparison of Extractive Model Summaries

Model	**Summary**
TF-IDF	(CNN)He's a blue chip college basketball recruit. class. Ellie interjected.
Text Rank	darla meredith said ellie has struggled with friendships since elementary school, but a special program at eastern called best buddies had made things easier for her.at first glance trey moses and ellie meredith couldn't be more different.both trey and ellie have a lot of planning to do.
Latent Semantic Analysis	ellie interjected.so why is he taking ellie instead?darla meredith said ellie has struggled with friendships since elementary school, but a special program at eastern called best buddies had made things easier for her.

The three methods demonstrate the limitations of extractive-only approaches to summarisation, which often over-retain or over-simplify original text rather than performing thoughtful abstraction and compression.

For sequence-to-sequence (Table 5.4), the article provided as input was relatively long with more than 2000 words. This may have been too much for the encoder to effectively compress into a vector representation, leading to the decoder generating repetitive text. Having a pointer generator, the model could identify and copy some of the salient information from the input article.

Table 5.4: Seq2seq and BART Model Summaries

Model	**Summary**
Sequence-to-sequence model with Pointer Generator network	refund richmond richmond richmond adult longer adult longer ed chant grow advisers long long artificial odinga long long long factor long artificial american empire roosevelt patriarch girl doctor post spit high doctor high medics wound life de urge city options professors cautious go end end xi toward toward pay
BART	Trey Moses asked Ellie Meredith, a freshman with Down syndrome, to be his prom date. Trey made the prom-posal in the gym during Ellieś P.E. class. "Sheś great... she listens and sheś easy to talk to," he said.

BART uses a denoising autoencoder approach to generate fluent text. The model summary is a significant improvement over the extractive pointer generator models. It accurately captures the key details of the article. It demonstrates abstraction by paraphrasing key points rather than extracting sentences verbatim. There is a clear narrative flow and organisation to the summary.

Table 5.5: Comparison of T5 and Fine-tuned T5 Model Summaries

Model	**Summary**
T5	he was originally going to take his girlfriend to Eastern's prom. "She's great... she listens and she's easy to talk to" he said. "I'm so happy for him," Ellie's mom said. I'm so happy for him," Trey's coach said.
Fine-tuned T5	Trey Moses is a star on Eastern High School's basketball team. He was originally going to take his girlfriend to Eastern's prom. He made the prom-posal in the gym during Ellie's P.

Between the T5 variations (Table 5.5), fine-tuning helped produce a more condensed summary by removing extra quotes. This shows the benefits of adaptation to the specific summarisation task. Both summaries capture some important details about the article's original theme fluently without repetitions. The models generate natural

language showcasing abstraction capabilities. There could be some improvement in narrative flow and inclusion of all key points.

Table 5.6: Comparison of Llama-2 and Fine-tuned Llama-2 Model Summaries

Model	Summary
Llama-2	<start>Trey Moses, a college basketball star, asked Ellie Meredith, a high school freshman with Down syndrome, to be his prom date. Trey, who is going to play basketball at Ball State next year, originally planned to take his girlfriend to the prom but changed his mind after meeting Ellie. He made the prom-posal in the gym during Ellieś P.E. class and she tearfully accepted. The teacher at Eastern High School, Trina Helson, posted the photos of Trey and Ellie on Twitter which went viral. According to Trey, Ellie is "great" and "easy to talk to." <end> ’
Fine-tuned Llama-2	Trey Moses, a high school freshman with Down syndrome, asked his classmate, Ellie Meredith, to be his prom date. As a star on Eastern High School’s basketball team, Trey was originally going"

The non-fine-tuned LLAMA-2 (Table 5.6) model generates a fairly accurate and detailed summary of the key events described in the original article. The summary demonstrates paraphrasing of the article rather than direct extraction but fares poorly on conciseness. In contrast, the fine-tuned LLAMA-2 model generates an accurate, concise summary of the topic. Fine-tuning significantly improved the model’s ability to summarise the specific article.

5.3 Quantitative Interpretations

5.3.1 ROUGE

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is a widely used metric for assessing the quality of automatically generated texts by comparing them to reference texts. In particular, this evaluation focuses on the precision, recall and F1 scores achieved by the models.

5.3.1.1 Recall

The ROUGE recall scores presented in Figure 5.6 provide insights into the models ability to incorporate overlapping content from the reference summaries. The ROUGE-1 scores indicate how much of the relevant words (unigram) from the reference text was captured by each model summary. The traditional methods had a low recall, with TF-IDF at 0.2145, TextRank at 0.3087, and LSA at 0.2317. This shows these methods struggled to include all the key details from the reference.

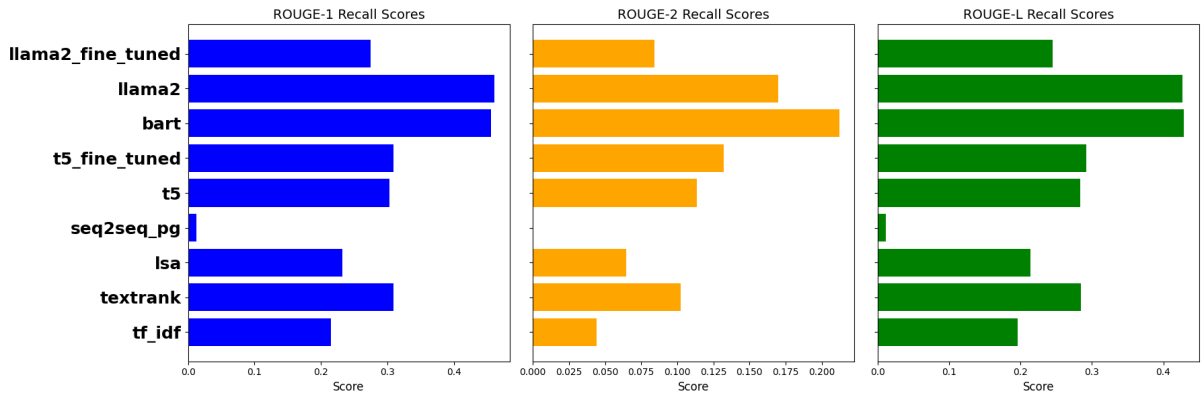


Figure 5.6: Recall ROUGE Scores.

Seq2seq pointer-generator, a neural network trained from scratch, performed the worst with a recall of just 0.0120, demonstrating it was deficient at summarising the salient content. The neural pre-trained and fine-tuned models achieved substantially higher recall scores. The T5 model scored a recall of 0.3029, while the fine-tuned

T5 model improved to 0.3089. BART performed the best with a recall of 0.4556, it was most successful at summarising the relevant details from the reference text. The LLMs were moderately high, with LLAMA-2 at 0.4608 and LLAMA2 fine-tuned at 0.2749.

The ROUGE-2 scores are lower overall compared to the ROUGE-1 scores, with the top scores around 0.21 compared to 0.46 for ROUGE-1. This suggests there is less consecutive bigram overlap between the generated and reference summaries. The gap between the scores is smallest for fine-tuned T5 compared to other models, suggesting it has relatively more bigram overlap despite the overall low scores.

The ROUGE-L score measures the longest common subsequence overlap between the generated and reference summaries. In general, the ROUGE-L scores for the models fall between ROUGE-1 and ROUGE-2, with the highest model score at 0.43. This indicates there is a moderate level of overlap in terms of the word order and longest co-occurring sequences between the generated and reference summaries. The overlap is higher than consecutive bigrams but lower than unigrams.

The consistently high scores for fine-tuned BART demonstrate its strength at retaining key information from the references in a concise summary. Meanwhile, traditional methods like TF-IDF and LSA struggle to recall relevant content compared to neural models. The very low ROUGE scores for seq2seq highlight difficulties in summarising salient points. The transformer models and their fine-tuned versions excel at capturing the core ideas from references.

5.3.1.2 Precision

Precision evaluates relevance of the information contained in the model summaries compared to the reference text. As shown in Figure 5.7 the traditional methods had low to moderate precision, with TF-IDF at 0.1859, TextRank at 0.2347540, and LSA at 0.2135. This indicates these methods included some irrelevant details not found in the reference.

Seq2seq achieved an extremely low precision of 0.0089, showing it generated text unrelated to the reference. The transformer models achieved higher precision. The T5 model scored 0.3423 precision, generating a fairly relevant summary. The fine-tuned T5 model improved substantially to 0.4523 precision and achieved the highest score, indicating its summary closely matched the reference text. BART achieved a precision of 0.3702960. The LLM scores were moderate. LLAMA-2 scored 0.2318, and its fine-tuned version outperformed the base model at 0.2594.

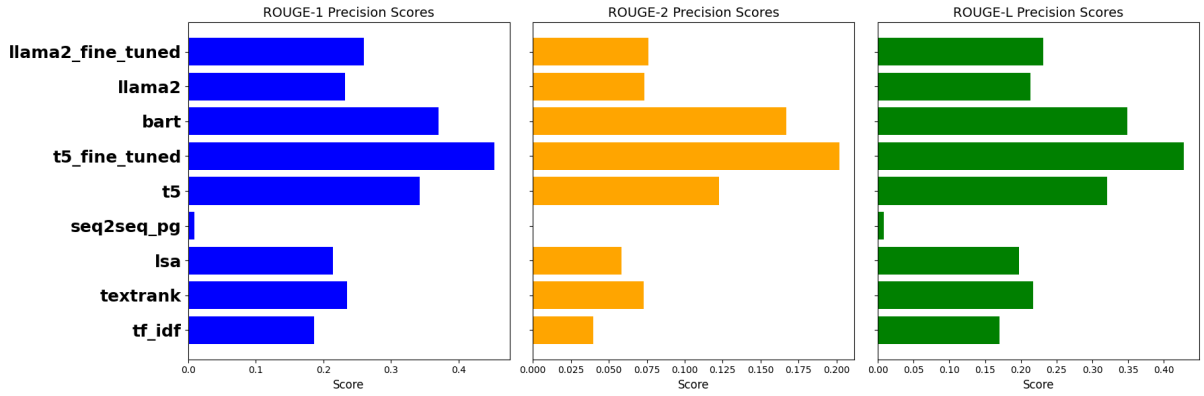


Figure 5.7: Precision ROUGE Scores.

The fine-tuned T5 model has the highest ROUGE-2 precision at 0.2017, significantly higher than the other models. This indicates the model is generating summaries that have a higher overlap with reference summaries in bigram matches. There is a consistency between the precision scores of unigram and bigram. For the ROUGE-L scores, we see a similar trend as ROUGE-1 and 2, with neural models performing better than extractive models. fine-tuned T5 has the highest ROUGE-L score of 0.4280.

The fine-tuning results show that both the LLAMA-2 and T5 models benefited significantly from fine-tuning the specific dataset used for evaluation. For LLAMA-2, the ROUGE-1 score improved from 0.2318 without fine-tuning to 0.2594 after fine-tuning. Similarly, its ROUGE-2 and ROUGE-L scores increased substantially with fine-tuning, from 0.0733 to 0.2135 and 0.0760 to 0.2315, respectively. For the T5 model, fine-tuning boosted its ROUGE-1 score from 0.3423 to 0.4523, its ROUGE-2 from 0.1224 to 0.2017 and its ROUGE-L from 0.3203 to 0.4280. This considerable increase in precision on all three metrics for both LLAMA-2 and T5 after fine-tuning highlights the benefits of adapting these large pre-trained models to the task and dataset.

5.3.1.3 F1 Score

The F1 score balances precision and recall, measuring how accurate and complete the model summaries are compared to the reference text. As presented in Figure 5.8, the traditional methods had low F1 scores, with TF-IDF at 0.1875, TextRank at 0.2517, and LSA at 0.2096. This indicates these methods struggled to generate summaries that were both precise and comprehensive.

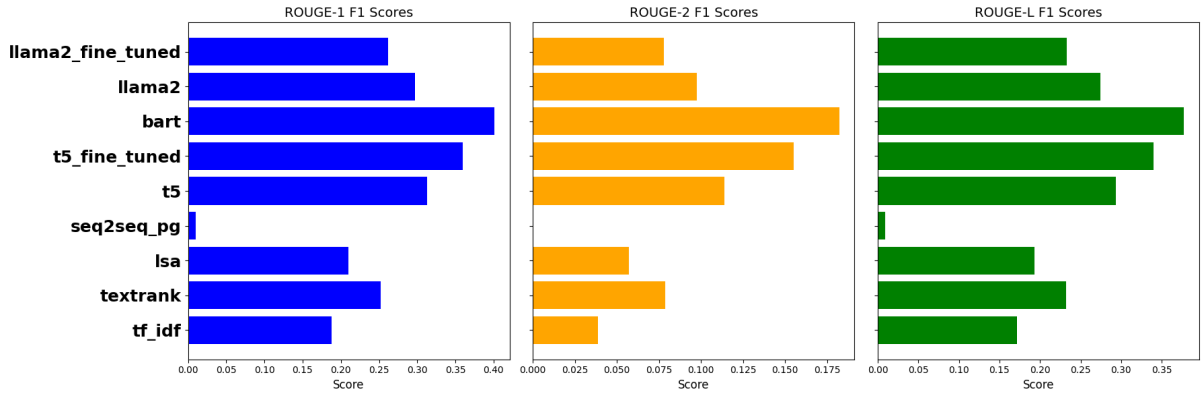


Figure 5.8: F1 ROUGE Scores.

Seq2seq achieved the lowest F1 score of just 0.0099, demonstrating major deficiencies in accurately summarising the salient information. The transformer models attained superior F1 scores overall. The T5 model scored a modest 0.3132, while fine-tuned T5 improved substantially to 0.3591. BART achieved the highest F1 score of 0.4008, it produced the summary with the best balance of precision and recall compared to the reference. The LLMs were moderate, with LLAMA-2 at 0.2973 F1 and LLAMA-2 fine-tuned at 0.2616. BART generated the most accurate and complete summary according to its high F1 score, while seq2seq struggled greatly to produce a precise and comprehensive summary. The neural methods achieved strong F1 scores, but the traditional methods failed to balance precision and recall well according to their low F1 scores.

Across all models, the ROUGE-L F1 scores are generally higher than the ROUGE-2 F1 scores. The fine-tuned T5 model achieved the highest ROUGE-2 score at 0.1822, while the highest ROUGE-L is 0.3773, scored by the same model.

The results also show an interesting contrast between the pre-trained LLAMA-2 model and its fine-tuned version. While the pre-trained LLAMA-2 achieved higher scores of 0.2973, 0.0975 and 0.2743 for LLAMA-1, LLAMA-2 and LLAMA-L respectively, its fine-tuned version scored slightly lower at 0.2616, 0.0780 and 0.2332. This reversal can be attributed to the pre-trained model's failure to maintain conciseness, resulting in longer summaries that recall more n-grams rather than importance. In contrast, the fine-tuned model produced more focused summaries compared to the untargeted pre-trained version, leading to lower recall but improved precision, as evidenced by its higher precision scores and qualitative analysis explained in the previous section. This difference demonstrates the value of fine-tuning, which captures the characteristics of summaries better in terms of recalling important information while remaining concise. The optimised conciseness of the fine-tuned LLAMA-2 model is a desirable quality for a summarisation system. When evaluated side-by-side with human reference summaries, the fine-tuned model summaries would likely be judged as higher quality despite having lower semantic similarity score.

5.3.2 METEOR

Metric for Evaluation of Translation with Explicit Ordering (METEOR) is another common evaluation metric that measures how well generated summaries match reference summaries. It calculates similarity based on explicit word-to-word matches, stemmed matches and semantic matches using WordNet synonyms.

The results illustrated in Figure 5.9 show the several methods utilised to score model's performance on the test data. TF-IDF yielded a score of 0.1946, indicating a lower level of relevance between the model's summary and the reference summary. TextRank produced a higher score of 0.3091 among extractive methods, suggesting the model summary contained more of the salient information from the reference. LSA resulted in a score of 0.2334, showing the model text partially captured the semantic concepts in the reference and performed better than TF-IDF approach.

The seq2seq pointer-generator model obtained the lowest score of 0.0178984, showing it struggled to accurately summarise the reference text. The neural pre-trained and fine-tuned models achieved higher scores. The T5 and

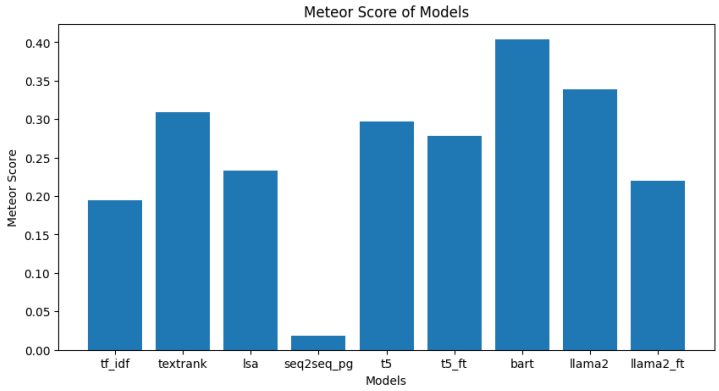


Figure 5.9: METEOR Scores.

fine-tuned T5 models scored 0.2964 and 0.2779, respectively, demonstrating a better ability to generate a relevant summary. BART achieved the highest score of 0.4038, indicating it generated the most relevant summary according to the reference. The LLAMA-2 and its fine-tuned version performed moderately well, with scores of 0.3392 and 0.219, respectively. The neural methods, particularly BART, were most capable of producing an informative summary. The traditional methods had more difficulty capturing the salient information.

It is worth noting that the fine-tuned models scored lower on the METEOR scale than the pre-trained versions. While the pre-trained models may have higher semantic similarity due to generating longer, verbose summaries, the fine-tuned models produce more concise and focused summaries. Fine-tuning allows the base models to better capture the characteristics of quality summaries for a specific dataset, generating only the most salient information rather than all possibly related concepts.

5.4 User Feedback

Manual evaluation was conducted to assess the performance of various text summarisation models on event, topic and article summaries. The evaluation rated summaries on four key criteria - coherence, fluency, redundancy and coverage of salient information. The criteria were rated on a 5-point scale explained in appendix C.1, with 1 being the lowest rating and 5 being the highest.

5.4.1 Event Summaries

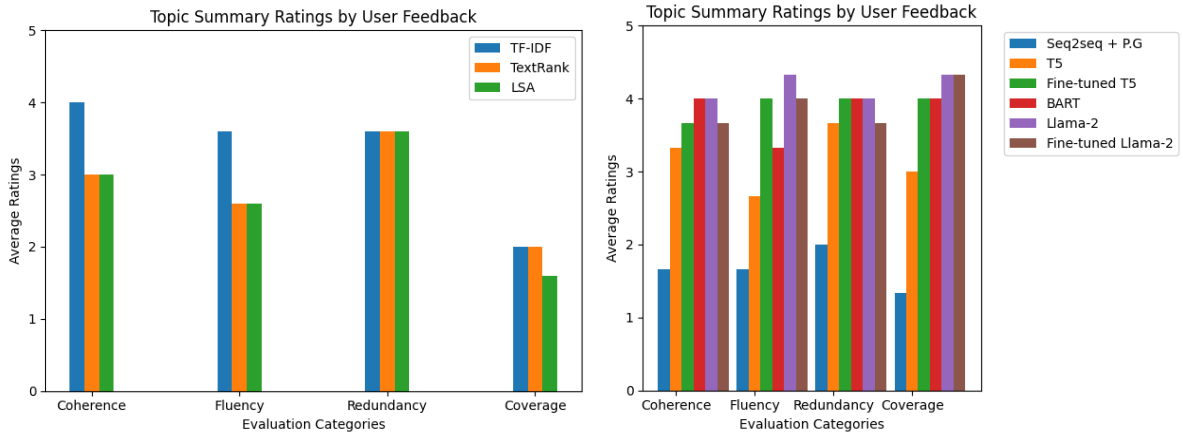
For event summaries generated using the BART model, coherence received an average rating of 3.3. This indicates the summaries were moderately coherent, with some logical connections between ideas, but parts were still disjointed. Fluency received a similar average rating of 3.6, suggesting moderate fluency but contained some awkwardness. Redundancy was rated 4.6 on average, showing no repetition. Coverage of salient information had an average of 3, meaning some key details to understand an event were missing.

5.4.2 Topic Summaries

The extractive summarisation models generated topic summaries that lacked topic coverage according to user evaluations (Figure 5.10a). TF-IDF scored 4 for coherence and 3.6 for fluency on average, indicating a better flow of sentences. TextRank and LSA performed worse, with averages around 3 for coherence and 2.6 for fluency. These extractive approaches could not perform well in producing logical, natural-sounding summaries for short topic-length text.

Redundancy was slightly better for the extractive techniques (Figure 5.10b), averaging between 3.6 to 3.8, while coverage of key details was severely lacking. All three models scored only 2 on average for coverage of salient information. The extractive methods seem ineffective at identifying and summarising the most important content from the topic text.

In contrast, the abstractive models produced more coherent, fluent topic summaries. The baseline T5 model achieved stronger coherence and fluency scores of 3.33. Fine-tuning T5 on domain-specific data further improved performance to 3.66 and 4 respectively. Similarly, fine-tuned and pre-trained LLAMA-2's coherence averaged around 3.66 to 4 and fluency from 4 to 4.33.

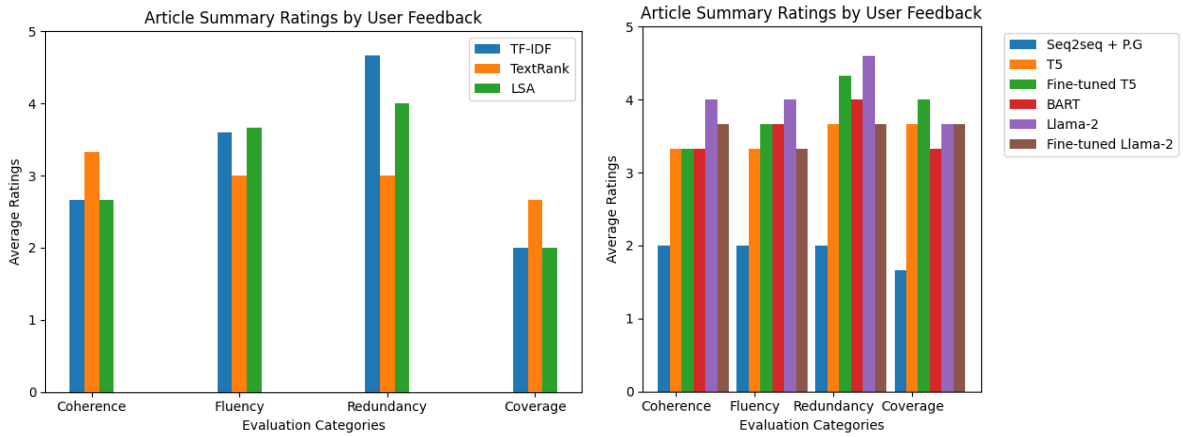


(a) Extractive Summaries. (b) Abstractive Summaries.

Figure 5.10: Manual Evaluation of Topic Summaries,

5.4.3 Article Summaries

For article summaries, the extractive summarisation techniques struggled to produce coherent (Figure 5.11a), fluent article summaries based on user ratings. TF-IDF received low average scores of 2.67 for coherence and 3.6 for fluency, while TextRank was rated slightly higher at 3.33 and 3 for those categories. However, both models scored poorly on coverage of salient information with averages below 3, indicating they did not effectively identify and summarise the most important details.



(a) Extractive Summaries. (b) Abstractive Summaries.

Figure 5.11: Manual Evaluation of Article Summaries.

In contrast, the abstractive models generated more coherent and fluent summaries (Figure 5.11b). The baseline T5 model achieved stronger ratings of 3.33 for coherence and 3.33 for fluency. Fine-tuning T5 on a large dataset further improved performance, raising coherence to 3.33 and fluency to 3.66 on average. Similarly, BART scored 3.33 and 3.66, while base LLAMA-2 scored well on fluency with an average score of 4.

The abstractive models pre-training on large corpora seems to allow them to produce more natural language and logical flow compared to the extractive techniques. Fine-tuning them further helps the models capture nuances of fluent writing in the target domain.

Chapter 6

Discussion

The results detailed in the previous chapter demonstrated that fine-tuning Transformers and Large Language MOdels on domain-specific datasets significantly improved their ability to generate summaries. Both T5 and LLAMA-2 showed a substantial increase in the quality of summaries generated after fine-tuning on the CNN/DailyMail news dataset, compared to their pre-trained versions. This aligns with findings from Zhang et al. (2020), which showed that adapting models like PEGASUS to downstream summarisation tasks through fine-tuning leads to considerable performance gains. Similarly, Lewis et al. (2020) found that fine-tuning BART models on in-domain corpora enhanced summarisation capabilities compared to off-the-shelf models. Our results reinforce these conclusions, demonstrating the value of task-specific adaptation even for massive pre-trained general language models.

Fine-tuning allowed the model to learn nuances and characteristics specific to the dataset genre and writing style. The CNN/DailyMail news articles have a particular structure and narrative style. Fine-tuning exposed the models to learn these consistent patterns, allowing them to specialise in generating summaries matching these linguistic qualities. As Raffel et al. (2020) discussed in their paper, large pre-trained models exhibit significant variability in performance across datasets and tasks. Their capabilities on one domain do not automatically transfer to other distributions without adaptation. Our experiments align with this observation - strong pre-training and general large language models alone are insufficient, task-specific fine-tuning is essential to align the model with target data and reach optimal results.

Qualitative analysis also revealed fine-tuning improves coherence, fluency and coverage of salient details compared to pre-trained models. This aligns with findings from Zhong et al. (2020), who showed domain-specific fine-tuning generates more coherent and focused summaries compared to general pre-training. Our human evaluation similarly found fine-tuned model summaries to be more coherent, fluent and representative of key information.

The fine-tuned LLAMA-2 model is particularly noteworthy and a highlight of this project, as it represents the first demonstration of effectively adapting this new 7 billion parameter model for abstractive summarisation. By fine-tuning LLAMA-2 on just 1000 examples from CNN/DailyMail for one epoch, we were able to specialise its broad capabilities for generating news summaries.

The results showed training abstractive summarisation models from scratch faced major challenges like insufficient language comprehension and generating incoherent summaries, requiring enormous datasets. In contrast, pre-trained transformer models and their fine-tuned versions vastly outperform models trained from scratch. Recent benchmarks also demonstrate transformers have substantially advanced the state-of-the-art in abstractive summarisation through pre-training and fine-tuning [1]. This highlights their advantages of strong language understanding over training summarisers from scratch and their promise as the leading approach for robust summarisation.

Comparing extractive and abstractive models, the project findings reveal greater capabilities of neural abstractive approaches over extraction-based methods. Although computationally simpler, extractive methods struggle to condense verbose input articles into concise summaries that read fluently while retaining key details. The extracted sentences often lacked coherence when combined, as they were selected independently based on sentence importance without regard for overall flow and transitions. Human evaluations also rated lower on coverage of key details compared to abstractive models.

Extractive methods provided a useful unsupervised baseline for benchmarking neural models. As Nallapati, Zhou, et al. (2016) discuss, extraction establishes a low-complexity approach to identifying salient sentences without training data. The inclusion of fundamental extractive techniques followed recommendations from past summarisation research (Gambhir et al., 2017) for thorough evaluation by testing sophisticated methods against simple unsupervised baselines.

This thesis made contributions in comprehensively evaluating extractive and abstractive summarisation techniques on long-form text. Both statistical baselines and cutting-edge neural models were systematically compared

[1] https://paperswithcode.com/sota/document-summarization-on-cnn-daily-mail

using ROUGE and METEOR metrics along with human feedback on a news corpus. The implemented models and detailed results provide a framework for researchers to select an appropriate summarisation approach based on their specific accuracy, efficiency and abstraction requirements.

This project also created an innovative web dashboard for interactive text susummarisation through three creative features: an event summariser, that consolidates information across news articles on an event into a multi-document summary; a customisable topic summariser, that allows customising the summary length, model and sentence extraction for any entity and provides an understanding of topic comprehensively from credible sources; and a custom article summariser, that summarises arbitrary text on demand while providing analytics for generated summary. The dashboard application enabled non-experts to interactively explore multi-document summarisation. The integration of diverse models within a unified interface is novel and offers a way to compare capabilities.

6.1 Limitations

A limitation of the project is the exclusive focus on the English language and news domain, which constrains generalisation. Expanding the datasets to other languages and genres could reveal whether findings transfer across domains. The experiments focused on news articles of a few hundred words. Summarising much longer documents like research papers or books poses challenges due to limitations in processing lengthy sequences. Testing on diverse corpora would further validate findings. The models could be expanded to handle longer input documents.

The posterior divergence problem in Large Language Models causes unnecessary hallucinations, requiring improved training techniques. The abstractive summarisation models also currently lack automatic semantic verification capabilities, sometimes generating factually inconsistent or hallucinated content that cannot be detected. Integrating modules to check summary faithfulness against knowledge bases and human feedback could improve reliability.

Chapter 7

Conclusion and Future Work

This research aimed to advance text summarisation capabilities by implementing and thoroughly evaluating a diverse range of techniques on a news article dataset. Both extractive algorithms and neural abstractive models were systematically compared using quantitative metrics and qualitative human assessments. The web application presents an end-to-end implementation applying research to practical usage.

The study demonstrates that current state-of-the-art abstractive summarisation models significantly outperform traditional extraction-based methods in generating coherent, relevant summaries. Adaptation of large pre-trained language models like T5 and LLAMA-2 through supervised fine-tuning led to considerable gains in summarisation quality. Meanwhile, fundamental extractive techniques struggled to produce concise, fluent summaries while retaining key details. Qualitative human evaluations further verified the advantages of fine-tuned abstractive models in coherence, fluency and coverage of salient content compared to extraction methods.

A substantial achievement of this project was successfully fine-tuning the novel, cutting-edge LLAMA-2 model for abstractive summarisation for the first time, signifying a major task-specific implementation for the largest state-of-the-art language model developed to date. Efficient techniques like LoRA enabled rapid adaption by updating only a small fraction of the parameters.

Overall, the research substantiates the greater maturity of abstractive summarisation techniques, with fine-tuning pre-trained models being the dominant area of focus. The work provides a framework for standardised evaluation of diverse approaches using both human and automated assessments.

7.1 Future Scope

The simplicity of adapting LLAMA-2 using techniques like LoRA could open up summarisation-like applications to practitioners without having access to massive compute. As Touvron, Martin, et al. (2023) discuss in their newly released paper on LLAMA-2, Large Language Models still benefit from task-specific tuning despite extensive pre-training, especially for complex text generation tasks. Our results provide a proof of concept for rapidly adapting summarisation expertise into these gigantic models. Future work should assess performance on low-resource summarisation and few-shot learning scenarios. Recent models have shown promise in these settings.

Evaluating the chronological summarisation of evolving news stories over time is another valuable direction, as past work has focused mainly on static documents. An interesting area for future work would be assessing how well the models can summarise new articles as they emerge about an unfolding event and incrementally integrate the new information with previous summaries to build a coherent chronological timeline.

The goal outlined in the scope was to develop an automated multi-document summarisation pipeline leveraging credible news articles to produce concise overviews of events.The results reveal current capabilities and limitations to guide future research towards matching human-level language understanding for high-quality summarisation. The conclusion summarises that this goal was successfully accomplished through a modular summarisation system evaluated on a news corpus. The conclusion also highlights building an innovative interactive dashboard allowing the exploration of diverse summarisation methods as a key achievement. Limitations acknowledged include the exclusive focus on English language news articles, the posterior divergence problem and the lack of semantic verification of generated summaries. Overall, the stated goal of an automated summarisation pipeline generating event summaries from credible journalism was fulfilled.

References

Akiyama, Kazuki, Akihiro Tamura, and Takashi Ninomiya (June 2021). "Hie-BART: Document Summarization with Hierarchical BART". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Student Research Workshop*. Online: Association for Computational Linguistics, pp. 159–165. DOI: 10.18653/v1/2021.naacl-srw.20. URL: https://aclanthology.org/2021.naacl-srw.20.

Anderson, Peter et al. (2016). "SPICE: Semantic Propositional Image Caption Evaluation". In: *ECCV*.

Bandara, Kasun, Christoph Bergmeir, and Slawek Smyl (2020). "Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach". In: *Expert Systems with Applications* 140, p. 112896. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.112896. URL: https://www.sciencedirect.com/science/article/pii/S0957417419306128.

Banerjee, Satanjeev and Alon Lavie (2005). "METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments". In: *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. Association for Computational Linguistics.

Barzilay, Regina and Michael Elhadad (1997). "Using Lexical Chains for Text Summarization". In: *Intelligent Scalable Text Summarization*. URL: https://aclanthology.org/W97-0703.

Bengio, Yoshua et al. (2003). "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research* 3. Submitted 4/02; Published 2/03, pp. 1137–1155.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan (Mar. 2003). "Latent Dirichlet Allocation". In: *J. Mach. Learn. Res.* 3.null, pp. 993–1022. ISSN: 1532-4435.

Brown, Tom B. et al. (2020). *Language Models are Few-Shot Learners*. arXiv: 2005.14165 [cs.CL].

Chopra, Sumit, Michael Auli, and Rush (2016). "Abstractive sentence summarization with attentive recurrent neural networks". In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 93–98.

Chowdhery, Aakanksha et al. (2022). "PaLM: Scaling language modeling with pathways". In: *arXiv preprint arXiv:2204.02311*.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805*.

Dong, Li et al. (2019). *Unified Language Model Pre-training for Natural Language Understanding and Generation*. arXiv: 1905.03197 [cs.CL].

Edmundson, Harold P (1969). "New methods in automatic extracting". In: *Journal of the ACM (JACM)* 16.2, pp. 264–285.

Erkan, G"unes and Radev (2004). "Lexrank: Graph-based lexical centrality as salience in text summarization". In: *Journal of artificial intelligence research* 22, pp. 457–479.

Evangelopoulos, Nicholas, Xiaoni Zhang, and V.R. Prybutok (Dec. 2012). "Latent Semantic Analysis: Five Methodological Recommendations". In: *European Journal of Information Systems* 21, pp. 70–86. DOI: 10.1057/ejis.2010.61.

Foltz, Peter W (1996). "Latent semantic analysis for text-based research". In: *Behavior Research Methods, Instruments, & Computers* 28.2, pp. 197–202.

Gambhir, Mahak and Vishal Gupta (Jan. 2017). "Recent automatic text summarization techniques: a survey". In: *Artificial Intelligence Review* 47.1, pp. 1–66. ISSN: 1573-7462. DOI: 10.1007/s10462-016-9475-9. URL: https://doi.org/10.1007/s10462-016-9475-9.

Giannakopoulos, George and Vangelis Karkaletsis (Jan. 2011). "AutoSummENG and MeMoG in Evaluating Guided Summaries". In.

Gong, Yihong and Xin Liu (2001). "Generic text summarization using relevance measure and latent semantic analysis". In: *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 19–25.

Hans, Christian, Agus Mikhael, and Suhartono Derwin (2016). "Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF)". In: *ComTech: Computer, Mathematics and Engineering Applications* 7.4, pp. 285–296.

Harris, Zellig S. (1954). "Distributional Structure". In: *WORD* 10.2-3, pp. 146–162. DOI: 10.1080/00437956.1954.11659520. eprint: https://doi.org/10.1080/00437956.1954.11659520. URL: https://doi.org/10.1080/00437956.1954.11659520.

Hochreiter, Sepp and J"urgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hou, Sheng-Luan et al. (2021). "A Survey of Text Summarization Approaches Based on Deep Learning". In: *Journal of Computer Science and Technology* 36.3, pp. 633–663.

Hovy, Eduard and Chin-Yew Lin (1998). "Automated text summarization and the Summarist system". In: *TIPSTER TEXT PROGRAM PHASE III: Proceedings of a Workshop held at Baltimore, Maryland, October 13–15, 1998*, pp. 197–214.

Hu, Edward J. et al. (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv: 2106.09685 [cs.CL].

Jones, Karen Sparck (1998). "Automatic summarising: factors and directions". In: *arXiv preprint cmp-lg/9805011*.

Landauer, Thomas K (2007). "LSA as a theory of meaning". In: *Handbook of latent semantic analysis* 3, p. 32.

Lavie, Alon and Abhaya Agarwal (2007). "METEOR: An Automatic Metric for MT Evaluation with High Levels of Correlation with Human Judgments". In: *Proceedings of the Second Workshop on Statistical Machine Translation*. Association for Computational Linguistics.

Lewis, Mike et al. (2020). "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *arXiv preprint arXiv:1910.13461*.

Li, Chenliang, Weiran Xu, et al. (2018). "Guiding generation for abstractive text summarization based on key information guide network". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 55–60.

Li, Yunghui, Latifa Nabila Harfiya, et al. (Sept. 2020). "Real-Time Cuffless Continuous Blood Pressure Estimation Using Deep Learning Model". In: *Sensors* 20. DOI: 10.3390/s20195606.

Lin, Chin-Yew (2004). "ROUGE: A Package for Automatic Evaluation of Summaries". In: *Text Summarization Branches Out*. Association for Computational Linguistics.

Lin, Chin-Yew and Eduard Hovy (1997). "Identifying topics by position". In: *Proceedings of the fifth conference on Applied natural language processing*, pp. 283–290.

Liu, Yang and Mirella Lapata (2019). "Text summarization with pretrained encoders". In: *arXiv preprint arXiv:1908.08345*.

Luhn, HP (1958). "The automatic creation of literature abstracts". In: *IBM Journal of research and development* 2.2, pp. 159–165.

Ma, Congbo et al. (2022). "Multi-document summarization via deep learning techniques: A survey". In: *ACM Computing Surveys* 55.5, pp. 1–37.

Mangrulkar, Sourab et al. (2022). *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods*. https://github.com/huggingface/peft.

Martin, Dian I and Michael W Berry (2007). "Mathematical foundations behind latent semantic analysis". In: *Handbook of latent semantic analysis*, pp. 35–55.

Mihalcea, Rada and Paul Tarau (2004). "Textrank: Bringing order into text". In: *Proceedings of the 2004 conference on empirical methods in natural language processing*, pp. 404–411.

Moratanch, N and S Chitrakala (2017). "A survey on extractive text summarization". In: *2017 international conference on computer, communication and signal processing (ICCCSP)*. IEEE, pp. 1–6.

Nallapati, Ramesh, Feifei Zhai, and Bowen Zhou (2017). "Summarunner: A recurrent neural network based sequence model for extractive summarization of documents". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1.

Nallapati, Ramesh, Bowen Zhou, et al. (Aug. 2016). "Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond". In: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, pp. 280–290. DOI: 10.18653/v1/K16-1028. URL: https://aclanthology.org/K16-1028.

Ozsoy, Makbule Gulcin, Ferda Nur Alpaslan, and Ilyas Cicekli (2011). "Text summarization using latent semantic analysis". In: *Journal of Information Science* 37.4, pp. 405–417.

Pan, Shirui et al. (2023). "Unifying Large Language Models and Knowledge Graphs: A Roadmap". In: *arXiv preprint arXiv:2306.08302*.

Papineni, Kishore et al. (July 2002). "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040.

Radev, Dragomir R, Hongyan Jing, Małgorzata Styś, et al. (2004). "Centroid-based summarization of multiple documents". In: *Information Processing & Management* 40.6, pp. 919–938.

Radev, Hongyan Jing, and Malgorzata Budzikowska (2000). "Centroid-based summarization of multiple documents: sentence extraction, utility-based evaluation, and user studies". In: *NAACL-ANLP 2000 Workshop: Automatic Summarization*. URL: https://aclanthology.org/W00-0403.

Radford, Alec et al. (2019). "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8, p. 9.

Raffel, Colin et al. (2020). "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *The Journal of Machine Learning Research* 21.1, pp. 5485–5551.

Rohde, Tobias, Xiaoxia Wu, and Yinhan Liu (2021). *Hierarchical Learning for Generation with Long Source Sequences*. arXiv: 2104.07545 [cs.CL].

Rosset, Corby (Feb. 2020). *Turing-NLG: A 17-billion-parameter language model by Microsoft*. URL: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/.

Rush, Sumit Chopra, and Jason Weston (Sept. 2015). "A Neural Attention Model for Abstractive Sentence Summarization". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 379–389. DOI: 10.18653/v1/D15-1044. URL: https://aclanthology.org/D15-1044.

Saggion, Horacio and Guy Lapalme (Dec. 2002). "Generating Indicative-Informative Summaries with SumUM". In: *Computational Linguistics* 28.4. _eprint: https://direct.mit.edu/coli/article-pdf/28/4/497/1797865/089120102762671963.pdf, pp. 497–526. ISSN: 0891-2017. DOI: 10.1162/089120102762671963. URL: https://doi.org/10.1162/089120102762671963.

Salton, Gerard and Christopher Buckley (1988). "Term-weighting approaches in automatic text retrieval". In: *Information processing & management* 24.5, pp. 513–523.

See, Abigail, Liu, and Christopher Manning (2017). "Get to the point: Summarization with pointer-generator networks". In: *arXiv preprint arXiv:1704.04368*.

Sparck Jones, Karen (1972). "A statistical interpretation of term specificity and its application in retrieval". In: *Journal of documentation* 28.1, pp. 11–21.

Touvron, Hugo, Thibaut Lavril, et al. (2023). *LLaMA: Open and Efficient Foundation Language Models*. arXiv: 2302.13971 [cs.CL].

Touvron, Hugo, Louis Martin, et al. (2023). "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv preprint arXiv:2307.09288*.

Vaswani, Ashish et al. (2017). "Attention is all you need". In: *Advances in neural information processing systems*, pp. 5998–6008.

Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly (2015). *Pointer Networks*. arXiv: 1506.03134 [stat.ML].

Wang, Chenglong et al. (June 2023). "Low-rank Adaptation Method for Wav2vec2-based Fake Audio Detection". In.

Zaheer, Manzil et al. (2021). *Big Bird: Transformers for Longer Sequences*. arXiv: 2007.14062 [cs.LG].

Zhang, Jingqing et al. (2020). "PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization". In: *arXiv preprint arXiv:1912.08777*.

Zhong, Ming et al. (2020). *Extractive Summarization as Text Matching*. arXiv: 2004.08795 [cs.CL].

Appendix A

Code

A.1 Git Repository

https://git.cs.bham.ac.uk/projects-2022-23/vxj250

A.2 Reference for Submitted Code

All codes included in the report were written by me with reference from the following sites:

Sequence-to-sequence

Encoder-Decoder: https://github.com/karanthakkar97/Text-summarization-with-Seq2Seq

T5: https://huggingface.co/docs/transformers/model_doc/t5

Fine-tuned T5: https://huggingface.co/docs/transformers/training

BART: https://huggingface.co/facebook/bart-large-cnn

LLAMA-2: https://github.com/facebookresearch/llama

Fine-tuned LLAMA-2:

PEFT: https://huggingface.co/docs/peft/task_guides/clm-prompt-tuning (Mangrulkar et al., 2022)

LoRA: https://huggingface.co/docs/diffusers/training/lora

Dashboard: https://dash.plotly.com/

A.3 Project Structure

The Project contains separate folders for each of the implemented models and dashboard applications. Each folder for the model has Python implementation using .ipynb file, batched outputs(if applicable), summary file zipped, markdown files(if applicable), requirement file, metrics and fine-tuned models(if applicable).

1. aroha
2. bart
3. dash_app
4. eda
5. llama2
6. llama2_fine_tuned

7. lsa
8. seq2seq_pg
9. t5
10. t5_fine_tuned
11. text_rank
12. tf_idf
13. utils

A.4 Running the Application Code

1. Fist step is to install the following python libraries from the pip command:

```
python -m pip install -U pip
pip install -r requirements.txt
```

   a. To run individual summarisation model, navigate to the specific folder and install packages from the specific requirements.txt file

   b. To run dashboard application, navigate to dash_app folder and execute requirements.txt

2. Download CNN/Dailymail dataset from HuggingFace using either the curl command or python library

   a. Curl Command:

```
curl -X GET \
"https://datasets-server.huggingface.co/splits?dataset=cnn_dailymail"
```

   b. Python Library:

```
from datasets import load_dataset
dataset = load_dataset('cnn_dailymail', '3.0.0')
```

3. Configure config.txt file

   a. Navigate to config.txt file in dash_app folder

   b. Replace the configuration parameters with the desired paths

```
named_entities,/content/drive/MyDrive/aroha/eda/named_entities/
bart_tokenizer_cache_dir_path,/content/drive/MyDrive/aroha/bart_cache
bart_model_cache_dir_path,/content/drive/MyDrive/aroha/bart_cache
t5_tokenizer_cache_dir_path,/content/drive/MyDrive/aroha/t5_cache
t5_model_cache_dir_path,/content/drive/MyDrive/aroha/t5_cache
t5_fine_tuned_model_path,/content/drive/MyDrive/aroha/t5_transfer_learning/model
llama2_access_token_path,XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
llama2_tokenizer_cache_dir_path,/content/drive/MyDrive/aroha/llama2_cache
llama2_model_cache_dir_path,/content/drive/MyDrive/aroha/llama2_cache
peft_model_path,/content/drive/MyDrive/aroha/llama2_fine_tuned/model
seq2seq_encoder_model_path,/content/drive/MyDrive/aroha/seq2seq_pg/encoder_model.h5
seq2seq_decoder_model_path,/content/drive/MyDrive/aroha/seq2seq_pg/decoder_model.h5
summary_tokenizer_path,/content/drive/MyDrive/aroha/seq2seq_pg/summary_tokenizer.pickle
article_tokenizer_path,/content/drive/MyDrive/aroha/seq2seq_pg/article_tokenizer.pickle
summary_vocabulary_path,/content/drive/MyDrive/aroha/seq2seq_pg/summary_vocabulary.json
```

c. Contents for "named_entities" is available in eda folder

d. Contents for "t5_fine_tuned_model_path" ia available at "t5_transfer_learning model" folder and can be accessed from https://drive.google.com/drive/folders/1HrrRRJdHtBRefazyLKB7PB1mXP-4rNsr?usp=sharing

e. "llama2_fine_tuned" model has been published to HuggingFace and can be downloaded directly

```
config = PeftConfig.from_pretrained("vijayjawali/llama2_query_tuned")
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b-chat-hf")
model = PeftModel.from_pretrained(model, "vijayjawali/llama2_query_tuned")
```

f. Contents for summary_tokenizer_path, article_tokenizer_path and summary_vocabulary_path are avilable in seq2seq_pg folder

g. Contents for Encoder and Decoder models are available at https://drive.google.com/drive/folders/14TpP1ivxGL5WkOhVI

4. Get Llama-2 Aceess from Meta and HuggingFace

   a. Open Request Form: https://ai.meta.com/resources/models-and-libraries/llama-downloads/

   b. Enter details and wait to get access, it should take 2-3 hours

   c. Request the same from HuggingFace: https://huggingface.co/meta-llama

   d. After having both the access, navigate to settings page in HuggingFace and create a read token: https://huggingface.co/settings/tokens

   e. Replace the config parameter llama2_access_token_path with the token received

5. After the configuration file is setup, navigate to dash_app folder, it contains two files, dash_app.py and run_dash_app.ipynb

   a. copy dash_app.py and place it in the execution environment

   b. open run_dash_app.ipynb and navigate to last line containing python execution code

```
! python /content/drive/MyDrive/aroha/dash_app/dash_app.py
```

   c. replace the python execution file location with dash_app.py path

   d. run_dash_app executes the dash_app.py file to run dashboard application

6. Get Ngrok authtoken from https://dashboard.ngrok.com/auth and replace the existing token with the one generated from ngrok

```
NGROK_AUTH_TOKEN = "xxxxxxxxxxxxxxxxxxxxx"
pyngrok.set_auth_token(NGROK_AUTH_TOKEN)
```

7. Execute run_dash_app.ipynb file

   a. It is recommended to use NVIDIA A100 in colab for smooth execution.

   b. Alternative T4 GPU can be used to execute.

   c. Copy public url logged by ngrok

```
logger.info(f"Dash app:{ngrok_tunnel.public_url}")
```

   d. Open the copied link in browser to access dashboard application

Appendix B

Qualitative Evaluation

B.1 Qualitative Evaluation

Table B.1: Extended Original Article and Golden summary

Original Article
(CNN)He's a blue chip college basketball recruit. She's a high school freshman with Down syndrome. At first glance Trey Moses and Ellie Meredith couldn't be more different. But all that changed Thursday when Trey asked Ellie to be his prom date. Trey – a star on Eastern High School's basketball team in Louisville, Kentucky, who's headed to play college ball next year at Ball State – was originally going to take his girlfriend to Eastern's prom. So why is he taking Ellie instead? "She's great... she listens and she's easy to talk to" he said. Trey made the prom-posal (yes, that's what they are calling invites to prom these days) in the gym during Ellie's P.E. class. Trina Helson, a teacher at Eastern, alerted the school's newspaper staff to the prom-posal and posted photos of Trey and Ellie on Twitter that have gone viral. She wasn't surpristed by Trey's actions. "That's the kind of person Trey is," she said. To help make sure she said yes, Trey entered the gym armed with flowers and a poster that read "Let's Party Like it's 1989," a reference to the latest album by Taylor Swift, Ellie's favorite singer. Trey also got the OK from Ellie's parents the night before via text. They were thrilled. "You just feel numb to those moments raising a special needs child," said Darla Meredith, Ellie's mom. "You first feel the need to protect and then to overprotect." Darla Meredith said Ellie has struggled with friendships since elementary school, but a special program at Eastern called Best Buddies had made things easier for her. She said Best Buddies cultivates friendships between students with and without developmental disabilities and prevents students like Ellie from feeling isolated and left out of social functions. "I guess around middle school is when kids started to care about what others thought," she said, but "this school, this year has been a relief." Trey's future coach at Ball State, James Whitford, said he felt great about the prom-posal, noting that Trey, whom he's known for a long time, often works with other kids . Trey's mother, Shelly Moses, was also proud of her son. "It's exciting to bring awareness to a good cause," she said. "Trey has worked pretty hard, and he's a good son." Both Trey and Ellie have a lot of planning to do. Trey is looking to take up special education as a college major, in addition to playing basketball in the fall. As for Ellie, she can't stop thinking about prom. "Ellie can't wait to go dress shopping" her mother said. "Because I've only told about a million people!" Ellie interjected.
Golden Summary
College-bound basketball star asks girl with Down syndrome to high school prom . nPictures of the two during the "prom-posal" have gone viral .

Appendix C

User Feedback

C.1 Scoring metrics

Table C.1: Explanation of Coherence Points

Points	Coherence Level
1 star	Incoherent and disjointed. No logical flow or connection between sentences.
2 stars	Disjointed and difficult to read fluently due to poor grammar, word usage, or sentence structure.
3 stars	Moderately coherent with some logical connections between ideas and a build-up of concepts.
4 stars	Good coherence with a logical flow and progression of ideas that clearly connect to each other.
5 stars	Highly coherent and unified. Smooth transitions between sentences and a logical build-up of concepts.

Table C.2: Explanation of Fluency Points

Points	Fluency Level
1 star	Disjointed and very difficult to read fluently due to poor grammar, word usage, or sentence structure.
2 stars	Some fluency issues with awkward phrases, poor word choices, and unnatural sentence structures that make it choppy.
3 stars	Moderate fluency. Sentences generally read well but some parts are awkward or choppy.
4 stars	Good fluency overall. Most sentences flow well with good grammar, word usage, and sentence structure.
5 stars	Reads extremely fluently with smooth, natural sentences and excellent grammar, word choice, and sentence structure.

Table C.3: Explanation of Redundancy Points

Points	Redundancy Level
1 star	Contains a lot of repetitive information and redundant ideas that add no new information.
2 stars	Some redundant sentences that repeat information unnecessarily.
3 stars	A little redundancy, but most information is non-repetitive.
4 stars	Minimal redundancy, with almost all information presented uniquely.
5 stars	Contains no redundant information, and all content is unique and non-repetitive.

Table C.4: Explanation of Topic Coverage Points

Points	Topic Coverage Level
1 star	Missing almost all of the key points and most salient information from the source.
2 stars	Missing many key points and fails to cover much salient event information.
3 stars	Covers some salient information but misses several important key points about the event.
4 stars	Covers most salient information and key points but is missing a few details.
5 stars	Comprehensively covers almost all salient information and key points about the event.

C.2 Topic summary scores

Table C.5: Average for Extractive Text Summarisation Model Topic Summaries

Model	Coherence	Fluency	Redundancy	Coverage
TF-IDF	4	3.6	3.6	2
TextRank	3	2.6	3.6	2
LSA	3	2.6	3.6	1.6

Table C.6: Average for Abstractive Text Summarisation Model Topic Summaries

Model	Coherence	Fluency	Redundancy	Coverage
Seq2seq + P.G	1.66	1.66	2	1.33
T5	3.33	2.66	3.66	3
Fine-tuned T5	3.66	4	4	4
BART	4	3.33	4	4
Llama-2	4	4.33	4	4.33
Fine-tuned Llama-2	3.66	4	3.66	4.33

C.3 Article summary scores

Table C.7: Average for Extractive Text Summarisation Model Article Summaries

Model	Coherence	Fluency	Redundancy	Coverage
TF-IDF	2.667	3.6	4.66	2
TextRank	3.33	3	3	2.66
LSA	2.66	3.66	4	2

Table C.8: Average for Abstractive Text Summarisation Model Article Summaries

Model	Coherence	Fluency	Redundancy	Coverage
Seq2seq + P.G	2	2	2	1,66
T5	3.33	3.33	3.66	3.66
Fine-tuned T5	3.33	3.66	4.33	4
BART	3.33	3.66	4	3.33
Llama-2	4	4	4.6	3.66
Fine-tuned Llama-2	3.66	3.33	3.66	3.66