Received: 1 June 2025 | Revised: 12 January 2026 | Accepted: 7 May 2026 | Published online: 21 May 2026

RESEARCH ARTICLE

Artificial Intelligence and Applications
2026, Vol. 00(00) 1–22
DOI: 10.47852/bonviewAIA62026330

BON VIEW PUBLISHING

A Comparative Study of Hybrid Text Summarization Techniques Using Traditional, Static, and Contextual Word Embeddings

Lilakant Pokhrel[1], Sangita Pokhrel[1,*], Swathi Ganesan[1] and Nalinda Somasiri[1]

[1] *Department of Computer Science and Data Science, York St John University, UK*

Abstract: Currently, there is an exponential growth in textual data due to the rapid expansion of internet technology and various social and entertainment platforms. This is causing widespread negative effects on document and text management, text classification, and information retrieval, among other areas. To address the challenge of translating large volumes of text into natural-sounding language that retains citations, quotes, and references, summarization research is gaining significant international attention. The focus is on developing an effective hybrid (extractive and abstractive) summarization system by implementing different word embedding and summarization techniques. The study examines various embedding methods, from traditional approaches (Count Vector, TF-IDF) to static word embeddings (Word2Vec, GloVe, FastText), and contextualized embeddings (ELMo, GPT, BERT). It employs efficient extractive methods (TF-IDF, Textrank, LSA, Word2Vec, ELMo with K-means) and abstractive methods (Bi-LSTM, GPT) to create an effective summarization system. By integrating eight summarization techniques into Django-based (Python) web applications, the research finds that GPT performs particularly well, followed by TextRank and Elmo-based systems, with their ROUGE scores compared. The study also discusses the challenges faced in developing such a system and outlines future directions for ongoing research.

Keywords: text summarization, hybrid summarization, contextual embeddings, word embeddings, natural language processing

1. Introduction

In the digital age, vast amounts of text data, including user messages, blog posts, webpages, news, and content on social media platforms, have been rapidly accumulating. Besides these, there is extensive rich textual content in publications, books, novels, legal records, scientific papers, biomedical documents, and other archives. As the volume of information increases, a significant information explosion has occurred. Users must spend considerable time sorting through complex texts and removing redundant content, which greatly hampers their productivity. Finding quicker ways to extract necessary information from text resources, as well as summarize and compress it, has become an urgent and fundamental challenge. Manual summarization involves reading each article and then summarizing it, which is costly and risks losing valuable information. Text summarization has become one of the most challenging tasks in natural language processing (NLP) due to the complexity of the input text. When dealing with complex or nuanced language, traditional text summary techniques often fail to produce summaries that are both useful and fluid. The main problem with the research is the large volume of readily available textual content, which is a major obstacle for readers trying to swiftly understand the main points of lengthy articles, documents, or research papers, where traditional summarization techniques fail to achieve efficient results. One solution to this problem is to leverage deep neural network-based contextual embedding techniques, which can also capture the relations and context of the text, for deep learning-based text summarization. Deep learning-based text summarization offers an effective solution to this problem [1]. As explained by [2], understanding what text summarization is and what it can achieve is crucial. Their proposal focused on the automatic extraction of summaries from technical papers, news portals, magazine articles, and other sources. The team of researchers [3] utilized pre-trained encoders to develop a system capable of identifying important details within an original text, thereby generating a high-quality summary that captures the most crucial information taken out of the input document.

Deep learning-based text summarization (extractive and abstractive) with NLP enables the creation of short, accurate, and more natural reading summaries; however, it remains challenging, especially when the input text is very complex and lengthy.

2. Literature Review

The rapid expansion of digital text poses a substantial obstacle to those trying to quickly assimilate important information from documents, articles, or research papers. Although techniques for automatic text summarization mostly fall into the extractive and abstractive categories, text summarization has become an essential tool to address this issue. To create more reliable

***Corresponding author:** Sangita Pokhrel, Department of Computer Science and Data Science, York St John University, UK. Email: s.pokhrel@yorksj.ac.uk

and fluid summarizing systems, recent research has focused on traditional text summarization techniques, including hybrid approaches that integrate deep learning with NLP.

2.1. Word embedding techniques

The machine cannot directly interpret or understand texts or articles. Therefore, it requires NLP. The field of NLP provides a way to convert textual or word data into numerical embeddings or representations, commonly referred to as vectorization in standard terminology [4]. The literature review on word embedding techniques examines various prevailing methods, their disadvantages, and pros as illustrated in Figure 1.

2.1.1. Traditional techniques for embedding

The traditional approaches for word embedding include statistical-based techniques for word representation, such as count vector, TF-IDF, and co-occurrence matrix. These techniques are also categorized [4] as frequency-based word embeddings. [5] Count Vector, or Bag of Words, is a simple approach to word embedding in NLP that involves creating a vocabulary of all unique words from a given corpus or set of articles. It forms a matrix with dimensions $M \times N$, where "N" represents the vocabulary and "M" corresponds to the documents or articles. The values in the (i, j) position indicate the frequency of the word in the document. For example, to represent the following documents using the CountVectorizer, we need the following steps:

Doc 1: "I love word embedding techniques"

Doc 2: "Word embedding techniques are interesting"

As shown in Table 1, each column represents a unique vocabulary, and each row represents a document. Each (i, j) value indicates the frequency of a vocabulary in the ith document. Each document is transformed into a vector; for n documents, there are n vectors. In the above tabular representation of the text data, the vector representations are Doc 1: [1, 1, 1, 1, 1, 0, 0] (Vector 1) and Doc 2: [0, 0, 1, 1, 1, 1, 1] (Vector 2). For the document "d" and a vocabulary "V" of size n, the count vector representation of document "d" can be expressed as [$V1$, $V2$, $V3$,. . . , Vn], with each element calculated using the following Equation (1) given below:

Vi = count($word_i$) where it gives the number of times the i^{th} term in the vocabulary "V" appears in the document "d."

Next, TF-IDF is a statistical method that converts words into numerical vectors to create word embeddings, operating in two steps. First, it calculates the term frequency of a word in a specific document, and then it computes the inverse document frequency (IDF). Based on these two values, the importance of words within a document is assessed and represented as a vector. It is an improved version of the technique described in [6], known as "Bag of Words," for word embedding. The term frequency of the word "t" in a document "d'" is denoted as TF (t, d) and calculated using the following Equation (1) below [7]:

$$TF(t,d) = \frac{No.\ of\ times\ the\ term\ `t'\ appears\ in\ the\ document\ `d'}{Total\ number\ of\ Terms\ or\ Tokens\ in\ the\ document\ `d'} \quad (1)$$

The value of TF (t, d) highlights how often a term "t" appears in a document, indicating that the more a term occurs, the more significant it is for that document. In contrast, the IDF of the term "t" across the entire corpus is represented by IDF (t, D) and is calculated using the provided Equation (2) below [8]:

$$IDF(t,D) = \frac{Total\ No.\ of\ Documents}{No\ of\ documents\ the\ term\ \text{"t" appears}} \quad (2)$$

Figure 1
Word embedding techniques

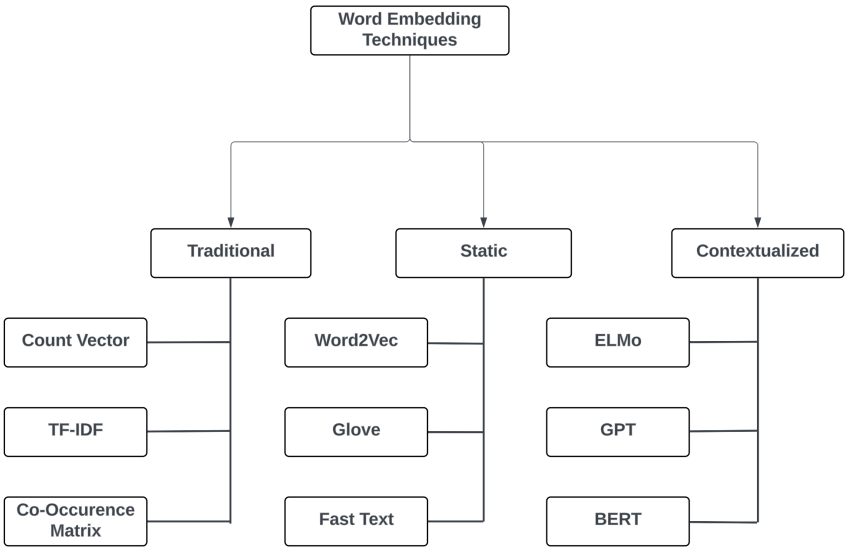


Table 1
Word embedding techniques: count vector

Vocab/docs	I	Love	Word	Embedding	Techniques	Are	Interesting
Doc 1	1	1	1	1	1	0	0
Doc 2	0	0	1	1	1	1	1

The log is used to normalize the values. The measure of IDF (*t, D*) determines how important a term "*t*" is across the entire corpus, and the logic is that the more times the term "*t*" appears in the documents, the less significant it is than a term that appears in fewer documents.

Finally, the product of two TF (*t, d*) and IDF (*t, D*), denoted and given below, determines how important a word is in a document. TF-IDF (*t, d, D*) = TF (*t, d*) × IDF (*t, D*), so the higher the value of TF-IDF for a particular term "*t*," the more important the word or term "*t*" is in the document.

The co-occurrence matrix is a traditional word embedding technique based on matrices that captures how often pairs of words co-occur within a specific corpus. The context is defined by another concept called a sliding window. The three key concepts involved in constructing the co-occurrence matrix are words, vocabulary, window size, and matrix structure. A word is an individual token or term, while vocabulary is the set of unique words in the corpus. The window size specifies the number of words surrounding a target word that are included in the context. For example, if the window size is 1, it includes the target word and the words before and after it. The shape of the co-occurrence matrix is a square of size $M \times N$ (where $M = N$), with both rows and columns representing the vocabulary. The value at the position defined by the *i*th row and *j*th column indicates the frequency with which word *i* and word j appear together within the context in the corpus. For example, consider the following two simple sentences:

Sentence 1: "I like the word embedding technique."

Sentence 2: "I like machine learning algorithms."

Analyzing the above two sentences and the construction vocabulary of unique words, the following vocabulary list has been created: Vocabulary ["I," "like," "word," "embedding," "technique," "machine," "learning," "algorithm"]. Given a window size of 1, a co-occurrence matrix of dimensions $M \times N$ is constructed in the following table representation. Table 2 illustrates the example.

As illustrated in Table 2, the word pairs ("I" and "like") appear twice, once in sentence 1 and once in sentence 2. The calculation of the co-occurrence matrix is quite straightforward; you just need to map each pair of words (one from the row and one from the column). If it has occurred, increment its count; if not, it remains zero, and an empty cell if the pair has already been considered in a previous row or column.

2.1.2. Static approaches for word embedding

Static word embedding techniques can capture semantic relationships between words based on their relevance across the entire corpus. This category includes many popular methods, such as Word2Vec, Glove, and FastText, as briefly introduced in the section given below:

Word2Vec is a variant of the static word embedding category, which is a neural network-based model that produces dense vector representations for individual tokens or words. The core idea behind Word2Vec is to train a neural network to identify the semantic or contextual words associated with a given target word and to generate a representation that captures the target word's semantic meaning [4]. Developed by Google in 2013, this technique enables mathematical operations between words. For example, when trained on a large corpus, the operation between the words "queen" and "man" produces interesting results, demonstrating its ability to capture semantic relationships. It utilizes two concepts—Continuous Bag of Words (CBOW) and Skip-Gram—to capture semantic connections between words.

Skip-Gram: This technique estimates the context (surrounding words) of a given target word. Its aim is to optimize the probability of context words given a target word within the corpus. The mathematical representation of Skip-Gram for a specific target word *i* involves predicting the context words Wt-*j*, Wt-1, . . . , Wt+1, . . . , Wt+*j*, where '*j*' denotes the size of the context window. Suppose we have a sequence of words *w*1, *w*2, . . . , *wt* for training, so the goal of the Skip-Gram is to optimize the log probability, expressed by its Equation (3) [9]:

$$\text{Skip-Gramp} = \frac{1}{T}\sum_{t=1-j\le k\le j, k\neq 0}^{T} \sum \log \mathrm{P}(w_{t+k}|w_t) \quad (3)$$

where w_t the target word w_{t+k} is the context word of window size j and T is the total number of the corpus. The conditional probability of $\mathrm{P}(w_{t+k}|w_t)$ is calculated using the SoftMax function, and this function is defined by the following Equation (4) given below:

$$\mathrm{P}(w_o|w_I) = \frac{exp(V'_{w0} \times V_{w1})}{\sum_{w=1}^{W} exp(V'_w \times V_{w1})} \quad (4)$$

where w_o is the output context, w_I is the input or target word, w_{wI} is the vector presentation of the input w_I, V'_{wo} is the vector representation of the output word w_o, and W is the size of the vocabulary.

Continuous Bag of Words (CBOW): The main objective of this concept is to predict the word that is targeted for a given context. It optimizes to predict a single target word from multiple context words, whereas Skip-Gram predicts multiple context words from a single target word. Assume for a sequence of training words $w_1, w_2 \ldots, w_T$, the prime goal of this technique is to optimize the log-based probability, which is given by the following Equaion (5) [10]:

$$\text{CBOW} = \frac{1}{T}\sum_{t=1}^{T} logP\left(w_t|w_{t-j,}\ldots\ldots.., w_{t-1}, w_{t+1,}\ldots\ldots w_{t+j}\right) \quad (5)$$

Table 2
Word embedding techniques: co-occurrence matrix

	I	Algorithms	Embedding	Learning	Like	Machine	Technique	Word
I	0		1	1				
Algorithm	1	0	0	1		1	0	0
Embedding	1	0	0	0		0	0	
Learning	1	1	0	0			0	0
Like	2	1	1	1	0			
Machine	1	1	0	1	1	0	0	0
Technique	1	0	1	0	1	0	0	
Word	1	0	1	0	1	0	0	0

where the conditional probability $P(w_t|w_{t-j}, \ldots\ldots, w_{t-1}, w_{t+1}, \ldots\ldots\ w_{t+j})$ is calculated using the following SoftMax Equation (6) [10]:

$$\mathrm{P}(w_I|w_C) = \frac{exp(V'_{wI} \times V_{wC})}{\sum_{w=1}^{W} exp(V'_w \times V_{wC})} \tag{6}$$

Figure 2 [11] illustrates the working mechanism of Word2vec.

Global Vector for Word Representation: It is an unsupervised method for obtaining the dense vector embedding for the words, whose prime focus is to catch both global word co-occurrence statistics and local context windows based on the statistical information, such as Word2Vec, to innovate high-quality vector representation of the words from the corpus [12]. This technique was developed by a Stanford University researcher in 2014. The core concepts behind this technique are global context, co-occurrence matrix, and probability ratios. This technique collects global statistical information about word co-occurrences across an entire corpus, meaning it considers the frequency with which words co-occur in the dataset rather than just the local context window [13]. After the global context [14], it constructs a large co-occurrence matrix, where each element denotes the number of times a word occurs in the context of another word across the entire dataset. For example, if the two words "cat" and "dog" appear in a similar context, such as the action "sat" or "chased," then their word vectors will be similar. So, the product of v_cat and v_dog, and v_{dog} would calculate the logarithm of their co-occurrence count. The principal aim of the GloVe model is to learn word vectors so that the dot product of two-word vectors, suppose for words *i and j*, calculates the logarithmic values of their co-occurrence count X_{ij} and is given by the following Equation (7) [12]:

$$vi, vj + bi + bj = log(X_{ij}) \tag{7}$$

where *vi and vj* vectors for words *i and j bi and bj* are the bias terms for words *i and j* and X_{ij} is the co-occurrence count for the words *i and j*. Another useful function called the cost function, denoted "J," whose role is to minimize the cost and calculates the difference between the predicted and true logarithms of co-occurrence counts and is given by the following Equation (8) [15]:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(V_i.V_j + b_i + b_j - \log(X_{ij})\right)^2 \tag{8}$$

where "V" is the size of the vocabulary, $f(X_{ij})$ is a weighting function that lessens the effect of frequent word pairs and domination from learning rare co-occurrence words, and it is defined as Equation (9) [15] given below:

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{X_{max}}\right)^{\alpha} & if\ (X_{ij}) < (X_{max}) \\ 1\ if\ (X_{ij}) \geq (X_{max}) \end{cases} \tag{9}$$

where X_{max} is a tunable parameter that acts as the threshold for how influential a co-occurrence pair word can have and α is a parameter that monitors the scaling of the weights, and its value is usually 0.75. Figure 3 [16] illustrates the working mechanism of the GloVe.

FastText Embedding: A powerful word embedding technique developed by the AI research team at Facebook's FAIR lab in 2016 is FastText. It extends the basic Word2Vec concept and addresses some of its limitations, particularly in handling out-of-vocabulary (OOV) and rare words. The key concept [17] in FastText is the use of sub-word information, which ultimately enables the construction of word embeddings sensitive to the internal structure of words. It breaks individual words into smaller parts, called sub-words or n-grams. Suppose we have the word "embedding," this word can be split down into various sub-words like "emb," "bed," "edd," "din," "ing," and so on. This interesting approach enables FastText to construct more semantically meaningful word embeddings for words that are not even in the training vocabulary, thereby solving the OOV problem. To split words into sub-words, it uses the n-gram technique, applying character n-grams of lengths 3–6. For example, suppose we have the word "apple"; by applying an n-gram of length 3, this word can be divided into "app," "ppl," and "ple." Here, the representation of each word is the sum of its n-grams, and the final word vector is the average of sub-word vectors. By considering sub-word information, FastText is better suited to handle morphologically rich languages, where words often include prefixes and suffixes. Suppose we have the word "*w*," then its vector representation v_w is calculated by the sum of the vectors of its n-grams, and the Equation (10) [18] is given by:

$$v_w = \Sigma_{g \varepsilon G_w} v_g \tag{10}$$

where G_w is the set of n-grams for the word "w" and v_g is the vector representation of the n-gram g.

For FastText, the Skip-Gram objective Equations (10) and (11) [9] are given by:

$$\text{Skip-Gramp} = \frac{1}{T} \sum_{t=1-j \leq k \leq j, k \neq 0}^{T} \sum \log \mathrm{P}(w_{t+k}|w_t) \tag{11}$$

Figure 2
Word embedding technique architecture: Word2Vec

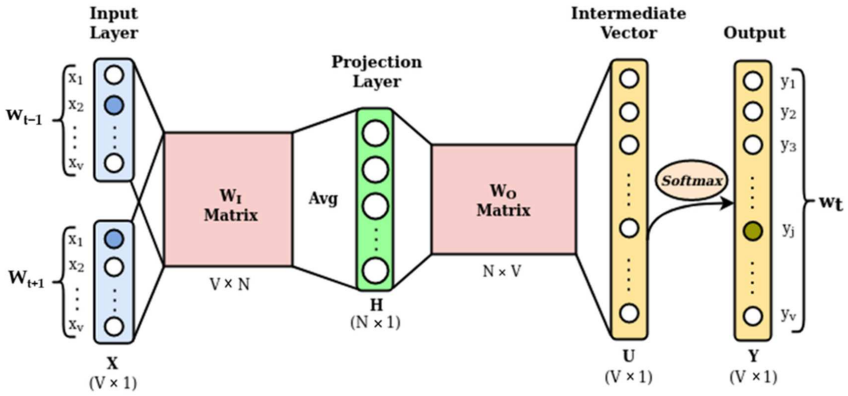
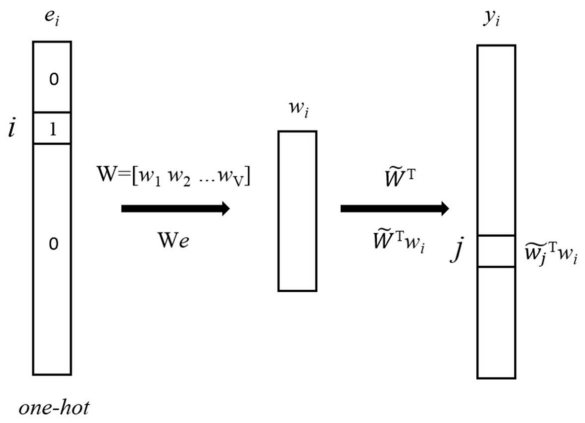


Figure 3
Embedding technique architecture: GloVe



where w_t is represented as the sum of its n-gram vectors instead of single vectors. Figure 4 [19] illustrates the working mechanism of FastText.

2.1.3. Contextualized approaches for embedding

These word embedding categories are very advanced, powerful, and based on neural network architectures. These embedding techniques can capture the complex characteristics of word use, such as syntax and semantics [20]. They can produce context-sensitive word representations that allow them to perform mathematical operations on words. Examples of this class of word embeddings include ELMO, GPT, and BERT, as briefly introduced in the following section.

1) ELMO

It stands for embedding from a language model, a deep neural network-based, conceptualized word embedding technique that is powerful at capturing and generating the context-sensitive relationships among words. It can model the complex characteristics of the word, such as syntax and semantics, and capture morphological information, including prefixes, suffixes, and roots [20]. It is a dynamic word representation technique that generates scenario-based word representations using a deep neural network architecture called a bidirectional long short-term memory (Bi-LSTM). The main concept behind the ELMo embedding can be explained as follows:

Contextual Word Embedding: The embeddings are dynamic; the embedding for each token or word is not fixed but context sensitive. For example, if we have the word "bank," it will have different embeddings. The word "bank" can be used in the context of rivers and financial institutions, such as "riverbank" vs "financial bank."

Character-Level Representation: ELMo uses a character-level representation of words based on a convolutional neural network (CNN). Each word is split into its characters to represent in the CNN, making it possible to capture the morphological information such as suffixes, prefixes, and roots. Suppose C_w is the sequence of characters representing the single word "w," then the character-level representation of this word can be denoted as x_w, and it is computed using the CNN, followed by a highway as Equation (12) [21] given below:

$$x_w = \text{Highway}\,(CNN(C_w)) \tag{12}$$

where $CNN(C_w)$ CNN layers apply convolutional filters to the sequences and Highway denotes the neural network that can fine-tune the output to generate the word-level embedding x_w.

Neural Network-Based Representation: ELMo embedding techniques use a directional long short-term memory (LSTM), which processes the entire sentence in both directions (left-to-right and right-to-left), enabling it to capture the context of each word from both directions. Suppose we have a sequence of words $(w_1, w_2, \ldots, w_T)$. In a sentence, the respective word embedding $(x_{w1}, x_{w2}, \ldots, x_{wT})$ from the character-level CNN architecture is inputted into a deep directional LSTM, and the hidden state h_t^L for each word and each layer "L" is computed as Equation (13) given below:

$$h_t^L = \left[\underset{h_t^L}{\rightarrow}; \underset{h_t^L}{\leftarrow}\right] \tag{13}$$

where $\underset{h_t^L}{\rightarrow}$ is the hidden state from LSTM while processing the sentence left-to-right, $\underset{h_t^L}{\leftarrow}$ is the hidden state processing the sentence from right to left, and [.;.] is the concatenation of two vectors. The ultimate ELMo embedding for a word w_t is calculated as the weighted sum of the hidden states of the Bi-LSTM and is given by the following Equation (14) [21]:

$$ELMo(w_t) = \gamma \sum_{L=0}^{L} s_L h_t^L \tag{14}$$

where γ is a scaling factor, s_L is SoftMax-normalized weights, and h_t^L is the hidden state of the Lth layer for the word w_t. Figure 5 [22] illustrates the working mechanism of the ELMo.

2) GPT

It stands for Generative Pre-Trained Transformer and is a transformer-based language model consisting of self-attentive encoders and decoders. The encoder generates context-sensitive embeddings using a unidirectional (e.g., left-to-right) text-processing algorithm. Another component, that is, the decoder part of the transformer-based architecture, enables GPT to process its input text using multiple layers of self-attention mechanisms. This allows GPT to capture long-range dependencies between words in a sentence. The transformer-based architecture with a self-attention mechanism, pre-training on a large corpus, fine-tuning on a specific dataset, and positional encoding features make the GPT-based embedding much more powerful at capturing the complex semantic and syntactic relationships among words in a sentence from Yenduri et al.'s work [23]. Suppose w_1, w_2,…, w_n are the sequences of the words for a given sentence. Here, each word w_i is first mapped to a vector x_i of fixed length using the embedding matrix W_e as defined by the following Equation (15) [20]:

$$x_i = W_e\,[w_i] \tag{15}$$

where W_e is the embedding matrix and x_i is the embedding vector of fixed length for the given word w_i. The beauty of GPT is its use of positional encoding. This concept makes GPT capture the positional information of each word in the given sentences. If p_i a positional encoding vector is added to the word embedding, the calculation of the positional encoding is based on a sinusoidal function and described by the mathematical Equation (16) given below:

$$z_i = x_i + p_i \tag{16}$$

Figure 4
Word embedding technique architecture: FastText

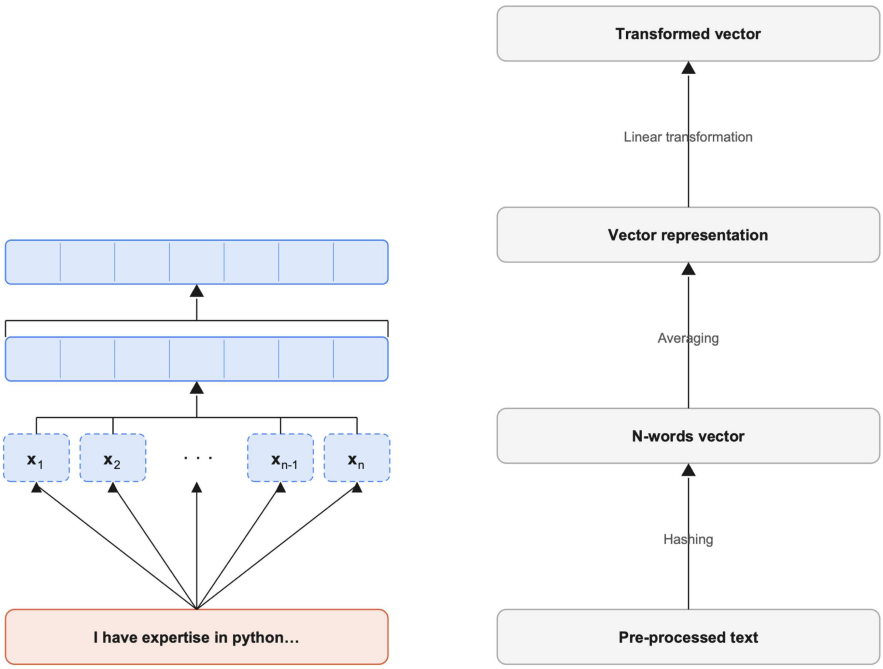
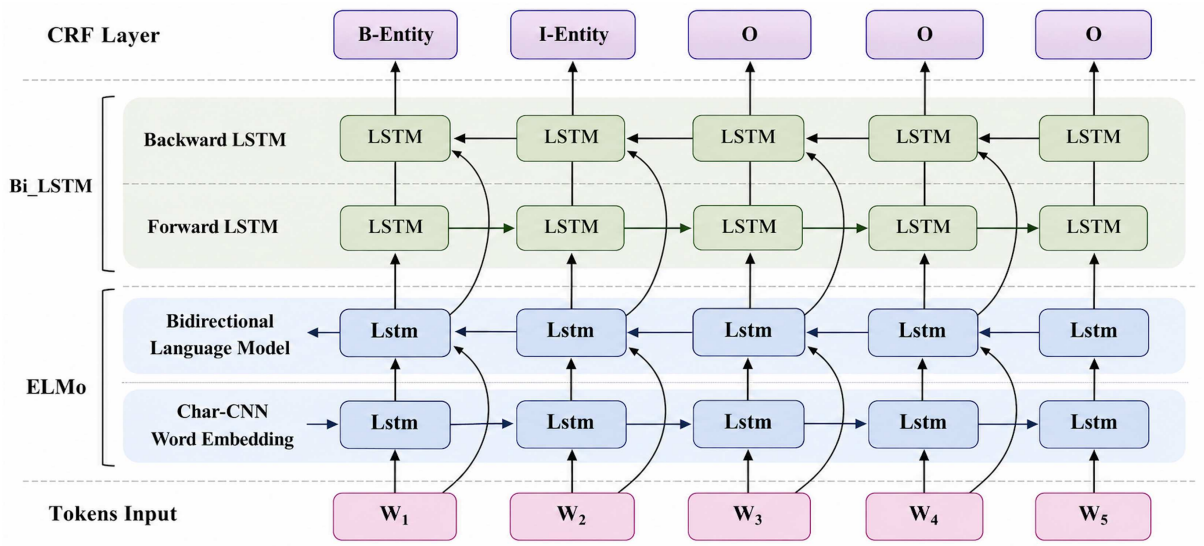


Figure 5
Embedding technique architecture: ELMo



where p_i is the positional encoding vector for position i. Another key aspect of GPT-based word embedding is the self-attention mechanism. In transformer-based embedding, each embedding for a word denoted by z_iIt communicates with all other words in the word sequence by the self-attention mechanism technique. This mechanism calculates the attention for each of the words, and it is defined by the mathematical Equation (17) [20] given below:

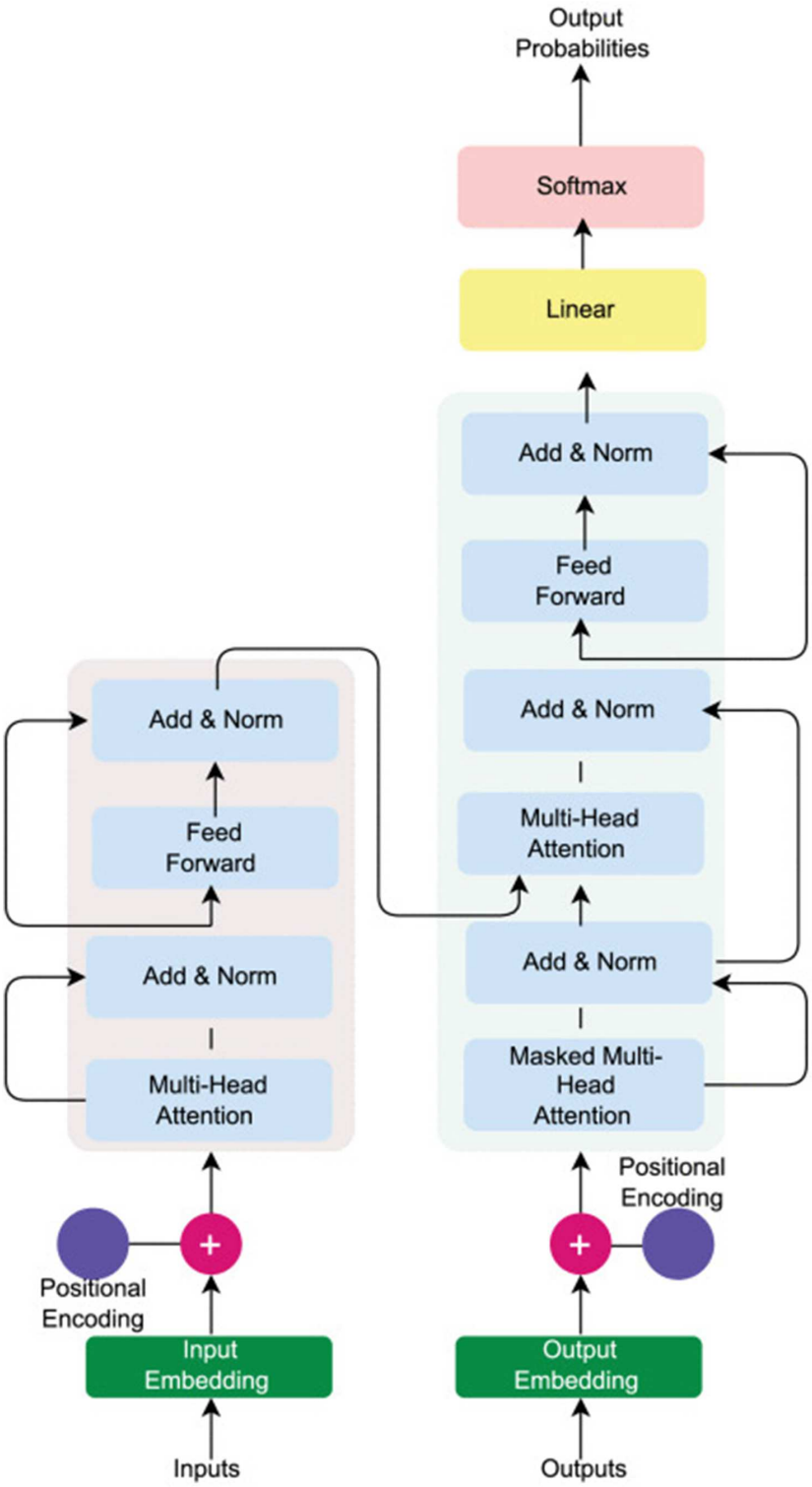
$$Attention(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \times V \qquad (17)$$

where Q defines the query, K defines the keys, V defines the values derived from z_i, d_k the dimension of the keys, and the SoftMax function makes sure that attention scores sum to 1. The GPT employs multiple attention layers and feed-forward networks to further filter the representation. Figure 6 [23] illustrates the working mechanism of the GPT.

3) BERT

The Bidirectional Encoder Representation from Transformers (BERT) is another transformer-based language model that produces context-aware embeddings by considering the entire sentence bidirectionally (left-to-right and right-to-left). BERT generates several embeddings for each of the words based on the meaning it carries in the sentence. The encoder is the primary component of the BERT-based text architecture and comprises several transformer encoder layers. Each encoder layer is defined

Figure 6
Word embedding techniques architecture: GPT



by a bidirectional multi-head self-attention mechanism. The attention layer can process the input sequence in both directions, with a nonlinear transformation applied to the output, and the encoder includes a position-wise feed-forward network. BERT is pre-trained on large unlabeled text data using techniques called Masked Language Modeling (MLM) and Next Sentence Prediction (NSP) [24]. In MLM, some tokens are randomly selected and replaced with a mask token, and the model is trained to estimate the masked tokens based on the context. In NSP, the algorithm is trained to estimate the likelihood of two consecutive sentences in the original text. All the tasks described till now are done in the model pre-training; later, BERT can be fine-tuned with specific datasets in NLP, such as text summarization, text classification, sentiment analysis, question answering, and so on.

BERT follows a special input format to process, which includes [CLS] token, [SEP] token, and Word Piece tokenization. The first input sequence used to classify tokens is [CLS]; the token used to separate sentences is [SEP]; and word-piece tokenization splits words into sub-words or characters. Suppose w_1, w_2,…, w_n are the sequences of the tokens given to BERT input and defined by the following Equation (18) [25]:

$$\text{Input } [CLS, w_1, w_2, \ldots, w_n, SEP] \tag{18}$$

The BERT also uses positional encoding to gather information about the position of each token and is defined by:

$z_i = x_i + p_i$, where x_i is the embedding vector for the word w_i and p_i is the positional encoding vector for position i. As a certain percentage of the input tokens is randomly masked, the model is trained to predict original tokens based on the surrounding tokens and is computed using the mathematical Equation (19) [25] given below:

$$\mathrm{P}(w_i | w_1, \ldots, w_{i-1}, [MASK], w_{i+1}, \ldots, w_n) \tag{19}$$

It makes BERT understand the context on both sides of a masked word. Another concept BERT uses is the self-attention mechanism, which computes attention scores for each word with respect to every other word in the sequence. It makes the model assign different words varying importance when generating word embeddings, using the following Equation (20) [26]:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \times V \tag{20}$$

where Q defines the query, K defines the keys, V defines the values derived from z_i, d_k is the dimension of the keys, and the SoftMax function makes sure that attention scores sum to 1. Figure 7 [27] illustrates the working mechanism of the BERT.

2.2. Text summarization techniques

This section delves into previous studies or research on techniques or models used in the text summarization domain. The accuracy or contextuality of the summarization depends heavily on the word embedding technique used; based on these techniques, summarization is categorized as extractive or abstractive. Figure 8 illustrates the text summarization techniques.

Figure 7
Word embedding techniques: BERT

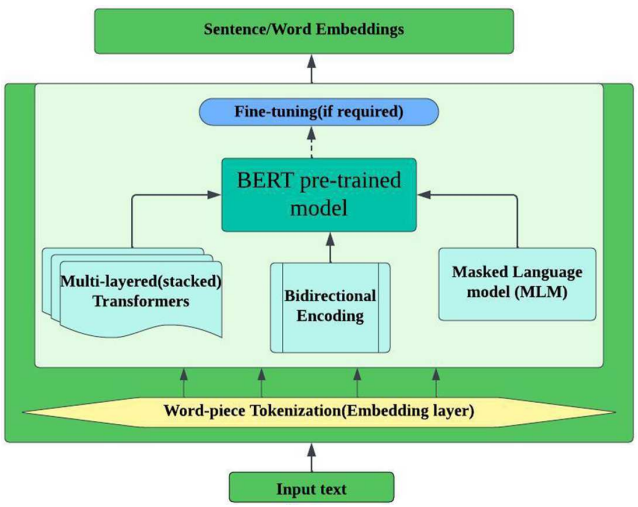
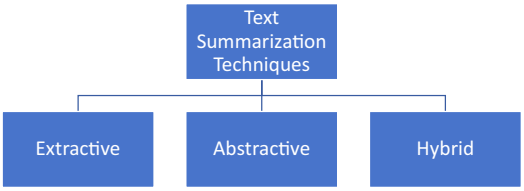


Figure 8
Text summarization techniques



As discussed in section 2.2 of the report, traditional word embedding techniques (Count Vector, TF-IDF, and co-occurrence) are less able to capture the semantic and syntactic relationships of the words, whereas static word embedding (Word2Vec, GloVe, FastText) techniques are a bit more able to hold contextual information as compared to traditional embedding. The contextualized approach (ELMo, GPT, BERT) is superior to generating both types of summarizations, that is, extractive and abstractive.

2.2.1. Extractive summarization techniques

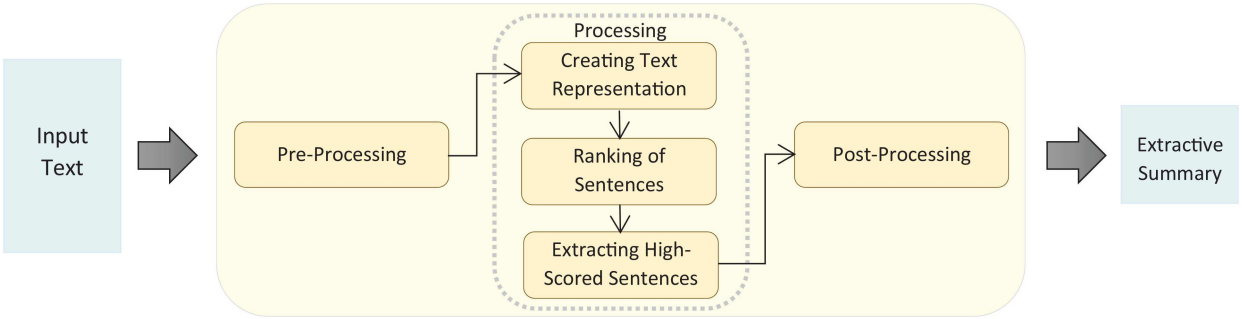
Extractive techniques function similarly to a text highlighter. After analyzing the original text, they identify the most important sentences by considering factors such as keyword frequency, sentence similarity, and overall document position. The summary is then created by explicitly extracting and combining these important sentences, ensuring factual correctness but potentially reducing readability in some cases [28]. Extractive text summarization is executed in three steps. In the first one, a representation of the document or the whole corpus is generated. Depending on the number of tokens produced from the input document, they would be separated into sentences or paragraphs. As a preprocessing technique, stop-word removal is sometimes performed. In the next step, a measure of the importance of the sentences based on a scoring function is attempted to determine whether the sentences convey helpful information for summarizing the paper generally or how much they provide insight into the relationship between certain themes. We would expect the "importance" of a sentence to the "understandability" of the text to be reflected in the score. The last step of the summarization process would consist of generating the summary by combining the scores achieved in the previous steps [29]. Figure 9 [30] illustrates the architecture of the extractive summarization technique.

This category of text summarization uses traditional way of word embedding techniques such as Count Vector, TF-IDF, and co-occurrence matrix as explained in the section below:

1) TextRank

Zaware et al. [31] used the TextRank approach to summarize the text by treating phrases from the original text or scraping from the webpages, drawing inspiration from Google's PageRank algorithm. Like website linkages, sentences are connected based on their similarity, and TextRank assigns a score to each sentence indicating its importance in the network. Sentences that exhibit a high degree of coherence and relevance concerning other significant sentences that share comparable content are deemed to be the key theme of the summarized text [32].

Figure 9
Architecture of the extractive summarization technique



2) LexRank

LexRank is a graph-based summarizing method that emphasizes textual centrality, as explained by the researchers [33]. It treats sentences as nodes connected by similarity-based edges. These relationships are analyzed iteratively by LexRank. LexRank gives precedence to a sentence's relationship to previously identified central sentences, whereas TextRank considers overall connectivity. As a result, significant sentences reinforce one another's significance, creating a snowball effect. Sentences that are judged to be quite like other key sentences end up forming the main body of the summary [34].

3) TF-IDF/LSA

The significance of words within a document and throughout a collection of documents is examined using statistical and frequency techniques. Term frequency-inverse document frequency, or TF-IDF, considers a term's frequency of occurrences both inside a document (TF) and throughout the complete document collection (IDF). TF-IDF uses [35] the static word embedding techniques. This makes it easier to find terms that are particular to and pertinent to the current content. Latent semantic analysis (LSA) [33], on the other hand, is not limited to single words. By locating hidden topics, it investigates the underlying thematic structure of a collection of documents. Because of this, LSA can identify semantic connections between words even when they aren't explicitly mentioned together. For tasks such as information retrieval, document classification, and text summarization, both TF-IDF and LSA are useful tools [36].

4) Extractive summarization using pre-trained embedding

This category of text summarization uses pre-trained word embedding techniques such as GloVE, Word2Vec, FastText, and so on, as explained in the section below:

In the field of NLP, as suggested by Karim et al.'s work [37], the widely used word embedding technique is used to generate the extractive summary from the original text. This embedding technique can be used with K-means to generate an efficient extractive summary, where each cluster defines a set of data points with common characteristics. Word2Vec first encodes sentences into a vector space, learning the semantic content of each sentence. K-means clustering is used to map sentence vectors to clusters in the vector space so that similar sentences are assigned to the same cluster. A summary is produced by selecting sentences from the centroid for each cluster [38]. This approach has the advantage that the sentences contained in the summarization will likely remain cohesive and carry more semantic information. Similarly, as explained by Nguyen and Do's work [39], another powerful approach to increase the cohesion of the summarized text is to use FastText embeddings to capture the meaning of words and sub-word counts, thus further increasing the representation of rare words and word variations, and then use K-means clustering (a type of data-driven or unsupervised learning) to determine the clusters into which these vectors should be grouped [40]. This way, the most representative sentences in each cluster are those that both capture the most important ideas and help maintain text structure, continuity, and flow.

2.2.2. Abstractive summarization techniques

The goal of abstractive summarization is to understand the text's underlying meaning as a human reader would. It uses sophisticated deep learning models, frequently built using strong transformers or sequence-to-sequence architectures. These models can generate entirely original phrases and sentences that capture the main concepts of the original text, resulting in summaries that are often more succinct and fluid than those produced by extractive techniques [1]. Figure 10 illustrates the abstractive technique [41].

As illustrated in Figure 10, extractive summaries can also be used to generate abstractive summaries using an encoder–decoder architecture. This section of the literature survey presents some of the highly efficient deep neural network methods used to generate contextual abstractive summaries of texts. Examples of such abstractive summarization techniques include Bi-LSTM, transformer-based architecture (BERT, GPT), and so on, as explained below.

1) Bi-LSTM

Bi-LSTM is the most used abstractive algorithm to generate context-sensitive text summarization from large texts; it plays an important role in the process of abstractive text summarization. Coherent, more context-accurate summaries are possible because the Bi-LSTM enables modeling of context from both directions, that is, preceding and succeeding words in a sequence. When considering an encoder–decoder architecture for abstractive text summarization, a Bi-LSTM is found to be best. The encoder part of Bidirectional Long Short-Term Memory (Bi-LST) can learn context from both directions of the input, that is, right-to-left (forward) and left-to-right (backward), generating rich context vectors [42]. More recently, [43] Preethi et al.'s work showed that incorporating an attention mechanism into an encoder–decoder model with a Bi-LSTM helps the model focus on relevant parts of the input, building more fluent, context-aware summaries by improving context-awareness and paraphrasing of the input text.

Figure 10
Architecture of abstractive summarization

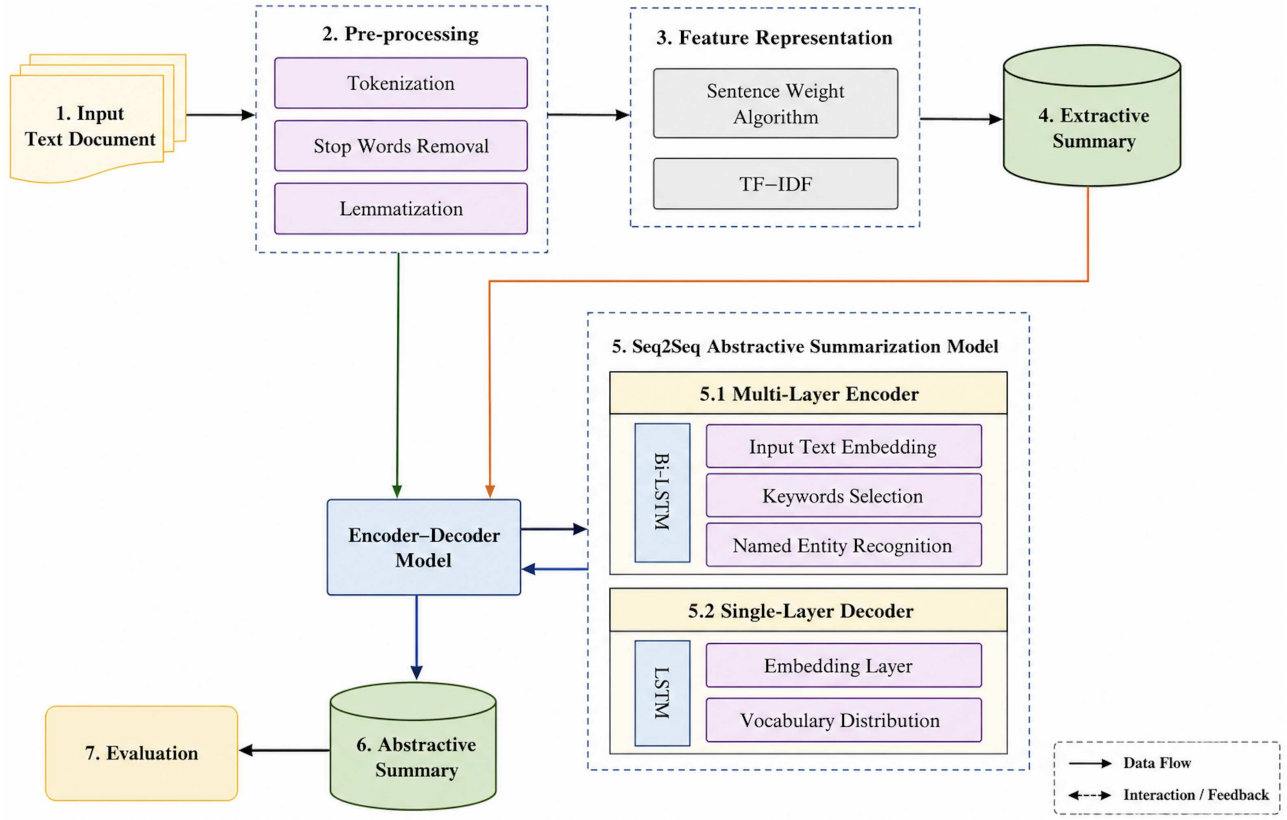
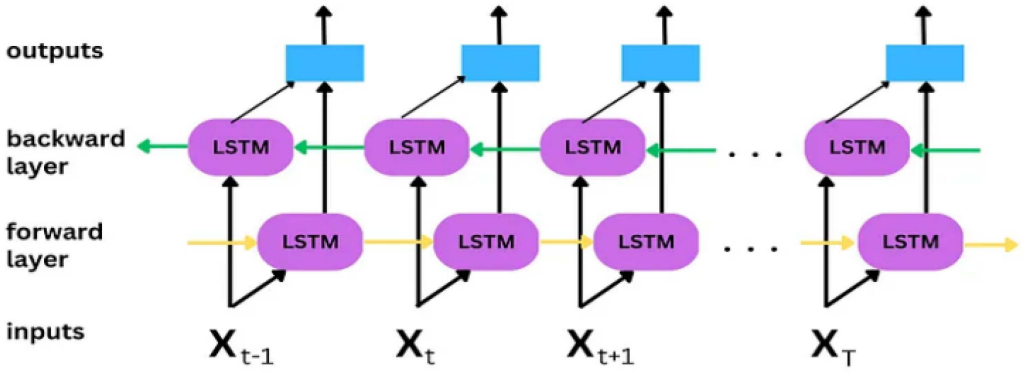


Figure 11
Architecture Bi-LSTM



Bi-LSTM consists of two main components as described in the section below:

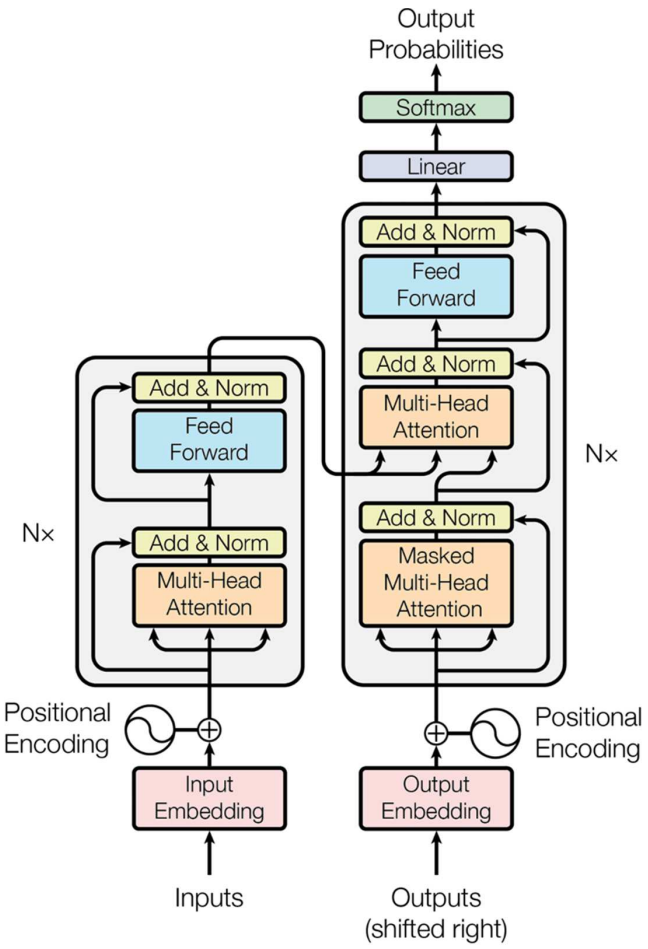
Encoder: This Recurrent Neural Network (RNN) scans the input text word by word and records the word relationships and sequence. It is frequently an LSTM variation. It distills this data into a condition of concealment that embodies the book's main ideas [44].

Decoder: The encoder's secret state serves as the foundation for this RNN. The summary is then created word by word, with each word generated based on the encoded context and the words that came before it. Until the summary reaches a predetermined length or an "end of summary" signal is generated, this iterative procedure is continued [45]. Figure 11 [27] illustrates the working mechanism of the LSTM.

2) BERT

The bidirectional transformer architecture of BERT has made it a popular model for extractive and abstractive text summarization, as it helps to capture possible complex sentential interactions between words. Kades et al.'s work [46] demonstrated summarization in clinical text using BERT to summarize text can be done via fine-tuning it for extractive summarization, that is, ranking source-text sentences by salience and generating them as the summary, or for abstractive summarization, that is, machine learning paraphrased human-sounding paraphrases and retaining as much of the source text while keeping citations and quotes intact. For instance, a fine-tuned BERT model can rank sentences within a document by importance,

Figure 12
Architecture: BERT



achieving state-of-the-art results on various extractive summarization benchmark datasets [47]. The BERT-based text summarizer employs BERT word embedding techniques, and the mathematical equation aligns with the BERT architecture illustrated in Figure 12, as described in section 2.2 of the report [47].

3) GPT

One of the biggest breakthroughs in text models has been the development of well-written text summarization methods that bypass earlier extractive methods [23]. While extractive methods involve selecting parts of the text and gluing them back together into a new, shorter version, developed GPT models can use their powerful language-generation abilities to create new, concise summaries that shorten the original passage while still preserving citations and quotes.

Traditional text-model summarizations tend to be less effective; See et al.'s work [48] demonstrated that GPT models can paraphrase and rephrase the text in a natural-sounding way. The transformative idea behind this approach (apart from using powerful language models) is that these systems have been trained on enormous amounts of data, making them highly robust to noise, noise reduction, quotation, and construction issues that datasets with smaller volumes might not handle as well. It has been utilized for a range of applications, including summarizing news items and longer research papers. Recently, GPT systems have been trained to cover an even broader scope of topics.

Most recently, a system called GPT-3 has been developed that can take a piece of text and generate human-like summaries, demonstrating a remarkable proficiency that seems quite promising [49].

The GPT-based text summarizer uses GPT word embedding techniques, and the mathematical equation follows the GPT architecture shown in Figure 13, as described in section 2.2 of the report [23].

2.2.3. Hybrid text summarization techniques

Hybrid techniques carefully combine extractive and abstractive features to get the best of both approaches. One method is a pipeline model where key phrases are first identified and then processed by an abstractive model, which enhances and refines them to produce a more natural-sounding summary. Another approach is to jointly train models to generate new text and select important information, balancing accuracy and coherence fluency [50].

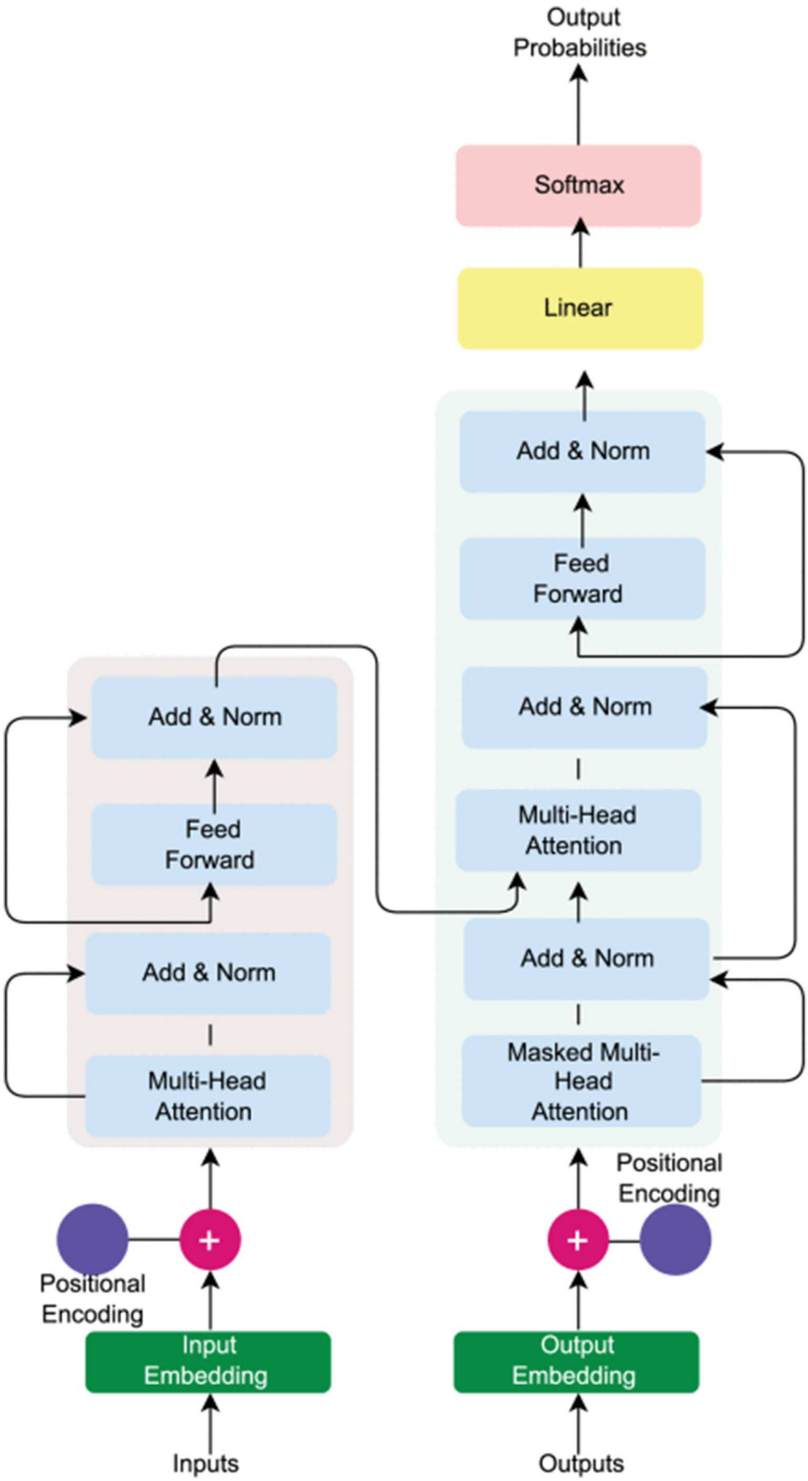
2.2.4. Evaluation metrics

It is an important task because, among other reasons, it allows us to assess the quality and type of summarization being produced. What approach should we take? There are various metrics in use, which are gradually being replaced and enhanced by newer ones: ROUGE (Recall-Oriented Understudy for Gisting Evaluation) evaluations are widely adopted, including ROUGE-N (where N represents the number of characters in the n-grams used to measure overlap between the generated summary and a set of reference summaries) and ROUGE-L (which measures the length of the longest common subsequence between the generated and reference summaries). BLE evaluation, an acronym for Bilingual Evaluation Understudy, is a translation metric designed for machine translation that assesses precision at the n-gram level by comparing the generated text to one or more reference summaries. METEOR (Metric for Evaluation of Translation with Explicit Ordering) also combines precision and recall, considering the importance of synonyms and stemming, and aims to address the limitations of existing metrics caused by out-of-order words [51]. This list could continue. However, all these metrics can be easily manipulated by the summarizer, which is why human evaluation is usually included in assessments to determine whether the generated summary is coherent, fluent, and sufficiently informative. This need for potential human intervention also highlights how challenging it is to develop fully automatic evaluation tools for summarization.

1) ROUGE (Recall-Oriented Understudy for Gisting Evaluation)

ROUGE, short for Recall-Oriented Understudy for Gisting Evaluation, is a suite of metrics for summarization evaluation based on the textual overlap between the target (output) and reference summaries. The fundamental premise of ROUGE is to use recall-oriented gisting similarity measures to capture the inclusion of citations, quotes, quotes with citations, or other important concepts from reference summaries into generated summaries. Variants of ROUGE include ROUGE-N (ROUGE-n means n-grams: so, ROUGE-1 means unigrams, ROUGE-2 means unigrams and bigrams, etc.), ROUGE-L (measuring the longest common subsequence), and ROUGE-S (measuring the skip-bigrams). ROUGE provides a quantitative measure of the fidelity of generated summaries to reference summaries, greatly facilitating our evaluation of summarization systems. Recent studies, such as the one by Gao et al.'s work [49], show that

Figure 13
General architecture of GPT



ROUGE remains relevant for evaluating summarization and provides avenues for a better understanding of the desiderata of a good summary [52].

2) BLEU (Bilingual Evaluation Understudy)

BLEU (Bilingual Evaluation Understudy) is an accuracy-based metric originally designed for machine translation, but it is also widely used for evaluating summarization tasks. BLEU relies on n-grams: scores are calculated by dividing the number of n-grams in a generated text by those in one or more reference texts and then comparing this to the n-grams in the candidate summary. More specifically, BLEU measures how many n-grams from the candidate summary appear in the reference summary, and it is often expressed as a probability score, for example, BLEU-3 = 0.9007, BLEU-4 = 0.9014, and BLEU-n = 0.9014. Although precision is central, as noted earlier,

it may not adequately reflect recall (which relates to the amount of information in the generated text, often minimal according to the metric), so alternative approaches and metrics have been proposed recently. In his 2018 study on BLEU over the years, Post discusses several advantages and potential drawbacks of BLEU and suggests various modifications to improve its suitability for summarization and other NLP tasks. This demonstrates that BLEU remains a dynamic field of research and continues to find diverse applications rather than being replaced by newer, more sophisticated metrics; nonetheless, this is an important point to consider [52]

3) METEOR (Metric for Evaluation of Translation with Explicit Ordering) is an evaluation metric designed to address additional limitations of BLEU, such as its focus solely on n-gram precision. Unlike BLEU, which measures only the number of exact matching n-grams between a reference and a candidate text, METEOR combines four precision and recall measures, including exact matches, stemmed matches, synonym matches, and phrase order. METEOR provides a more balanced and nuanced assessment of the quality of a generated summary than BLEU, making it better suited to situations that require sensitivity to subtle differences in summarization and translation, demonstrating that it remains a relevant evaluation measure in NLP today [53].

2.3. Theoretical framework

The various text summarization and embedding methods explained in the above section of the literature review have both advantages and limitations. The advanced word embedding techniques can capture context-sensitive information; however, they are computationally expensive to fine-tune the custom data, whereas traditional or static word embedding techniques are computationally more efficient, but their incapability to capture the contextualized embeddings makes them less useful in real-world applications for text summarization [54]. They have substantially advanced the development of the techniques for text summarization a hundredfold [55]. Finally, the literature review showed ways to measure the accuracy of the summarization system, identified and analyzed the challenges of adopting existing state-of-the-art summarization technology, and presented justification for adopting text summarization models. The research is grounded mainly in the theoretical foundation of NLP, focusing on text summarization techniques and summarizations. The literature survey showcases the comparative strengths and limitations of both static and contextual embedding. Contextual embeddings, such as ELMo [26], BERT, and GPT [21], can capture semantic relations by preserving the contextual meaning of words, though they are computationally expensive. In other words, traditional embedding techniques like Word2Vec and GloVe are more efficient but struggle to capture dynamic meaning in variable contexts.

3. Research Methodology

The research employs scientific methods and techniques to study the problem statement or the chosen field of study, materials, and sources, with the primary goal of establishing facts or drawing conclusions about the problem. The process begins with selecting the title or topic, which is carried out in several stages using established techniques and paradigms. Quantitative research involves measuring or assessing the numerical accuracy of various text models, also known as extractive features, whereas qualitative research evaluates accuracy based on the level of abstraction or the quality of the text summarizer models.

3.1. Research design

Figure 14 illustrates the architecture of the text summarization system, implemented with a Django application. It begins with the frontend, where the frontend component offers a way to interact with the summarization system. It includes file upload, URL, and custom text-area fields through which the user can enter the necessary information for summarization. The API call handler manages the HTTP request and response. The system then preprocesses the data, including scraping (if the summary is generated from a website URL), cleaning, and tokenization. The preprocessed data is passed to a statistical model for extractive summarization, and the output from this model is fed into a selected neural network-based abstractive summarization model. The results from the models are then displayed back to the user via a frontend interface using AJAX and jQuery for handling requests and responses. Similarly, Figure 15 illustrates the outlook of the project implemented.

3.2. Methods of data collection

Data is the most critical component of the applied research project, and without its availability, conducting such a project becomes impossible. The data used in this research is the CNN/Daily Mail dataset [55], collected from publicly accessible sources. Various datasets, such as XSum, SAMSum, and PubMed/arXiv, are compared with CNN/Daily Mail based on quality, relevance, and researcher popularity. Ultimately, the CNN/Daily Mail dataset is selected for the study. This large-scale text collection is commonly utilized for training and testing text summarization models. It comprises news articles and their corresponding human-written summaries. Collected and presented by researchers from Stanford University, this structured dataset contains over 300,000 news articles and is distinguished by its two sources, CNN and Daily Mail. Each article includes several bullet-pointed summaries that capture its semantic content. These summaries serve as the target output for both abstractive and extractive summarization processes. The summaries derived from the CNN/Daily Mail dataset are relatively comprehensive, and the annotated data is of high quality, making this dataset suitable for training various text models and for benchmarking state-of-the-art summarization algorithms in NLP.

3.3. Word embedding techniques applied

As described and presented in Section 2, this study employs each of the word embedding techniques from the categories of traditional, static, and contextualized methods. For statistical-based text summarization techniques such as TF-IDF, TextRank, and LSA, the TF-IDF method is used for word embedding. In contrast, for RNN-based architectures, GloVe and ELMO are utilized, while the transformer-based model employs GPT and BERT.

3.4. Evaluation metrics applied

The study employs ROUGE metrics to evaluate both extractive and abstractive summaries produced by the various summarization methods used in the research. It is utilized because it is recognized as a widely adopted technique for assessment summaries.

Figure 14
Project design

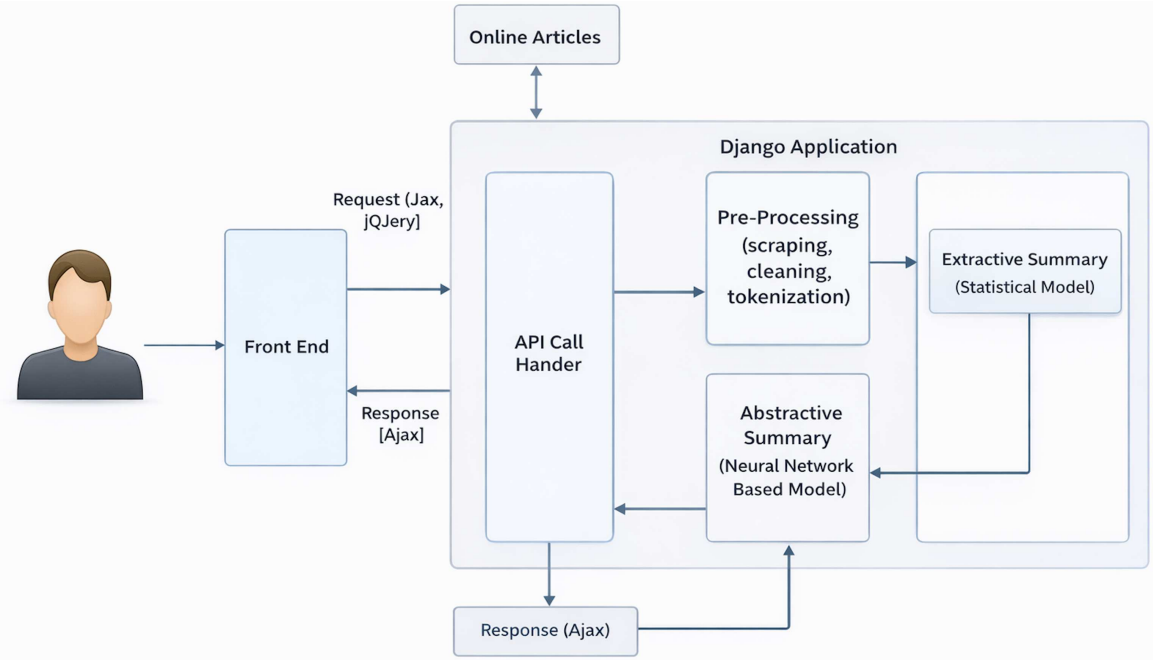
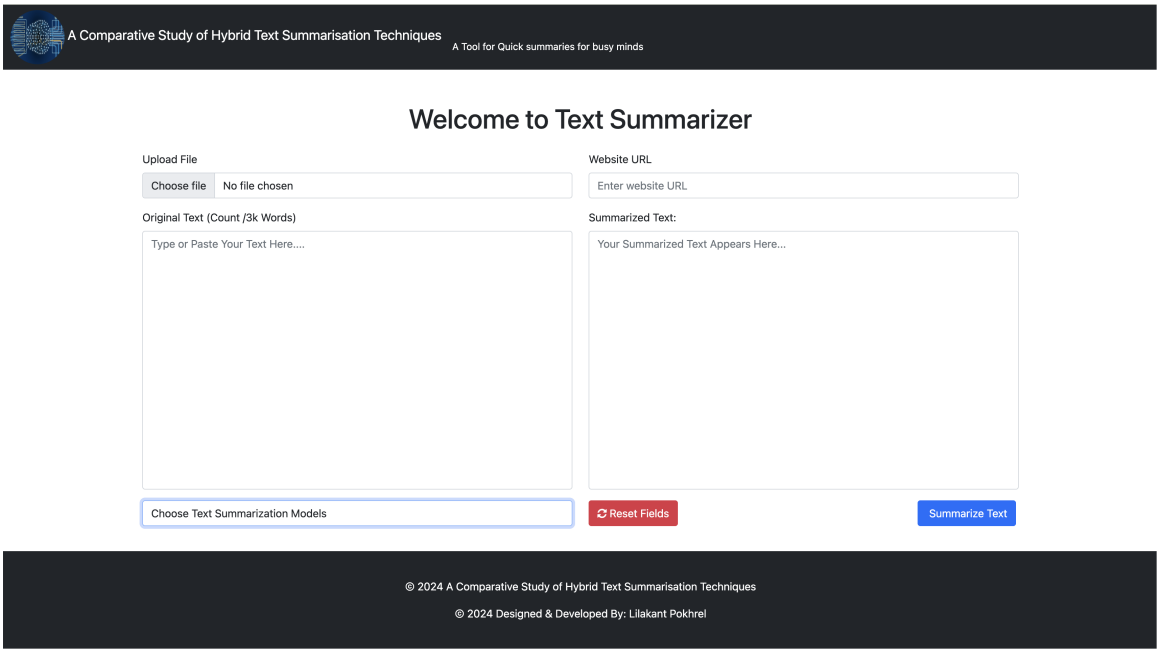


Figure 15
Project frontend look



3.5. Tools and technology used

The tools and technologies used include NLTK (NLP), TensorFlow (for implementing deep learning methods), and Scikit-learn (used for comprehensive evaluation, data analysis, dataset splitting, and testing). Additionally, web technologies such as Frontend (CSS, Bootstrap, jQuery) and Backend (Django) were used to deploy each of the models implemented in this study.

4. Results and Discussion

4.1. Project frontend look

Figure 15 in the previous chapter shows the frontend of the text summarization system. The frontend offers a field for uploading files (txt, docx), a field for URLs, and a custom text area for pasting the longer text to be summarized. The system also features a dropdown menu to select different text models for the summarization.

4.2. Experimental result

4.2.1. Experimental result of TF-IDF

As shown in Figure 16, ROUGE-1: The recall (*r*) value of 0.1578 indicates that approximately 15.78% of the single words in the generated summary match those in the reference summary. The precision value of 0.1406 shows that 14.06% of the unigrams in the generated summary are also present in the reference summary, and the F1-score of 0.14 reflects the unigram contribution in the overall generated summary.

ROUGE-2: The values for recall, precision, and F1-score in bigrams are all 0, indicating that there is no overlap between the pairs of words in the model-generated summary and the reference summary.

ROUGE-L: The recall value of 0.122 says that about 12.2% of the longest common sequences are identified between the generated and reference summaries. Figures 16 and 17 illustrate the ROUGE score and text used to evaluate the TF-IDF method.

4.2.2. Experimental result of TextRank

The result for the TextRank in ROUGE-1 displays higher recall (0.596) and moderate precision (0.333), which means the unigram or single word performed well. ROUGE-2 displays a higher recall (0.286), which indicates there was some kind of bigram overlap. ROUGE-L displays a reasonable sequence match (recall = 0.509, F1-score = 0.365). Thus, we can say that the summary is more normalized. Figures 18 and 19 illustrate the ROUGE score and text used to evaluate the TextRank method.

4.2.3. Experimental result of LSA

Figures 20 and 21 illustrate the ROUGE score and text used to evaluate the LSA.

4.2.4. Experimental result of Word2Vec

The result obtained using Word2Vec embedding is illustrated in Figure 22.

4.2.5. Experimental results of ELMo

The result obtained using ELMo embedding is illustrated in Figure 23.

4.2.6. Experimental result of GPT

The result obtained for the model GPT is illustrated in Figure 24.

4.3. Performance comparison

As shown in Table 3, an in-depth analysis of ROUGE scores from various text summarization models, such as TF-IDF, TextRank, LSA, Word2Vec, ELMo, Bi-LSTM with GloVe, and Bi-LSTM with ELMo embeddings, reveals challenges in accurately capturing words and sequence patterns in the generated summaries. The primary aim of the ROUGE measures is to evaluate the overlap or matching of words and phrases between the generated summary and the reference summary. High ROUGE scores indicate good content coverage and suggest that the generated summary closely matches the reference summary. The significance of high ROUGE scores varies between the two summarization approaches. High scores from extractive summarization methods such as TF-IDF, TextRank, LSA, and others indicate that the models have effectively identified important sentences or phrases from the original text. Therefore, high ROUGE scores in extractive summarization are considered positive and reflect the original content accurately.

Figure 16
TF-IDF result

```
#rouge score is to assess the performance text model
from rouge import Rouge
```

```
#checking rouge accuracy
rouge=Rouge()#create instance
rouge.get_scores(text_summary,df['summary'][2],avg=True,ignore_empty=True)#first one
```

```
{'rouge-1': {'r': 0.15789473684210525, 'p': 0.140625, 'f': 0.1487603255952464},
 'rouge-2': {'r': 0.0, 'p': 0.0, 'f': 0.0},
 'rouge-l': {'r': 0.12280701754385964,
  'p': 0.109375,
  'f': 0.1157024743555703}}
```

Figure 17
Text used for summarization

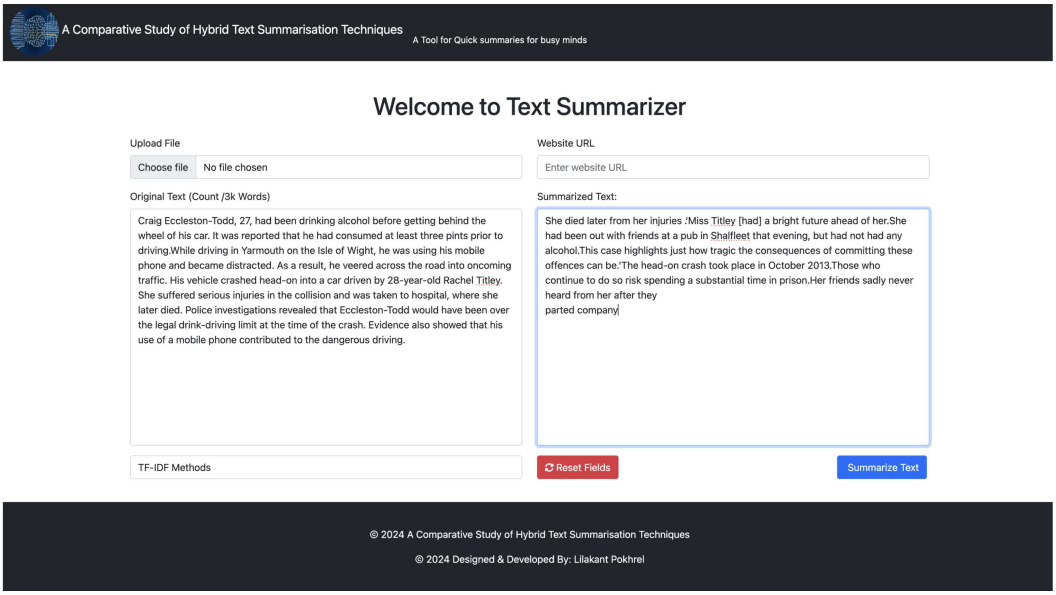


Figure 18
TextRank ROUGE score

```
In [ ]: #checking rouge accuracy
        rouge=Rouge()#create instance
        rouge.get_scores(summary_gen,df['summary'][2])#first one is generated, second is reference

Out[51]: [{'rouge-1': {'r': 0.5964912280701754,
             'p': 0.3333333333333333,
             'f': 0.42767295137534117},
            'rouge-2': {'r': 0.2857142857142857,
             'p': 0.1395348372093023,
             'f': 0.1874999955908204},
            'rouge-l': {'r': 0.5087719298245614,
             'p': 0.28431372549019607,
             'f': 0.3647798696143349}}]
```

Figure 19
Text used for summarization

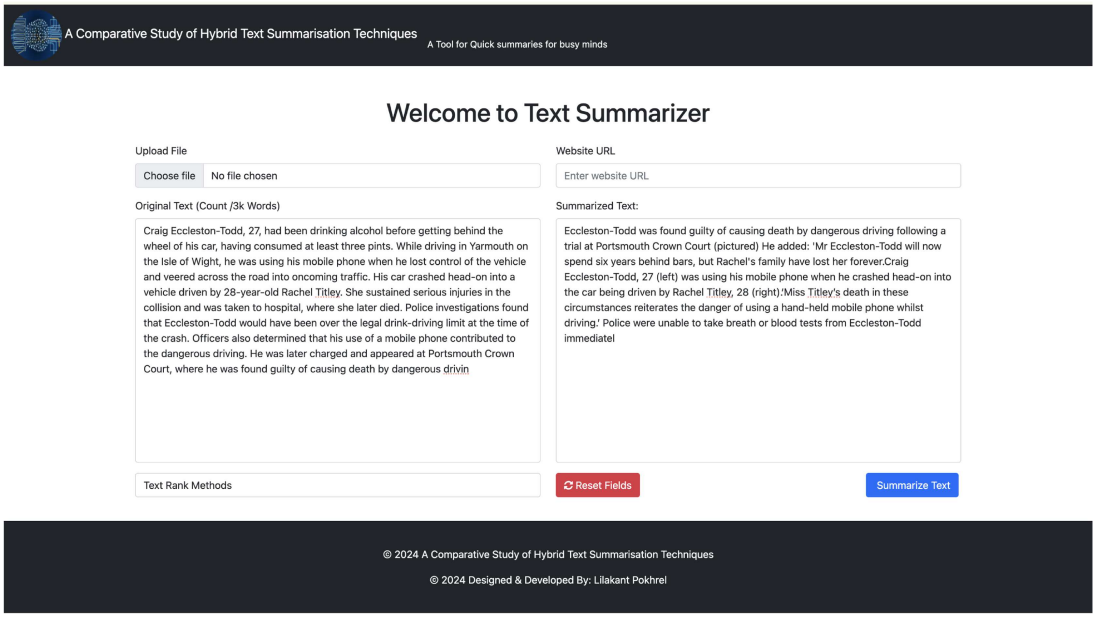


Figure 20
ROUGE score: LSA

```
#display the generated summary
#checking rouge accuracy
rouge=Rouge()#create instance
rouge.get_scores(gen_sum,df['summary'][2])
```

```
[{'rouge-1': {'r': 0.14035087719298245,
   'p': 0.08602150537634409,
   'f': 0.10666666195466687},
  'rouge-2': {'r': 0.015873015873015872,
   'p': 0.008771929824561403,
   'f': 0.011299430443361047},
  'rouge-l': {'r': 0.14035087719298245,
   'p': 0.08602150537634409,
   'f': 0.10666666195466687}}]
```

Figure 21
Text used for summarization: LSA

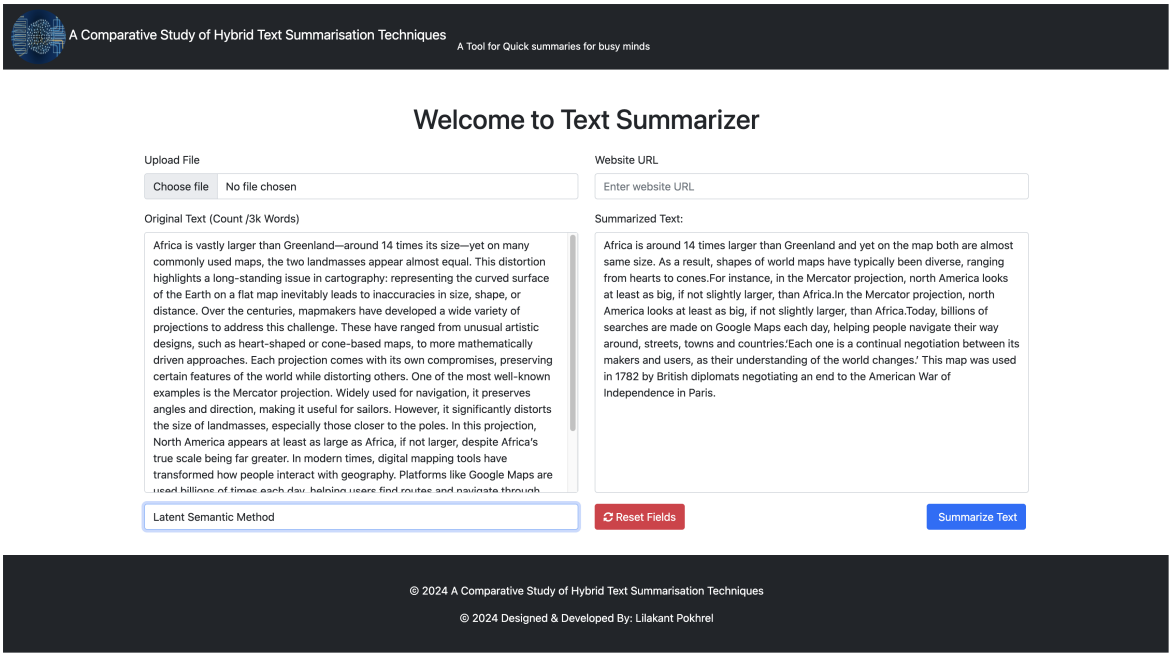


Figure 22
ROUGE score

```
#checking rouge accuracy
print("Rouge Score With Human Written Summary:")
rouge=Rouge()#create instance
rouge.get_scores(summary,df['highlights'][5])
```

```
Rouge Score With Human Written Summary:

[{'rouge-1': {'r': 0.3333333333333333,
   'p': 0.2033898305084746,
   'f': 0.25263157424044325},
  'rouge-2': {'r': 0.10256410256410256,
   'p': 0.0547945205479452,
   'f': 0.07142856688934979},
  'rouge-l': {'r': 0.3333333333333333,
   'p': 0.2033898305084746,
   'f': 0.25263157424044325}}]
```

Figure 23
ROUGE score ELMo

```
#checking rouge accuracy
print("Rouge Score With Human Written Summary:")
rouge=Rouge()#create instance
rouge.get_scores(generated_summary,df['highlights'][5])
```

```
Rouge Score With Human Written Summary:

[{'rouge-1': {'r': 0.4722222222222222,
   'p': 0.23943661971830985,
   'f': 0.31775700488077563},
  'rouge-2': {'r': 0.20512820512820512, 'p': 0.1, 'f': 0.1344537771061367},
  'rouge-l': {'r': 0.4166666666666667,
   'p': 0.2112676056338028,
   'f': 0.2803738273106822}}]
```

Figure 24
ROUGE score GPT

```
print("Rouge Score With Human Written Summary:")
rouge=Rouge()#create the instance
rouge.get_scores(summary,df['highlights'][5])
```

```
Rouge Score With Human Written Summary:

[{'rouge-1': {'r': 0.5277777777777778,
   'p': 0.25675675675675674,
   'f': 0.3454545410512397},
  'rouge-2': {'r': 0.23076923076923078,
   'p': 0.08653846153846154,
   'f': 0.12587412190718386},
  'rouge-l': {'r': 0.5277777777777778,
   'p': 0.25675675675675674,
   'f': 0.3454545410512397}}]
```

Table 3
Performance comparison

Methods	Roug-1		Rough-2		Rough-l		Type
	r	0.16	r	0	r	0.12	Extractive
TF-IDF	p	0.14	p	0	p	0.10	
	f	0.14	f	0	f	0.11	
	r	0.59	r	0.28	r	0.51	
TextRank	p	0.33	p	0.14	p	0.28	
	f	0.43	f	0.19	f	0.36	
	r	0.14	r	0.06	r	0.14	
LSA	p	0.086	p	0.008	p	0.086	
	f	0.11	f	0.011	f	0.10	
	r	0.33	r	0.10	r	0.33	
Word2Vec	p	0.20	p	0.054	p	0.20	
	f	0.25	f	0.071	f	0.25	
	r	0.47	r	0.20	r	0.41	Abstractive
ELMo	p	0.23	p	0.1	p	0.211	
	f	0.31	f	0.13	f	0.288	
	r	0.42	r	0.012	r	0.042	
Bi-LSTM GloVe	p	0.55	p	0.18	p	0.55	
	f	0.078	f	0.022	f	0.78	
	r	0.087	r	0.0	r	0.052	
Bi-LSTM ELMo	p	0.45	p	0.0	p	0.27	
	f	0.14	f	0.0	f	0.088	
	r	0.52	r	0.23	r	0.52	
GPT	p	0.256	p	0.086	p	0.25	
	f	0.34	f	0.12	f	0.34	

5. Conclusion

The performance of both summarization techniques (extractive and abstractive) using various word embedding methods (TF-IDF, Word2Vec, ELMo, GloVe) was examined to generate meaningful summaries that closely resemble the original content. Exploring the literature on word embedding techniques, identifying effective summarization methods, and identifying gaps and challenges in implementing a summarization system usable in real-world scenarios were the primary aims of the study. Among the eight summarization methods tested, the GPT-based embedding performed best, followed by TextRank and ELMo. GPT achieved high F1 scores, TextRank showed strong results on ROUGE-1, and ELMo performed well on ROUGE-2 and ROUGE-L metrics, producing coherent, meaningful summaries from the original text. The research also highlighted significant challenges and limitations in the summarization methods used. For example, developing an efficient summarization system requires large amounts of text data, powerful computing resources such as GPUS, and fast processing speeds. Additionally, the model needs to be trained over many epochs to achieve the desired accuracy, which is costly for students. This explains why early summarization methods tend to be trained on smaller datasets for fewer epochs, despite negatively affecting performance.

This study aims to compare various categories of text summarization methods, which have been implemented via a Django-based REST API. The proposed system is able to summarize input in different formats, such as URLs, raw text, and documents. Additionally, the system is able to present the output in a concise form. It applies all categories of the different text summarization methods. In fact, most previous studies have been limited to a single summarization method. Furthermore, this study offers detailed comparisons of the different categories of summarization techniques. This work generates systematic observations regarding the evaluation of various methods and also assists in selecting the most suitable method for future application studies.

6. Limitation

The text models are trained on the CNN/Daily Mail datasets with human-written summaries. The model may develop dataset-specific biases, which could limit the summarization system's language diversity and its ability to generalize to other datasets. Another limitation of the study concerns summarization approaches: extractive versus abstractive summarization. The summarization system may perform well with extractive methods but struggles to accurately capture information in datasets that require abstractive summarization. Additionally, the evaluation metrics present limitations. The commonly used metric for assessing summarization model performance is ROUGE. It mainly measures lexical overlap, which can make it difficult to evaluate the quality of abstractive summarization. Abstractive summaries often require human evaluation to improve accuracy. Finally, a challenge in the research is dealing with long documents. The available data consists of medium-length articles and associated human-written summaries. Models trained on such datasets might struggle to summarize longer texts or articles effectively.

7. Future Work

The research still requires further work to become more generalized in the field of text summarization. This could involve creating summarization systems that better adapt to a wider variety of texts, articles, and journals, ensuring improved coherence in abstractive summaries, and maintaining higher factual accuracy in more abstractive summaries. It would also entail developing new metrics beyond ROUGE scores that more accurately reflect how humans assess natural language input. These objectives aim to develop summarization tools that are more robust and versatile across a broader range of applications. The tools would become more personalized, adaptable, and capable of functioning in real time. The research also faces additional challenges related to scaling longer documents, integrating context and world knowledge, and reducing bias in systems output.

Acknowledgment

We acknowledge the use of generative AI tools, including ChatGPT, to enhance the clarity, flow, and presentation of the English language in this paper. These tools were employed solely for language refinement and structural coherence. The core ideas, research content, analysis, and conclusions presented herein are entirely our own and reflect our original intellectual contributions.

Ethical Statement

This study does not involve any research with human or animal subjects conducted by the author, but the study in question may raise long-term ethical questions about bias. One bias the model might learn is those present in the training dataset, which could raise ethical concerns. Another ethical consideration relates to misinformation about data privacy, where a model could (intentionally or unintentionally) propagate inaccuracies within a summary, resulting in important details being omitted or the misrepresentation of an athlete or team in a significant, potentially stereotypical way. There is also a risk that any automated summarization process may be overly simplistic or reductive, leading to a loss of nuance and complexity in analysis or interpretation. Similarly, there is potential for sensitive data to be exposed or misused during training or deployment, for instance, leading to the exposure of medical or other ethically protected data or data that should not be disclosed, as the model is trained on sources like the CNN/Daily Mail dataset.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

The data that support the findings of this study are openly available in Kaggle at https://www.kaggle.com/datasets/gowrishankarp/newspaper-text-summarization-cnn-dailymail/data, reference number [56].

Author Contribution Statement

Lilakant Pokhrel: Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing – original draft, Visualization. **Sangita Pokhrel:** Conceptualization, Methodology, Resources, Writing – review & editing, Supervision, Project administration. **Swathi Ganesan:** Validation, Resources,

Supervision. **Nalinda Somasiri:** Validation, Resources, Data curation, Supervision.

References

[1] Zhang, M., Zhou, G., Yu, W., Huang, N., & Liu, W. (2022). A comprehensive survey of abstractive text summarization based on deep learning. *Computational Intelligence and Neuroscience*, *2022*(1), 7132226. https://doi.org/10.1155/2022/7132226

[2] Huang, D., Cui, L., Yang, S., Bao, G., Wang, K., Xie, J., & Zhang, Y. (2020). What have we achieved on text summarization? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 446–469.

[3] Zhang, H., Yu, P. S., & Zhang, J. (2025). A systematic survey of text summarization: From statistical methods to large language models. *ACM Computing Surveys*, *57*(11), 1–41. https://doi.org/10.1145/3731445

[4] Johnson, S. J., Murty, M. R., & Navakanth, I. (2024). A detailed review on word embedding techniques with emphasis on word2vec. *Multimedia Tools and Applications*, *83*(13), 37979–38007. https://doi.org/10.1007/s11042-023-17007-z

[5] Ayman, U., Khan, A. R., Mahi, M. H., Akter, T., Koli, Z. M., & Bijoy, M. H. I. (2025). Optimizing Bengali sentiment analysis: A comparison of countvectorizer and TF-IDF with machine learning. In *2025 International Conference on Electrical, Computer and Communication Engineering*, 1–6. https://doi.org/10.1109/ECCE64574.2025.11013476

[6] Dai, S., Li, K., Luo, Z., Zhao, P., Hong, B., Zhu, A., & Liu, J. (2024). AI-based NLP section discusses the application and effect of bag-of-words models and TF-IDF in NLP tasks. *Journal of Artificial Intelligence General Science*, *5*(1), 13–21. https://doi.org/10.60087/jaigs.v5i1.149

[7] Zhou, H. (2022). Research of text classification based on TF-IDF and CNN-LSTM. *Journal of Physics: Conference Series*, *2171*(1), 012021. https://doi.org/10.1088/1742-6596%2F2171%2F1%2F012021

[8] Das, M., & Alphonse, P. J. A. (2023). A comparative study on TF-IDF feature weighting method and its analysis using unstructured dataset. *arXiv*. https://doi.org/10.48550/arXiv.2308.04037

[9] Hung, P. T., & Yamanishi, K. (2021). Word2vec skip-gram dimensionality selection via sequential normalized maximum likelihood. *Entropy*, *23*(8), 997. https://doi.org/10.3390/e23080997

[10] İrsoy, O., Benton, A., & Stratos, K. (2021). Corrected CBOW performs as well as skip-gram. In *Proceedings of the Second Workshop on Insights from Negative Results in NLP*, 1–8. https://doi.org/10.18653/v1/2021.insights-1.1

[11] de Oliveira, N. R., Pisa, P. S., Lopez, M. A., de Medeiros, D. S. V., & Mattos, D. M. (2021). Identifying fake news on social networks based on natural language processing: Trends and challenges. *Information*, *12*(1), 38. https://doi.org/10.3390/info12010038

[12] Wang, Y., Dong, L., Jiang, X., Ma, X., Li, Y., & Zhang, H. (2021). KG2Vec: A node2vec-based vectorization model for knowledge graph. *Plos One*, *16*(3), e0248552. https://doi.org/10.1371/journal.pone.0248552

[13] Iqbal, M. A., Sharif, O., Hoque, M. M., & Sarker, I. H. (2021). Word embedding based textual semantic similarity measure in Bengali. *Procedia Computer Science*, *193*, 92–101. https://doi.org/10.1016/j.procs.2021.10.010

[14] Zhou, J., Liu, L., Wei, W., & Fan, J. (2022). Network representation learning: From preprocessing, feature extraction to node embedding. *ACM Computing Surveys*, *55*(2), 1–35. https://doi.org/10.1145/3491206

[15] Wang, H. (2022). Extremal GloVe: Theoretically accurate distributed word embedding by tail inference. In *Proceedings of the 7th International Conference on Communication and Information Processing*, 1–3. https://doi.org/10.1145/3507971.3507972

[16] Alawadh, H. M., Alabrah, A., Meraj, T., & Rauf, H. T. (2023). Semantic features-based discourse analysis using deceptive and real text reviews. *Information*, *14*(1), 34. https://doi.org/10.3390/info14010034

[17] Wadud, M. A. H., Mridha, M. F., & Rahman, M. M. (2022). Word embedding methods for word representation in deep learning for natural language processing. *Iraqi Journal of Science*, *63*(3), 1349–1361. https://doi.org/10.24996/ijs.2022.63.3.37

[18] Sazan, S. A., Miraz, M. H., & Rahman, A. M. (2024). Enhancing depressive post detection in Bangla: A comparative study of TF-IDF, BERT and FastText embeddings. *Annals of Emerging Technologies in Computing (AETiC)*, *8*(3), 34–50. https://doi.org/10.33166/AETiC.2024.03.003

[19] Aljohani, N. R., Aslam, M. A., Khadidos, A. O., & Hassan, S.-U. (2022). A methodological framework to predict future market needs for sustainable skills management using AI and big data technologies. *Applied Sciences*, *12*(14), 6898. https://doi.org/10.3390/app12146898

[20] Hofmann, V., Pierrehumbert, J., & Schütze, H. (2021). Dynamic contextualized word embeddings. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, *1*, 6970–6984. https://doi.org/10.18653/v1/2021.acl-long.542

[21] Affi, M., & Latiri, C. (2021). BE-BLC: BERT-ELMO-based deep neural network architecture for English named entity recognition task. *Procedia Computer Science*, *192*, 168–181. https://doi.org/10.1016/j.procs.2021.08.018

[22] Zhang, M., Geng, G., & Chen, J. (2020). Semi-supervised bidirectional long short-term memory and conditional random fields model for named-entity recognition using embeddings from language models representations. *Entropy*, *22*(2), 252. https://doi.org/10.3390/e22020252

[23] Yenduri, G., Ramalingam, M., Selvi, G. C., Supriya, Y., Srivastava, G., Maddikunta, P. K. R., . . . , & Gadekallu, T. R. (2024). GPT (generative pre-trained transformer)—A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *IEEE Access*, *12*, 54608–54649. https://doi.org/10.1109/ACCESS.2024.3389497

[24] Tanaka, H., Shinnou, H., Cao, R., Bai, J., & Ma, W. (2020). Document classification by word embeddings of BERT. In *Computational Linguistics: 16th International Conference of the Pacific Association for Computational Linguistics*, 145–154. https://doi.org/10.1007/978-981-15-6168-9_13

[25] Grail, Q., Perez, J., & Gaussier, E. (2021). Globalizing BERT-based transformer architectures for long document summarization. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational*

Linguistics: Main Volume, 1792–1810. https://doi.org/10.18653/v1/2021.eacl-main.154

[26] Cheng, X., Mei, T., Zi, Y., Wang, Q., Gao, Z., & Yang, H. (2024). Algorithm research of ELMo word embedding and deep learning multimodal transformer in image description. In *2024 IEEE 6th International Conference on Power, Intelligent Computing and Systems*, 1391–1395. https://doi.org/10.1109/ICPICS62053.2024.10796626

[27] GeeksforGeeks. (2025). *How to generate word embedding using BERT?* https://www.geeksforgeeks.org/nlp/how-to-generate-word-embedding-using-bert/

[28] Azam, M., Khalid, S., Almutairi, S., Khattak, H. A., Namoun, A., Ali, A., & Bilal, H. S. M. (2025). Current trends and advances in extractive text summarization: A comprehensive review. *IEEE Access*, *13*, 28150–28166. https://doi.org/10.1109/ACCESS.2025.3538886

[29] Watanangura, P., Vanichrudee, S., Minteer, O., Sringamdee, T., Thanngam, N., & Siriborvornratanakul, T. (2023). A comparative survey of text summarization techniques. *SN Computer Science*, *5*(1), 47. https://doi.org/10.1007/s42979-023-02343-6

[30] El-Kassas, W. S., Salama, C. R., Rafea, A. A., & Mohamed, H. K. (2020). EdgeSumm: Graph-based framework for automatic text summarization. *Information Processing & Management*, *57*(6), 102264. https://doi.org/10.1016/j.ipm.2020.102264

[31] Zaware, S., Patadiya, D., Gaikwad, A., Gulhane, S., & Thakare, A. (2021). Text summarization using TF-IDF and TextRank algorithm. In *2021 5th International Conference on Trends in Electronics and Informatics*, 1399–1407. https://doi.org/10.1109/ICOEI51242.2021.9453071

[32] Fakhrezi, M. F., Bijaksana, M. A., & Huda, A. F. (2021). Implementation of automatic text summarization with TextRank method in the development of Al-Qur'an vocabulary encyclopedia. *Procedia Computer Science*, *179*, 391–398. https://doi.org/10.1016/j.procs.2021.01.021

[33] Deo, S., & Banik, D. (2022). Text summarization using Textrank and Lexrank through latent semantic analysis. In *2022 OITS International Conference on Information Technology*, 113–118. https://doi.org/10.1109/OCIT56763.2022.00031

[34] Ghorpade, S., Khan, A., Chaurasia, A., Rao, V., & Chhabria, A. (2022). A comparative analysis of TextRank and LexRank algorithms using text summarization. In *International Joint Conference on Advances in Computational Intelligence*, 379–393. https://doi.org/10.1007/978-981-97-0180-3_30

[35] Dessi, D., Helaoui, R., Kumar, V., Reforgiato Recupero, D., & Riboni, D. (2020). TF-IDF vs word embeddings for morbidity identification in clinical notes: An initial study. In *CEUR Workshop Proceedings*, *2596*, 1–12. https://iris.unica.it/handle/11584/293694

[36] Mridha, M. F., Lima, A. A., Nur, K., Das, S. C., Hasan, M., & Kabir, M. M. (2021). A survey of automatic text summarization: Progress, process and challenges. *IEEE Access*, *9*, 156043–156070. https://doi.org/10.1109/ACCESS.2021.3129786

[37] Karim, A. A., Ibrahim, M. A., Hosny, M. M., Ibrahim, A. H., Mohamed, R. A., Elbanbawy, M., . . . , & Moawad, I. F. (2024). A Comparative analysis of Arabic text summarization techniques: Evaluating word frequency, K-Means clustering, and PageRank algorithm. In *2024 International Mobile, Intelligent, and Ubiquitous Computing Conference*, 257–264. https://doi.org/10.1109/MIUCC62295.2024.10783534

[38] Easwar, A., & Uthra, A. (2021). Automatic text summarization using word embeddings. In *International Conference on I-SMAC*, 1065–1079. https://doi.org/10.1109/I-SMAC52330.2021.9640746

[39] Nguyen, T.-H., & Do, T.-N. (2022). Extractive text summarization on large-scale dataset using K-means clustering. In *35th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 737–746. https://doi.org/10.1007/978-3-031-08530-7_62

[40] Pokhrel, C., & Adhikari, R. (2023). Automatic extractive text summarization for text in Nepali language with bidirectional encoder representation transformers and k-mean clustering. *Research Gate*. https://doi.org/10.13140/RG.2.2.15823.87200

[41] Shafiq, N., Hamid, I., Asif, M., Nawaz, Q., Aljuaid, H., & Ali, H. (2023). Abstractive text summarization of low-resourced languages using deep learning. *PeerJ Computer Science*, *9*, e1176. https://doi.org/10.7717/peerj-cs.1176

[42] Singh, M., & Yadav, V. (2022). Abstractive text summarization using attention-based stacked LSTM. In *2022 Fifth International Conference on Computational Intelligence and Communication Technologies*, 236–241. https://doi.org/10.1109/CCiCT56684.2022.00052

[43] Preethi, S., Krithick Shibi, M. S., Sheshan, S., Grace, R. K., & Geetha, M. S. (2022). Abstractive summarizer using Bi-LSTM. In *2022 International Conference on Edge Computing and Applications*, 1605–1609. https://doi.org/10.1109/ICECAA55415.2022.9936215

[44] Kades, K., Sellner, J., Koehler, G., Full, P. M., Lai, T. E., Kleesiek, J., & Maier-Hein, K. H. (2021). Adapting bidirectional encoder representations from transformers (BERT) to assess clinical semantic textual similarity: Algorithm development and validation study. *JMIR Medical Informatics*, *9*(2), e22795. https://doi.org/10.2196/22795

[45] Shi, T., Keneshloo, Y., Ramakrishnan, N., & Reddy, C. K. (2021). Neural abstractive text summarization with sequence-to-sequence models. *ACM Transactions on Data Science*, *2*(1), 1–37. https://doi.org/10.1145/3419106

[46] Liu, Y., & Lapata, M. (2019). Text summarization with pretrained encoders, In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing,*, 3730–3740.

[47] Zhang, Y., Liu, C., Liu, M., Liu, T., Lin, H., Huang, C. B., & Ning, L. (2024). Attention is all you need: Utilizing attention in AI-enabled drug discovery. *Briefings in Bioinformatics*, *25*(1), 1–22. https://doi.org/10.1093/bib/bbad467

[48] See, A., Liu, P. J., & Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, *1*, 1073–1083. https://doi.org/10.18653/v1/P17-1099

[49] Gao, T., Fisch, A., & Chen, D. (2021). Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, *1*. 3816–3830. https://doi.org/10.18653/v1/2021.acl-long.295

[50] Chang, T.-M., & Hsiao, W.-F. (2008). A hybrid approach to automatic text summarization. In *2008 8th IEEE International Conference on Computer and Information Technology*, 65–70. https://doi.org/10.1109/CIT.2008.4594651

[51] Koh, H. Y., Ju, J., Liu, M., & Pan, S. (2022). An empirical survey on long document summarization: Datasets, models, and metrics. *ACM Computing Surveys*, *55*(8), 154. https://doi.org/10.1145/3545176

[52] Akter, M., Bansal, N., & Karmaker, S. K. (2022). Revisiting automatic evaluation of extractive summarization task: Can we do better than ROUGE? In *Findings of the Association for Computational Linguistics: ACL 2022,* 1547–1560. https://doi.org/10.18653/v1/2022.findings-acl.122

[53] Thompson, B., & Post, M. (2020). Automatic machine translation evaluation in many languages via zero-shot paraphrasing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, 90–121.

[54] Roy, D., Fakhoury, S., & Arnaoudova, V. (2021). Reassessing automatic evaluation metrics for code summarization tasks. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 1105–1116. https://doi.org/10.1145/3468264.3468588

[55] Abubakar, H. D., Umar, M., & Bakale, M. A. (2022). Sentiment classification: Review of text vectorization methods: Bag of words, TF-IDF, Word2vec and Doc2vec. *Sule Lamido University Journal of Science & Technology*, *4*(1-2), 27–33.

[56] Chandra, M., Gupta, V., & Paul, S. K. (2011). A statistical approach for automatic text summarization by extraction. In *2011 International Conference on Communication Systems and Network Technologies*, 268–271. https://doi.org/10.1109/CSNT.2011.65

How to Cite: Pokhrel, L., Pokhrel, S., Ganesan, S., & Somasiri, N. (2026). A Comparative Study of Hybrid Text Summarization Techniques Using Traditional, Static, and Contextual Word Embeddings. *Artificial Intelligence and Applications*. https://doi.org/10.47852/bonviewAIA62026330