Acknowledgement

Throughout this study we received a great deal of support. First, we would like to thank our supervisor Dr. Amr Ghoneim who provided us support, guidance and motivation through every step in our project.

Table of contents

Abstract

Natural Language Processing (NLP) is a rapidly growing field that has seen an explosion of interest and research in recent years. This is largely due to the increasing amount of textual data available, as well as the need for machines to be able to understand and interact with human language. BART, Pegasus, and T5 are some of the most advanced and widely used NLP models available today, each with its unique strengths and weaknesses.

The CNN/Daily Mail dataset is a popular dataset used in NLP research that contains news articles and summaries. It has been used to evaluate the performance of various NLP models, including BART, Pegasus, and T5.

In this documentation, we focus on the concept of stacking ensemble, which involves combining multiple models to improve their overall performance. We show how to implement stacking ensemble with BART, Pegasus, and T5 models on the CNN/Daily Mail dataset.

Our experimental results demonstrate that stacking ensemble can significantly improve the performance of NLP models on the CNN/Daily Mail dataset, compared to using individual models alone. We discuss the potential impact of our work on the NLP community, as well as future research directions and improvements to the techniques presented.

This documentation is intended to provide a practical guide for NLP researchers, developers, and practitioners who are interested in improving the performance of NLP models using stacking ensemble techniques. By implementing the techniques presented in this documentation, researchers can improve the performance of their NLP models and contribute to the advancement of the field.

Keywords

- Natural Language Processing (NLP)
- BART
- Pegasus
- T5
- Stacking ensemble
- CNN/Daily Mail dataset
- Text data
- Hyperparameters
- Optimization techniques
- Performance evaluation
- Experimental results
- NLP tasks
- Language ambiguity
- Language diversity
- Language complexity
- NLP models
- NLP research
- NLP community
- Machine learning
- Artificial intelligence (AI)
- Deep learning
- Neural networks.

An Introduction

1.1 Overview:

Natural Language Processing (NLP) is an area of Artificial Intelligence (AI) that deals with the interaction between computers and humans using natural language. In recent years, several powerful NLP models have been developed, including BART, Pegasus, and T5. These models have shown great performance on various NLP tasks such as language generation, summarization, and text classification.

In this project, the focus is on merging BART, Pegasus, and T5 models using a stacking ensemble approach on the CNN Daily Mail dataset. The stacking ensemble approach involves combining multiple models to improve their performance on a given task. The CNN Daily Mail dataset is a widely used dataset in NLP for summarization tasks.

The three models - BART, Pegasus, and T5 - have different architectures and strengths. BART is a sequence-to-sequence model that uses a bidirectional encoder and a left-to-right decoder. Pegasus, on the other hand, is a transformer-based model that uses a pre-training and fine-tuning approach for abstractive summarization. T5 is a text-to-text transformer model that can be fine-tuned for various NLP tasks.

The models are preprocessed, meaning that the input data has been cleaned and transformed to a format that the models can understand. The merging approach involves combining the outputs of the three models to improve the overall performance. The stacking ensemble approach involves training a meta-model to learn the optimal combination of the three models' outputs.

Notably, this project does not involve any fine-tuning of the models. Fine-tuning is a process of training a pre-trained model on a specific task, which often leads to improved performance. However, the models used in this project are pre-trained and have not been fine-tuned on the CNN Daily Mail dataset.

In summary, this project aims to merge BART, Pegasus, and T5 models using a stacking ensemble approach on the CNN Daily Mail dataset. The models are preprocessed, and no fine-tuning is performed. The project's goal is to achieve improved performance on the summarization task through the combination of the three models.

1.2 Problem Statement:

Natural Language Processing (NLP) is a rapidly evolving field of study that aims to develop algorithms that can process, understand, and generate human language. Summarization is one of the essential tasks in NLP, where the goal is to generate a concise and coherent summary of a longer document.

Although recent developments in NLP have led to the creation of powerful models such as BART, Pegasus, and T5, individual models often have limitations in their performance. Therefore, there is a need to develop better summarization models that can achieve higher accuracy and better efficiency.

One approach to improving the performance of NLP models is through ensemble methods, which involve combining the outputs of multiple models to achieve better results than any individual model alone. Stacking is one such ensemble method that involves training a meta-model that learns how to combine the outputs of multiple base models. Stacking has been shown to be an effective approach in various NLP tasks, including summarization.

The CNN Daily Mail dataset is a popular benchmark dataset for summarization tasks, and it has been used to evaluate the performance of various NLP models. The dataset contains over 300,000 news articles and their corresponding summaries.

The problem statement of this project is to develop a better summarization model by merging BART, Pegasus, and T5 models using a stacking ensemble approach on the CNN Daily Mail dataset. The models are preprocessed, meaning that the input data has been cleaned and transformed to a format that the models can understand. Importantly, the models will not be fine-tuned, which is a common approach to improving the performance of NLP models.

The main goal of this project is to improve the performance of the summarization task on the CNN Daily Mail dataset by combining the strengths of the BART, Pegasus, and T5 models through a stacking ensemble approach. By doing so, the project aims to contribute to the development of more accurate and efficient NLP models for summarization tasks.

1.3 Scope and Objectives:

The scope of this project is to develop a better summarization model by merging BART, Pegasus, and T5 models using a stacking ensemble approach on the CNN Daily Mail dataset. The focus of the project is on improving the summarization task on the dataset, which involves generating a concise and coherent summary of a given news article. The project aims to achieve this through the following objectives:

1. Preprocess the CNN Daily Mail dataset: The dataset needs to be cleaned and transformed into a format that can be understood by the BART, Pegasus, and T5 models.
2. Develop a stacking ensemble approach: The project aims to develop a meta-model that learns how to combine the outputs of the three base models - BART, Pegasus, and T5 - to improve the overall performance of the summarization task.
3. Implement the stacking ensemble approach: The project will implement the developed stacking ensemble approach to merge the base models and generate summaries for the news articles in the CNN Daily Mail dataset.
4. Evaluate the performance of the model: The project will evaluate the performance of the developed model by comparing it with the individual base models' performance and other state-of-the-art models on the CNN Daily Mail dataset.

The objectives of the project will be achieved by following these steps:

1. Preprocessing the CNN Daily Mail dataset by cleaning and transforming the input data into a format that can be understood by the models.
2. Training the three base models - BART, Pegasus, and T5 - on the preprocessed dataset.
3. Generating summaries for the news articles in the CNN Daily Mail dataset using each of the base models.
4. Developing a meta-model that learns how to combine the outputs of the base models through a stacking ensemble approach.
5. Implementing the developed stacking ensemble approach to merge the base models and generate summaries for the news articles in the CNN Daily Mail dataset.
6. Evaluating the performance of the developed model by comparing it with the individual base models' performance and other state-of-the-art models on the CNN Daily Mail dataset.

In summary, the project's scope is to develop a better summarization model by merging BART, Pegasus, and T5 models using a stacking ensemble approach on the CNN Daily Mail dataset. The project's objectives are to preprocess the dataset, develop a stacking ensemble approach, implement the approach, and evaluate the performance of the model. By achieving these objectives, the project aims to contribute to the development of more accurate and efficient NLP models for summarization tasks.

1.4 Work Methodology:

During our project we encountered several phases as explained as follows:

Phase one: in this phase we focused on how to read and understand research papers, getting papers with high citations and reading about important journals in machine learning and specifically about NLP and text summarization field.

Phase two: during this phase we started taking courses to get better understanding, then we selected our individual models (T5, Bart, Pegasus) and applied it on CNN\DAILYMAIL dataset.

Phase three: in this point we began to read about ensemble learning and different merge techniques to get better understanding and selected the most proper method to merge our models to achieve the best performance.

Phase four: we began writing our comparative study and our list of experiments and its results.

Introduction to NLP

2.1 Explanation of NLP:

Natural Language Processing (NLP) is a rapidly growing field of study that focuses on developing algorithms to enable computers to understand, interpret, and generate human language. NLP is a subfield of Artificial Intelligence and has gained significant attention in recent years due to its wide range of applications in various industries such as healthcare, finance, and education.

The main goal of NLP is to enable computers to understand and communicate with humans in natural language, just like humans do with each other. NLP algorithms achieve this through various techniques such as machine learning, deep learning, and natural language understanding. These techniques allow computers to analyze and understand human language patterns, structures, and meanings, and generate responses accordingly.

One of the main applications of NLP is in language translation. NLP algorithms can translate text from one language to another, allowing for seamless communication between people who speak different languages. Another application of NLP is in sentiment analysis, where algorithms can analyze text and determine the sentiment behind it, whether positive, negative, or neutral. This application is used in social media monitoring and customer service to determine customer satisfaction levels and sentiment towards a product or service.

NLP is also used in chatbots and virtual assistants, where algorithms can understand and respond to human queries in natural language. This technology is used in customer service, healthcare, and education to provide personalized and automated responses to customer queries. Additionally, NLP is used in summarization and information extraction, where algorithms can extract relevant information from large volumes of text and summarize it for easy consumption.

Despite its wide range of applications, NLP is still a challenging field due to the complexity of human language. Human language is diverse and constantly evolving, making it difficult for algorithms to understand and interpret correctly. Additionally, NLP algorithms must consider the context, tone, and cultural nuances of language to provide accurate and meaningful responses.

In conclusion, NLP is a rapidly growing field of study that has significant implications for various industries. Its applications in language translation, sentiment analysis, chatbots, and summarization are transforming the way humans interact with computers. However, significant challenges still exist in the field, and researchers are continually working to develop more robust and accurate NLP algorithms. As NLP continues to evolve, it will undoubtedly play a vital role in shaping the future of human-computer interaction.

NLP is a multidisciplinary field that draws on linguistics, computer science, and cognitive psychology. It involves the use of statistical and machine learning techniques to extract meaning from text data and make predictions about language use. NLP algorithms can be trained on large datasets of text, allowing them to learn patterns and relationships in language and use this knowledge to analyze and generate text. As NLP continues to advance, it has the potential to revolutionize many aspects of society, including healthcare, education, and business. With further development and refinement, NLP could lead to faster, more accurate communication between people and computers and transform the way we interact with technology.

2.2 Challenges of NLP:

One of the major challenges of NLP is the ambiguity of human language. Human language is complex and often ambiguous, making it challenging for algorithms to understand and interpret correctly. This is particularly true for language that is context-dependent and relies heavily on cultural and social context, such as idiomatic expressions, sarcasm, and metaphorical language. NLP algorithms must be able to understand the nuances of human language to provide accurate and meaningful responses, which requires significant computational power and resources.

Another challenge of NLP is the lack of high-quality data. NLP algorithms require large datasets of text data to learn patterns and relationships in language. However, these datasets are often difficult to obtain, particularly for niche or specialized domains. Additionally, the quality of the data can vary widely, with some datasets containing errors, biases, or inconsistencies that can impact the accuracy of the algorithms.

NLP also faces challenges related to language diversity. Human language is diverse and constantly evolving, with thousands of languages and dialects spoken around the world. NLP algorithms must be able to handle this diversity and adapt to new languages and dialects as they emerge. However, this requires significant computational resources and expertise in linguistics and language-specific features, making it challenging for researchers to keep up with the pace of language evolution.

Another challenge of NLP is the lack of interpretability of the algorithms. NLP algorithms are often highly complex and difficult to interpret, making it challenging for researchers to understand how they arrive at their decisions. This lack of interpretability can be a significant barrier to the adoption of NLP algorithms in real-world applications, particularly in domains where transparency and accountability are crucial, such as healthcare and finance.

Privacy and ethical concerns are also a major challenge for NLP. NLP algorithms often require access to large amounts of personal data, such as social media posts and medical records, to train and develop accurate models. However, this poses significant privacy concerns, particularly in light of recent data breaches and privacy scandals. Additionally, there are ethical concerns around the use of NLP algorithms for applications such as surveillance, profiling, and censorship.

Finally, NLP faces challenges related to computational resources and scalability. NLP algorithms require significant computational power and resources to process large amounts of text data and generate meaningful responses. However, the cost of these resources can be prohibitive, particularly for smaller organizations and researchers with limited budgets. Additionally, as the amount of text data continues to grow exponentially, there is a need for scalable NLP algorithms that can handle this volume of data efficiently.

Despite these challenges, significant progress has been made in NLP in recent years, with many promising applications in healthcare, finance, and education. Researchers are continually developing new algorithms and techniques to overcome the challenges of NLP, such as developing more accurate models, improving the quality of data, and increasing the interpretability of algorithms. Additionally, advances in machine learning and cloud computing have made NLP more accessible and affordable, allowing smaller organizations and researchers to develop and deploy NLP models.

In conclusion, NLP is a rapidly evolving field that faces several challenges related to the complexity of human language, the lack of high-quality data, language diversity, interpretability, privacy and ethical concerns, and computational resources and scalability. These challenges limit the potential of NLP for real-world applications and underscore the need for continued research and development in the field. As NLP continues to evolve, it has the potential to transform many aspects of society, but researchers must address these challenges to unlock its full potential.

2.3 Overview of NLP Models:

Natural Language Processing (NLP) has revolutionized the way we interact with computers and has transformed many industries such as healthcare, finance, and education. At the heart of NLP are NLP models, which enable computers to understand, interpret, and generate human language. In this article, we will provide an overview of NLP models, their types, and applications.

Types of NLP Models:

1. Rule-Based Models: Rule-based models use explicit rules and patterns to analyze and generate text. These models rely on hand-crafted rules and linguistic knowledge to extract meaning from text. While these models can be effective in certain domains, they are limited by their inflexibility, as they are unable to learn from data and adapt to new contexts.
2. Statistical Models: Statistical models use probabilistic methods and machine learning techniques to analyze and generate text. These models learn from large datasets of text and use statistical methods to identify patterns and relationships in language. Examples of statistical models include Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs).
3. Neural Network Models: Neural network models are a type of machine learning model that uses artificial neural networks to analyze and generate text. These models are highly flexible and can learn from large datasets of text, making them well-suited for complex NLP tasks such as language translation and sentiment analysis. Examples of neural network models include Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformer models.
4. Pretrained Language Models: Pretrained language models are models that are trained on large datasets of text and can be fine-tuned for specific tasks. These models have achieved state-of-the-art results on a wide range of NLP tasks, including language translation, sentiment analysis, and

question answering. Examples of pretrained language models include BERT, GPT-3, and T5.

5. Hybrid Models: Hybrid models combine multiple NLP models to achieve better performance on complex NLP tasks. These models can combine various types of models, such as rule-based, statistical, and neural network models, to leverage their strengths and overcome their limitations.

Applications of NLP Models:

1. Language Translation: NLP models can be used to translate text from one language to another, allowing for seamless communication between people who speak different languages. Neural network models and pretrained language models have achieved state-of-the-art results in this domain.
2. Sentiment Analysis: NLP models can analyze text and determine the sentiment behind it, whether positive, negative, or neutral. This application is used in social media monitoring and customer service to determine customer satisfaction levels and sentiment towards a particular product or service.
3. Chatbots and Virtual Assistants: NLP models can be used to develop chatbots and virtual assistants that can understand and respond to human queries in natural language. This technology is used in customer service, healthcare, and education to provide personalized and automated responses to customer queries.
4. Summarization and Information Extraction: NLP models can extract relevant information from large volumes of text and summarize it for easy consumption. This application is used in news aggregation, research, and business intelligence.
5. Speech Recognition: NLP models can be used to recognize and transcribe human speech, allowing for hands-free interaction with computers and mobile devices. This application is used in virtual assistants, dictation software, and call centers.

In conclusion, NLP models have transformed the way we interact with computers and have enabled many applications in various industries. Rule-based, statistical, neural network, pretrained language, and hybrid models are some of the most used NLP models, each with their strengths and weaknesses. As the field of NLP continues to evolve, researchers and developers will continue to improve the performance and accuracy of NLP models, unlocking new opportunities for innovation and growth.

2.4 Stacking Ensemble in NLP:

There are many ways to ensemble models in machine learning, such as Bagging, Boosting, and stacking. ***Stacking is one of the most popular ensemble machine learning techniques used to predict multiple nodes to build a new model and improve model performance.*** Stacking enables us to train multiple models to solve similar problems, and based on their combined output, it builds a new model with improved performance.

In this topic, "**Stacking in Machine Learning**", we will discuss a few important concepts related to stacking, the general architecture of stacking, important key points to implement stacking, and how stacking differs from **bagging** and **boosting** in machine learning. Before starting this topic, first, understand the concepts of the ensemble in machine learning. So, let's start with the definition of ensemble learning in machine learning.

What is Ensemble learning in Machine Learning?

Ensemble learning is one of the most powerful machine learning techniques that use the combined output of two or more models/weak learners and solve a particular computational intelligence problem. E.g., a Random Forest algorithm is an ensemble of various decision trees combined.

Ensemble learning is primarily used to improve the model performance, such as classification, prediction, function approximation, etc. In simple words, we can summarise the ensemble learning as follows:

"An ensembled model is a machine learning model that combines the predictions from two or more models."

There are 3 most common ensemble learning methods in machine learning. These are as follows:

- Bagging
- Boosting
- Stacking

However, we will mainly discuss Stacking on this topic.

1. Bagging

Bagging is a method of ensemble modeling, which is primarily used to solve supervised machine learning problems. It is generally completed in two steps as follows:

- **Bootstrapping:** It is a random sampling method that is used to derive samples from the data using the replacement procedure. In this method, first, random data samples are fed to the primary model, and then a base learning algorithm is run on the samples to complete the learning process.
- **Aggregation:** This is a step that involves the process of combining the output of all base models and, based on their output, predicting an aggregate result with greater accuracy and reduced variance.

Example: In the Random Forest method, predictions from multiple decision trees are ensembled parallelly. Further, in regression problems, we use an average of these predictions to get the final output, whereas, in classification problems, the model is selected as the predicted class.

2. Boosting

Boosting is an ensemble method that enables each member to learn from the preceding member's mistakes and make better predictions for the future. Unlike the bagging method, in boosting, all base learners (weak) are arranged in a sequential format so that they can learn from the mistakes of their preceding learner. Hence, in this way, all weak learners get turned into strong learners and make a better predictive model with significantly improved performance.

We have a basic understanding of ensemble techniques in machine learning and their two common methods, i.e., bagging and boosting. Now, let's discuss a different paradigm of ensemble learning, i.e., Stacking.

3. Stacking

Stacking is one of the popular ensemble modeling techniques in machine learning. Various weak learners are ensembled in a parallel manner in such a way that by combining them with Meta learners, we can predict better predictions for the future.

This ensemble technique works by applying input of combined multiple weak learners' predictions and Meta learners so that a better output prediction model can be achieved.

In stacking, an algorithm takes the outputs of sub-models as input and attempts to learn how to best combine the input predictions to make a better output prediction.

Stacking is also known as **a stacked generalization** and is an extended form of the Model Averaging Ensemble technique in which all sub-models equally participate as per their performance weights and build a new model with better predictions. This new model is stacked up on top of the others; this is the reason why it is named stacking.

2.5 Target Audience of the Documentation:

the target audience is likely to be researchers, developers, and practitioners in the field of NLP and machine learning. This audience should have a basic understanding of NLP concepts and techniques, as well as experience with the programming languages and tools used in the project.

The documentation may also be of interest to students and educators in NLP and machine learning courses, as it provides a practical example of merging and stacking NLP models without fine-tuning on a real-world dataset. The documentation may also be helpful for professionals in related fields who are interested in the potential applications of NLP models.

The target audience is likely to have a technical background and a strong interest in NLP and machine learning research and development. They may be seeking to learn more about the performance and capabilities of different NLP models and techniques, as well as practical applications of these models in real-world scenarios.

3:Overview of BART, Pegasus, and T5

3.1 Explanation of the Architecture:

BART:

What is the BART Transformer Model in NLP?

HuggingFace Transformer models provide an easy-to-use implementation of some of the best performing models in natural language processing. Transformer models are the current state-of-the-art (SOTA) in several NLP tasks such as text classification, text generation, text summarization, and question answering.
The original Transformer is based on an encoder-decoder architecture and is a classic sequence-to-sequence model. The model's input and output are in the form of a sequence (text), and the encoder learns a high-dimensional representation of the input,which is then mapped to the output by the decoder. This architecture introduced a new form of learning for language-related tasks and, thus, the models spawned from it achieve outstanding results overtaking the existing deep neural network-based methods.

Since the inception of the vanilla Transformer, several recent models inspired by the Transformer used the architecture to improve the benchmark of NLP tasks. Transformer models are first pre-trained on a large text corpus (such as BookCorpus or Wikipedia). This pretraining makes sure that the model "understands language" and has a decent starting point to learn how to perform further tasks. Hence, after this step, we only have a language model. The ability of the model to understand language is highly significant since it will determine how well you can

further train the model for something like text classification or text summarization.

BART is one such Transformer model that takes components from other Transformer models and improves the pretraining learning. BART or Bidirectional and Auto-Regressive
Transformers was proposed in the BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension paper. The BART HugggingFace model allows the pre-trained weights and weights fine-tuned on question-answering, text summarization, conditional text generation, mask filling, and sequence classification.
So without much ado, let's explore the BART model – the uses, architecture, working, as well as a HuggingFace example.

What is BART Model used for?

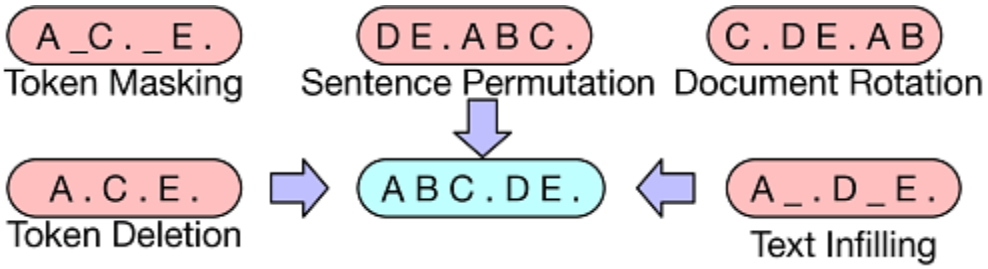
As mentioned in the original paper, BART is a sequence-to-sequence model trained as a denoising autoencoder. This means that a fine-tuned BART model can take a text sequence (for example, English) as input and produce a different text sequence at the output (for example, French). This type of model is relevant for machine translation (translating text from one language to another), question-answering (producing answers for a given question on a specific corpus), text summarization (giving a summary of or paraphrasing a long text document), or sequence classification (categorizing input text sentences or tokens). Another task is sentence entailment which, given two or more sentences, evaluates whether the sentences are logical extensions or are logically related to a given statement.
Since the unsupervised pretraining of BART results in a language model, we can fine-tune this language model to a specific task in NLP. Because the model has already been pre-trained, fine-tuning does not need massive labeled datasets (relative to what one would need for training from scratch). The BART model can be fine-tuned to domain-specific datasets to develop

applications such as medical conversational chatbots, converting natural text to programming code or SQL queries, context-specific language translation apps, or a tool to paraphrase research papers.

What Data was BART Model Trained on?

BART was trained as a denoising autoencoder, so the training data includes “corrupted” or “noisy” text, which would be mapped to clean or original text. The training format is similar to the training of any denoising autoencoder. Just how in computer vision, we train autoencoders to remove noise or improve the quality of an image by having noisy images in the training data mapped with clean, original images as the target.
So, what exactly counts as noise for text data? The authors of BART settle on using some existing and some new noising techniques for pretraining. The noising schemes they use are Token Masking, Token Deletion, Text Infilling, Sentence Permutation, and Document Rotation. Looking into each of these transformations:

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



However, not all transformations are employed in training the final BART model. Based on a comparative study of pre-training objectives, the authors use only text infilling and sentence permutation transformations, with about 30% of tokens being masked and all sentences permuted. These transformations are applied to 160GB of text from the English Wikipedia and BookCorpus dataset. With this dataset, the vocabulary size is around 29000, and the maximum length of the sequences is 512 characters in the clean data.

How many Parameters does BART have?

BART is constructed from a bi-directional encoder like in BERT and an autoregressive decoder like GPT. BERT has around 110M parameters while GPT has 117M, such trainable weights. BART being a sequenced version of the two, fittingly has nearly 140M parameters. Many parameters are justified by the supreme performance it yields on several tasks compared to fine-tuned BERT or its variations like RoBERTa, which has 125M parameters in its base model.
Below we can see more details about the number of parameters in different BART models. We can look at RoBERTa and its number of parameters on similar tasks for comparison.

Model	Description	# params
`bart.base`	BART model with 6 encoder and decoder layers	140M
`bart.large`	BART model with 12 encoder and decoder layers	400M
`bart.large.mnli`	`bart.large` finetuned on `MNLI`	400M
`bart.large.cnn`	`bart.large` finetuned on `CNN-DM`	400M
`bart.large.xsum`	`bart.large` finetuned on `Xsum`	400M

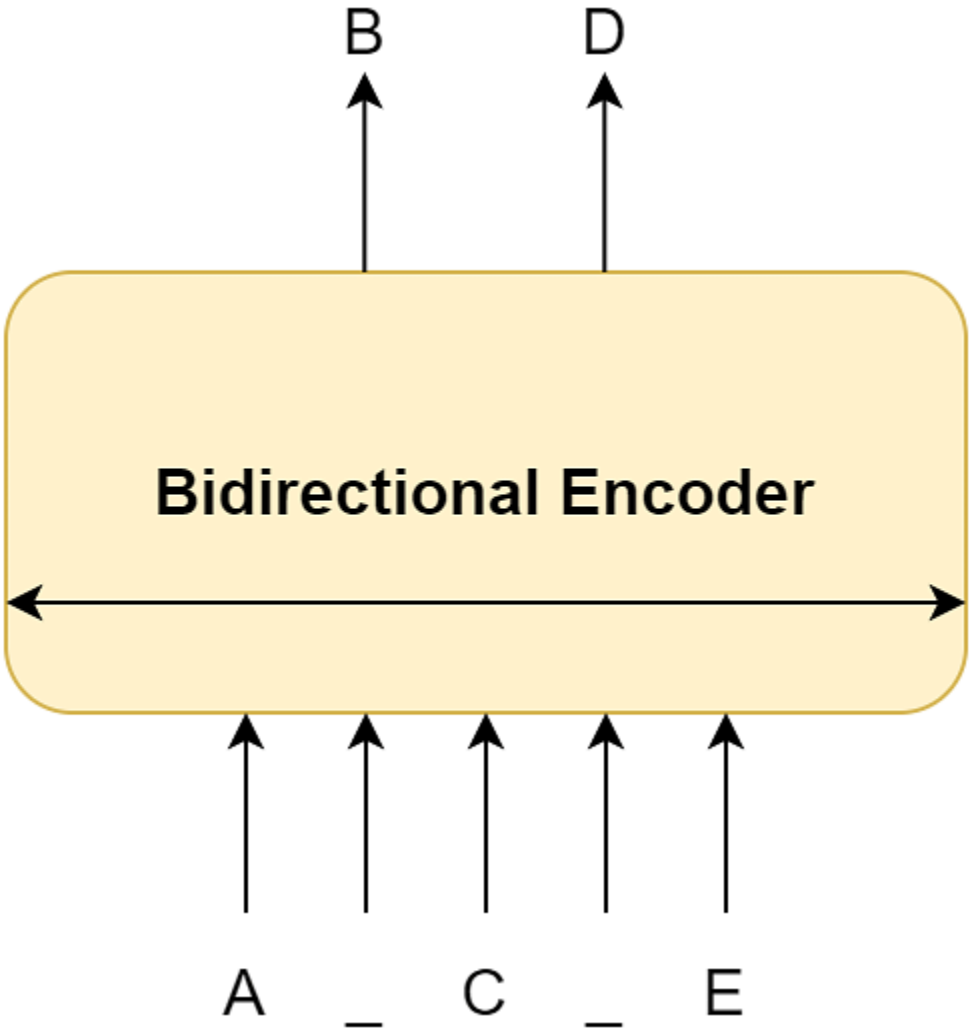
BART outperforms RoBERTa in several fine-tuning tasks, as detailed further in the paper. The number of parameters of the latter is as follows:

Model	Description	# params
`roberta.base`	RoBERTa using the BERT-base architecture	125M
`roberta.large`	RoBERTa using the BERT-large architecture	355M
`roberta.large.mnli`	`roberta.large` finetuned on MNLI	355M
`roberta.large.wsc`	`roberta.large` finetuned on WSC	355M

BART Architecture Explained

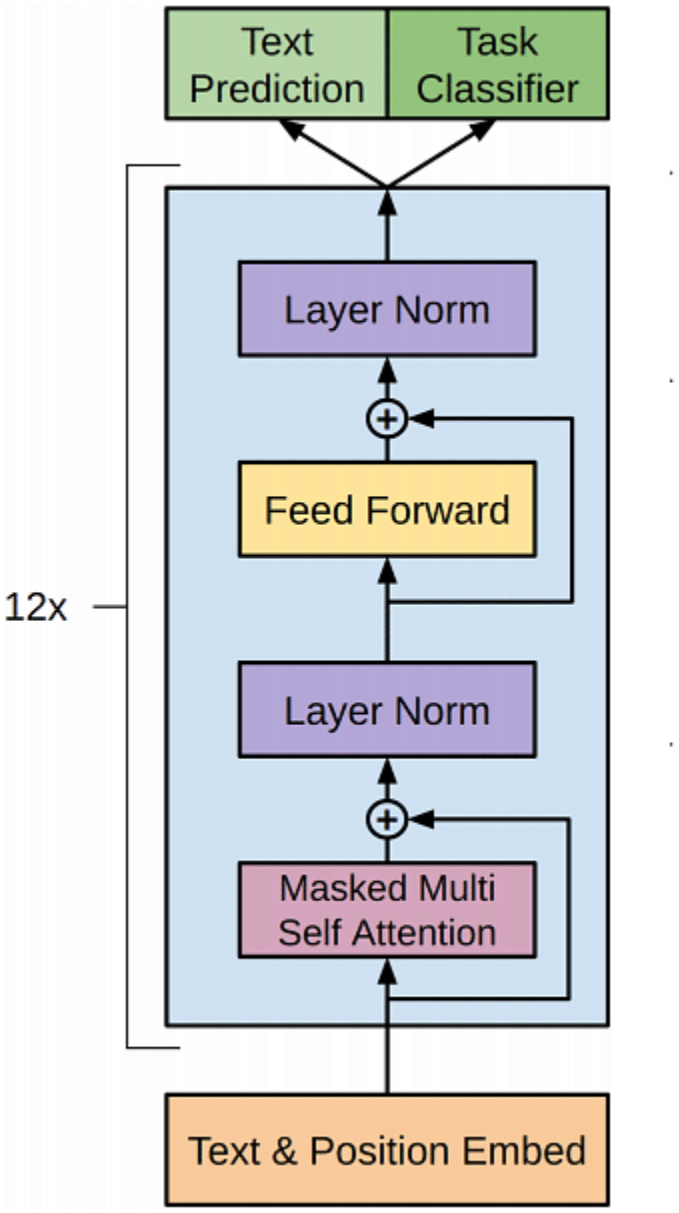
To understand the purpose and philosophy behind BART's architecture, first, we need to understand the nature of the NLP tasks it set out to solve. In tasks, such as question answering and text summarization, that require natural language understanding (or NLU) it is imperative for our model to read the text as a whole and understand each token in the context of what came, both, before and after it. For instance, training a masked language model with the sentence "the man went to the dairy store to buy a gallon of milk" could have a sentence like this as input based on the nosing schemes we saw above:
"the man went to the [MASK] store to buy a gallon of milk".
Now, for an NLU task, it is important for the model to read the sentence completely before predicting [MASK] since it highly depends on the terms like "store" and "milk". In such a case, the input sequence can be properly interpreted and learned by a bi-directional approach to reading and representing the text. The BERT (or Bidirectional Encoder Representations from Transformers) model incorporates this idea to greatly improve the language modeling task that happens in pre-training.
Thus, the first part of BART uses the bi-directional encoder of BERT to find the best representation of its input sequence. For every text sequence in its input, the BERT encoder outputs an embedding vector for each token in the sequence as well as an additional vector containing sentence-level

information. In this way, the decoder can learn for both token and sentence-level tasks making it a robust starting point for any future fine-tuning tasks.
The pre-training is done using the masked sequences as discussed previously and shown below. While BERT was trained by using a simple token masking technique, BART empowers the BERT encoder by using more challenging kinds of masking mechanisms in its pre-training.



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

autoregressive model that looks only at the past data to predict the future comes in handy.



The GPT-1 model used an architecture similar to the decoder segment of the vanilla Transformers. GPT sequentially stacks 12 such decoders such that learning from only the past tokens can affect the current token calculation. The architecture is shown above. As seen in the original Transformer decoder, the GPT decoder also uses a masked multiheaded self-attention block and a feed-forward layer.
Comparable to other models we discussed here, including BART, GPT also takes a semi-supervised approach to learning. First, the model is pre-

trained on tokens “t” looking back to “k” tokens in the past to compute the current token. This is done unsupervised on a vast text corpus to allow the model to “learn the language.”

$$L_1(T) = \sum_i log\, P(t_i|t_{i-k}, \ldots, t_{i-1}; \theta) \qquad (i)$$

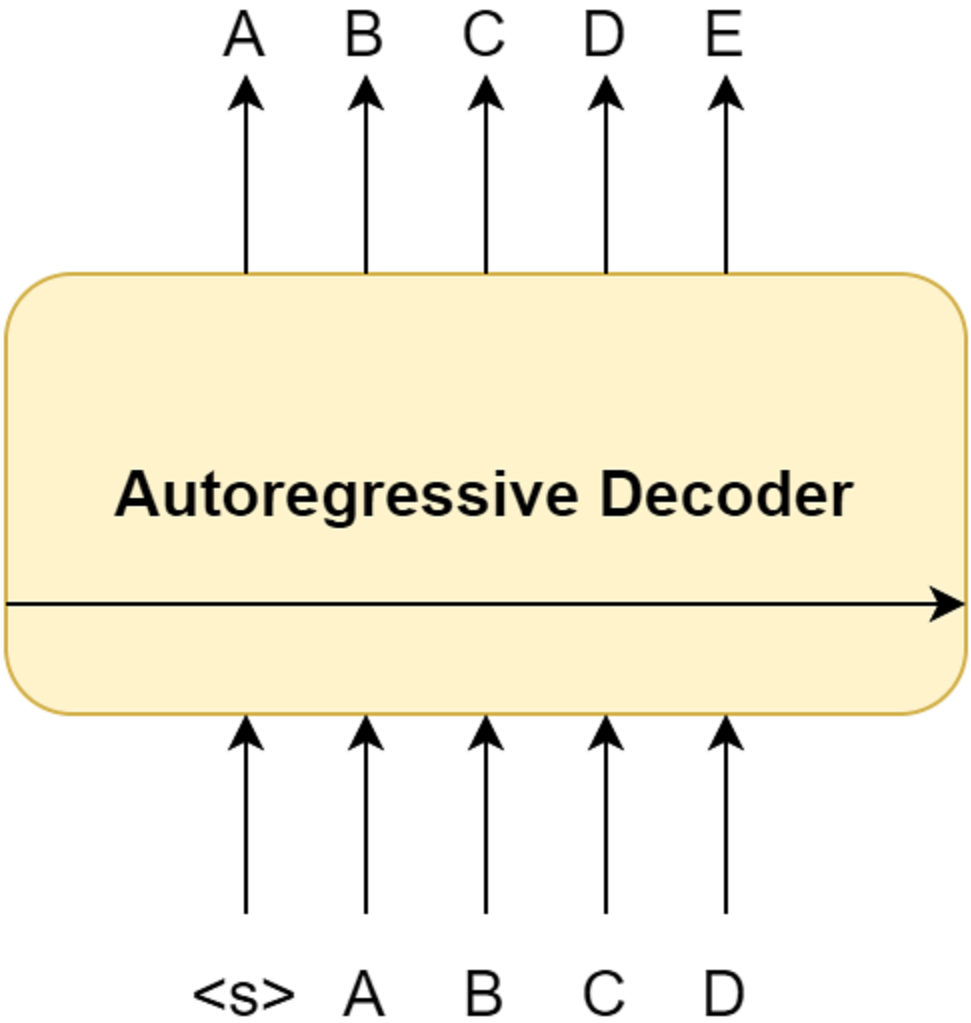
Next, to make the model robust on a specific task, it is fine-tuned in a supervised manner to maximize the likelihood of label “y” given feature vectors x1…xn.

$$L_2(C) = \sum_{x,y} log\, P(y|x_1, \ldots, x_n) \qquad (ii)$$

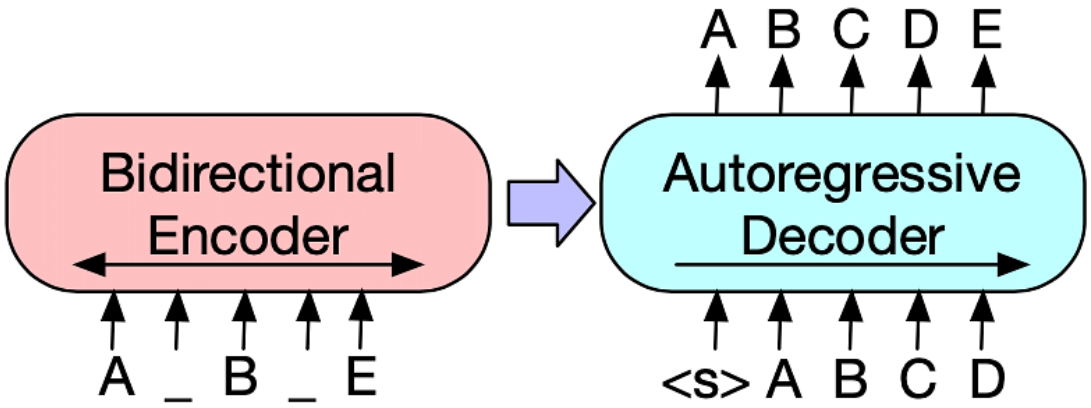
Combining 1 and 2, we get the objective in 3. Lambda represents a learned weight parameter to control the influence of language modeling.

$$L_3(C) = L_2(C) + \lambda L_1(C) \qquad (iii)$$

Below image shows how the autoregressive decoder processes its input.



Although we separate the decoder from an encoder, the input to the decoder would still be a learned representation (or embedding) of the original text sequence. Thus, BART attaches the bi-directional encoder to the autoregressive decoder to create a denoising auto-encoder architecture. And based on these two components, the final BART model would look something like this:



In the above figure, the input sequence is a masked (or noisy) version of [ABCDE] transformed into [A[MASK]B[MASK]E]. The encoder looks at the entire sequence and learns high-dimensional representations with bi-directional information. The decoder takes these thought vectors and regressively predicts the next token. Learning occurs by computing and optimizing the negative log-likelihood as mapped with the target [ABCDE].

BART Model for Text Summarization

As more and more long-form content burgeons on the internet, it is becoming increasingly time-consuming for someone like a researcher or journalist to filter out the content they want. Summaries or paraphrase synopsis help readers quickly go through the highlights of a huge amount of textual content and save enough time to study the relevant documents. Transformer models can automate this NLP task of text summarization. There are two approaches to achieve this: extractive and abstractive. Extractive summarization identifies and extracts the most significant statements from a given document as they are found in the text. This can be considered more of an information retrieval task. Abstractive summarization is more challenging as it aims to understand the entire document and generate paraphrased text to summarize the main points. Transformer models, including BART, perform the latter kind of summarization.

As we noted at the beginning of this article, HuggingFace provides access to both pre-trained and fine-tuned weights to thousands of Transformer models, BART being just one of them. For the text summarization task, you can choose fine-tuned BART models from the HuggingFace model explorer website. You can find the configuration and training description of every model uploaded there. Let's check the *bart-large-cnn* model for beginners. The model page: facebook/bart-large-cnn · Hugging Face also provides a hosted inference API instance for you to test text summarization on the text of your choice. For example, looking at the default text and running summarizer on it

CNN / Dailymail dataset

	R1	RL
Paper result	42.94	30.61
Our result	40.08	37.74

Pegasus:

PEGASUS model is a state-of-the-art language model developed by researchers at Google that is specifically designed for text summarization. The model is based on the Transformer architecture, which is a neural network slanguage processing tasks. It stands for **P**re-training **E**xtracted **G**ap-sentences for **A**bstractive **SU**mmarization **S**equence-to-sequence models.

What is Transformer architecture?

The main components of the Transformer architecture are:

1. Input Embedding: The input sequence is first embedded into a higher dimensional space through an embedding layer.
2. Positional Encoding: The positional encoding layer adds information about the position of each element in the sequence. This allows the model to differentiate between the different positions in the sequence.
3. Encoder: The encoder consists of multiple layers of self-attention and feed-forward neural networks. Each layer in the encoder processes the input sequence independently and passes it on to the next layer.
4. Decoder: The decoder is similar to the encoder, but it also has an additional attention mechanism that attends to the encoder output. The decoder is trained to generate an output sequence based on the input sequence and the encoder output.
5. Output Layer: The output layer takes the final representation of the decoder and generates the final output sequence.

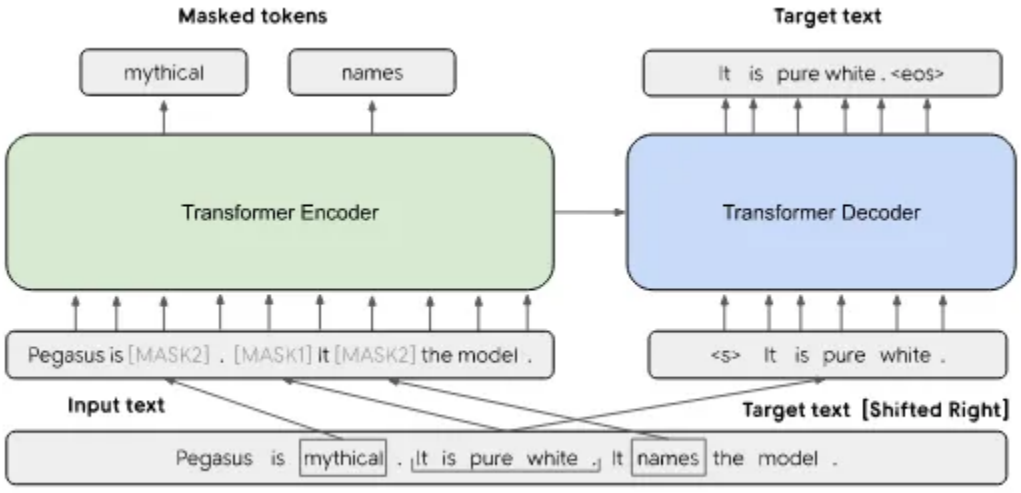


figure 1: The base architecture of PEGASUS is a standard Transformer encoder-decoder. Both GSG and MLM are applied simultaneously to this example as pre-training objectives. Originally there are three sentences. One sentence is masked with [MASK1] and used as target generation text (GSG). The other two sentences remain in the input, but some tokens are randomly masked by [MASK2] (MLM)

PEGASUS uses an encoder-decoder model for sequence-to-sequence learning. In such a model, the encoder will first take into consideration the context of the whole input text and encode the input text into something called context vector, which is basically a numerical representation of the input text. This numerical representation will then be fed to the decoder whose job is decode the context vector to produce the summary.

One of the main advantages of the PEGASUS model is that it can generate high-quality summaries of long documents with very little input. This is because the model is able to learn the underlying structure of the document and identify the most important information, which it then uses to generate a concise summary.

To achieve this, the PEGASUS model is trained using a technique called pre-training. During pre-training, the model is exposed to a large amount of text data, which it uses to learn the patterns and relationships between different words and phrases. This pre-training process enables the model to develop a deep understanding of the underlying structure of language, which in turn allows it to generate high-quality summaries.

What data was PEGASUS trained on?

In PEGASUS, important sentences are removed/masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary. We evaluated our best PEGASUS model on 12 downstream summarization tasks spanning news, science, stories, instructions, emails, patents, and legislative bills. Experiments demonstrate it achieves state-of-the-art performance on all 12 downstream datasets measured by ROUGE scores. Our model also shows surprising performance on low-resource summarization, surpassing previous state-of-the-art results on 6 datasets with only 1000 examples. Finally, we validated our results using human evaluation and

show that our model summaries achieve human performance on multiple datasets.

For pre-training we considered two large text corpora:

• C4, or the Colossal and Cleaned version of Common Crawl, introduced in (2019) consists of text from 350M Web pages (750GB).

• HugeNews, a dataset of 1.5B articles (3.8TB) collected from news and news-like websites from 2013- 2019. A whitelist of domains ranging from highquality news publishers to lower-quality sites such as high-school newspapers, and blogs was curated and used to seed a web-crawler. Heuristics were used to identify news-like articles, and only the main article text was extracted as plain text.

Downstream Tasks/Datasets For downstream summarization, we only used public abstractive summarization datasets, and access them through TensorFlow Summarization Datasets 1, which provides publicly reproducible code for dataset processing and train/validation/test splits. We used train/validation/test ratio of 80/10/10 if no split was provided, and 10% train split as validation if there was no validation split.

1. XSum consists of 227k BBC articles from 2010 to 2017 covering a wide variety of subjects along with professionally written single-sentence summaries.
2. CNN/DailyMail dataset contains 93k articles from the CNN, and 220k articles the Daily Mail newspapers. Both publishers supplement their articles with bullet point summaries. We use the non-anonymized variant.
3. NEWSROOM is a large dataset containing 1.3M article-summary pairs written by authors and editors in the newsrooms of 38 major publications between 1998 and 2017.
4. Multi-News is a multi-document summarization dataset consisting of 56k pairs of news articles and their human-written summaries from the site newser.com.
5. Gigaword contains 4M examples extracted from news articles (seven publishers) from the Gigaword corpus. The task is to generate the headline from the first sentence.

How many parameters does PEGASUS has?

Pegasus model released by Google has 568 million parameters. This makes it one of the largest pre-trained models for natural language processing tasks, and it requires a significant amount of computational resources to train and use. In addition to the original Pegasus model, Google has also released smaller versions of the model with fewer parameters. For example, the Pegasus-large model has 568 million parameters, while the Pegasus-small model has 124 million parameters. These smaller models are designed to be more efficient and easier to use, but they may not perform as well as the larger model on some tasks.

What is the topics PEGASUS good at?

here are some specific use cases where the PEGASUS model has been shown to outperform other summarization models:

1. Scientific Papers: PEGASUS has been shown to be highly effective at summarizing scientific papers. This is because scientific papers often contain complex technical jargon and require a deep understanding of the underlying concepts. PEGASUS is able to capture these nuances and generate accurate summaries that capture the key findings and insights of the paper.
2. News Articles: PEGASUS has also been shown to be highly effective at summarizing news articles. This is because news articles often contain a lot of important information that needs to be conveyed quickly and concisely. PEGASUS is able to identify the most important information and generate summaries that capture the essence of the story.
3. Legal Documents: PEGASUS has also been shown to be effective at summarizing legal documents such as contracts, briefs, and judgments. This is because legal documents often contain dense language and require a deep understanding of legal concepts. PEGASUS is able to identify the key legal arguments and generate summaries that capture the essence of the document.
4. Social Media Posts: PEGASUS has also been shown to be effective at summarizing social media posts such as tweets and Facebook updates. This is because social media posts often contain a lot of noise and extraneous information that needs to be filtered out. PEGASUS is able to identify the most important information and generate summaries that capture the essence of the post.

5. Long-form Content: PEGASUS has also been shown to be highly effective at summarizing long-form content such as books and long articles. This is because PEGASUS can summarize long-form content into a concise summary that captures the essence of the content, making it easier for readers to quickly grasp the main ideas and concepts.

It's important to keep in mind that the effectiveness of PEGASUS will depend on the specific use case and the quality of the training data. However, the PEGASUS model has demonstrated superior performance in many different types of summarization tasks, and it is likely to continue to be a useful tool for summarization across a wide range of applications.

What are the advantages and disadvantages of PEGASUS?

advantages:

High-quality summaries: The PEGASUS model is capable of generating high-quality summaries that are often more accurate and informative than those generated by other summarization models.

Multilingual capabilities: PEGASUS can generate summaries in multiple languages, making it a very versatile tool for summarization tasks that involve documents in different languages. These languages are English, French, German, Spanish, Portuguese, Italian, Dutch, Russian, Arabic, Chinese, and Japanese.

Customizability: The PEGASUS model can be fine-tuned to meet specific summarization tasks, which allows it to generate summaries that are tailored to the specific needs of the task.

Efficient: The PEGASUS model is highly efficient and can generate summaries of long documents with very little input, making it a valuable tool for large-scale summarization tasks.

Wide range of use cases: PEGASUS has been shown to be effective in summarizing a wide range of content types, including scientific papers, legal documents, news articles, and social media posts.

disadvantages:

Requires large amounts of training data: The PEGASUS model requires large amounts of training data to achieve optimal performance. This can be a significant challenge for organizations that lack the resources to collect and process large amounts of data.

Requires significant computational resources: The PEGASUS model is computationally intensive and requires significant computational resources to train and deploy. This can be a significant barrier to entry for organizations that lack the necessary hardware and infrastructure.

Limited interpretability: The PEGASUS model is a complex neural network model, which makes it difficult to interpret how it generates its summaries. This can make it challenging to identify and correct errors or biases in the summaries it generates.

May struggle with some types of content: While PEGASUS is highly effective at summarizing a wide range of content types, it may struggle with some types of content that are particularly complex or require a deep understanding of domain-specific knowledge.

Limited control over summaries: Because the PEGASUS model generates summaries autonomously, there is limited control over the content and tone of the summaries it generates. This can be a disadvantage in situations where specific messaging or tone is required.

Although PEGASUS has demonstrated superior performance in summarizing a wide range of content types, there are some content types that may pose challenges for the model.

Here is some of topics that Pegasus doesn’t perform well:

Creative Writing: PEGASUS may struggle with generating summaries of creative writing such as poetry, fiction, and other literary works that are characterized by their unique writing styles and creative use of language.

Sarcasm and Irony: PEGASUS may struggle with identifying and summarizing content that contains sarcasm, irony, or other forms of figurative language. This is because these forms of language often require a deep understanding of cultural and social contexts, which can be difficult for the model to capture.

Technical Documents: PEGASUS may struggle with summarizing highly technical documents such as scientific papers or engineering reports. These documents often contain complex terminology and require a deep understanding of domain-specific knowledge, which can be challenging for the model to capture.

Multimedia Content: PEGASUS may struggle with summarizing multimedia content such as videos or audio recordings. This is because these types of content often require a multimodal approach that takes into account both the audio and visual components of the content.

Highly Personalized Content: PEGASUS may struggle with summarizing highly personalized content such as emails or personal messages. This is because these types of content often contain a lot of noise and extraneous information that is specific to the individual, making it difficult for the model to identify the most important information and generate a concise summary.

	R1	RL
Paper result	41.79	38.39
Our result	38.82	36.66

T5:

-Brief overview of the T5 model

T5 (Text-to-Text Transfer Transformer) is a state-of-the-art language model developed by Google Brain that is based on the Transformer architecture. It was introduced in a paper published in 2019 by Colin Raffel et al. T5 is a versatile language model that can perform a wide range of natural language processing tasks, including text classification, question-answering, summarization, translation, and more.

The key innovation of T5 is its use of a "text-to-text" approach, where all tasks are cast as a text-to-text mapping problem. Specifically, T5 is trained on a large and diverse set of tasks that are represented as input-output pairs of text strings. During training, the model learns to generate the output text string that corresponds to a given input text string, based on the task at hand. This approach enables T5 to perform multiple tasks with a single model, by simply providing a different input-output pair for each task.

T5 is trained on a massive amount of data, using a combination of supervised and unsupervised learning. The model is pre-trained on a large corpus of text data using a variant of the Transformer architecture called the "BERT-style" pre-training, which involves masking some of the input tokens and predicting them based on the remaining tokens. T5 is then fine-tuned on specific tasks to improve its performance on those tasks

The Transformer architecture consists of two main components: the encoder and the decoder. The encoder takes an input sequence of tokens and produces a sequence of hidden states, while the decoder takes the encoder's output and generates a sequence of output tokens.

In T5, the input and output sequences are represented as text strings, and the model is trained to map one text string to another. Specifically, T5 uses a variant

of the Transformer architecture called the "Seq2Seq Transformer" or "T5 Transformer", which is designed for sequence-to-sequence tasks.

The T5 Transformer consists of a stack of identical layers, each of which has two sub-layers: a self-attention layer and a feedforward layer. The self-attention layer allows the model to attend to different parts of the input sequence when generating each output token, while the feedforward layer applies a non-linear transformation to the output of the self-attention layer.

Self-attention mechanism:

The self-attention mechanism is a key component of the Transformer architecture, and it is also used in T5. Self-attention allows the model to attend to different parts of the input sequence when generating each output token, based on the relevance of each input token to the output token.

In T5, the self-attention mechanism is applied in the encoder and decoder layers of the T5 Transformer. Specifically, each layer has a self-attention sub-layer that computes the attention weights for each input token, based on its similarity to all other input tokens.

The self-attention mechanism works by computing three matrices: the query matrix Q, the key matrix K, and the value matrix V. These matrices are derived from the input sequence, and they are used to compute the attention weights. The query matrix represents the tokens that are being attended to, while the key matrix and the value matrix represent the tokens that are used to compute the attention weights.

The attention weights are computed by taking the dot product of the query matrix and the transposed key matrix, and then applying a softmax function to obtain a probability distribution over the input tokens. The attention weights are then used to compute a weighted sum of the value matrix, which represents the final output of the self-attention mechanism.

In T5, the self-attention mechanism is enhanced with several modifications, including multi-head attention and residual connections. Multi-head attention allows the model to attend to different parts of the input sequence in parallel,

while residual connections enable the model to propagate information through multiple layers without vanishing gradients.

Pretraining

T5 is pre-trained using a combination of supervised and unsupervised learning approaches, with a focus on the unsupervised pre-training. During pre-training, T5 is trained on a large corpus of text data to learn general language representations that can be fine-tuned for specific downstream natural language processing tasks.

The pre-training approach used in T5 is based on a variant of the Transformer architecture called the "BERT-style" pre-training. This involves training the model to predict masked tokens in the input sequence, as well as predicting the next sentence in a pair of input sequences. The masked token prediction task involves randomly masking some of the input tokens and training the model to predict the original values of the masked tokens based on the remaining tokens in the input sequence. The next sentence prediction task involves training the model to predict whether a pair of input sequences are contiguous or not.

In addition to the masked token and next sentence prediction tasks, T5 also uses a denoising autoencoder task during pre-training. This involves adding noise to the input sequence and training the model to reconstruct the original sequence.

The supervised learning approach is used during the fine-tuning stage, where T5 is fine-tuned on specific downstream tasks using task-specific labeled data. During fine-tuning, a task-specific adapter layer is added to the pre-trained T5 model, and the adapter layer is trained on the labeled data while keeping the majority of the pre-trained parameters fixed.

The combination of unsupervised and supervised learning approaches used in T5 enables the model to learn general language representations that can be fine-tuned for a wide range of natural language processing tasks, while also leveraging task-specific labeled data to improve performance on those tasks.

Pre-training with a denoising autoencoder

T5 also employs a denoising autoencoder task during pre-training. The denoising autoencoder task involves adding noise to the input sequence and training the model to reconstruct the original sequence.

The denoising autoencoder task is designed to help T5 learn robust representations that can handle noisy or incomplete inputs. During pre-training, the input sequence is corrupted by randomly replacing some of the tokens with a special "mask" token or a random token. The model is then trained to reconstruct the original input sequence based on the corrupted sequence.

The denoising autoencoder task is implemented as follows in T5:

1. The input sequence is randomly corrupted by replacing some of the tokens with a "mask" token or a random token.

2. The corrupted sequence is fed into the model, which generates a sequence of hidden states.

3. The hidden states are then passed through a decoder, which generates a sequence of output tokens.

4. The output tokens are compared to the original input sequence, and the model is trained to minimize the reconstruction loss.

The denoising autoencoder task helps T5 learn more robust representations by forcing the model to learn to reconstruct the original input sequence from noisy or incomplete inputs. This can be especially useful for downstream tasks where the input data may be noisy or contain missing information.

Large corpus of text data

T5 is pre-trained on a large and diverse corpus of text data to learn general language representations. The corpus of text data used for pre-training T5 is massive and includes a variety of sources, such as web pages, books, and Wikipedia.

The exact size of the corpus used for pre-training T5 is not publicly disclosed, but it is likely to be in the range of hundreds of billions of tokens or even trillions of tokens, based on similar pre-training approaches for other large language models.

The large corpus of text data used for pre-training T5 is essential for enabling the model to learn a broad range of language patterns and structures. By training on a diverse set of text data, T5 can learn to capture the nuances and complexities of natural language, as well as the variations in language use across different domains and genres.

The pre-training corpus is also carefully filtered to remove low-quality data and ensure that the data is suitable for pre-training a high-quality language model. For example, the corpus may be filtered to remove duplicate or spammy content, as well as content that is not relevant for language modeling.

Overall, the use of a large and diverse corpus of text data is a key factor in the success of T5 as a general-purpose language model that can be fine-tuned for a wide range of natural language processing tasks.

Performance on different tasks

there are some examples of its performance on some benchmark datasets:

1. GLUE benchmark: T5 has achieved state-of-the-art performance on the General Language Understanding Evaluation (GLUE) benchmark, which consists of nine natural language understanding tasks, including text classification, sentence similarity, and question-answering. T5 has achieved a score of 90.4 on the benchmark, which is significantly higher than the previous best score of 89.4.

2. SuperGLUE benchmark: T5 has also achieved state-of-the-art performance on the SuperGLUE benchmark, which consists of eight challenging natural language understanding tasks, including Winograd Schema Challenge, and Argument Reasoning Comprehension Task. T5 achieved a score of 89.9 on the benchmark, which is significantly higher than the previous best score of 87.6.

3. Machine translation: T5 has achieved state-of-the-art performance on several machine translation tasks, including the WMT19 English-German and English-French translation tasks. T5 achieved a BLEU score of 29.8 on the English-German task and a BLEU score of 43.5 on the English-French task, which are both significantly higher than the previous best scores.

4. Summarization: T5 has achieved state-of-the-art performance on several summarization tasks, including the CNN/Daily Mail news article summarization task. T5 achieved a ROUGE-L score of 44.32 on the task, which is significantly higher than the previous best score of 43.13.

Comparison *of t5 model with other models:*

T5 has achieved high performance on a wide range of natural language processing tasks, and it has several advantages over other large language models. Here are some comparisons between T5 and other models:

1. BERT: T5 is based on the same Transformer architecture as BERT, and both models use a similar pre-training approach. However, T5 is designed to perform multiple tasks with a single model, while BERT is typically fine-tuned for a specific task. T5 has also achieved better performance than BERT on several benchmark datasets, including the GLUE and SuperGLUE benchmarks.

2. GPT-2 and GPT-3: T5 and GPT-2/GPT-3 are both large language models that are based on the Transformer architecture. However, T5 is designed to perform multiple tasks with a single model, while GPT-2/GPT-3 are primarily designed for language generation tasks. T5 has also achieved better performance than GPT-2/GPT-3 on several benchmark datasets, including the GLUE and SuperGLUE benchmarks.

3. XLNet: XLNet is another large language model that is based on the Transformer architecture, and it uses a permutation-based pre-training approach that allows it to capture bidirectional context. T5 also uses a bidirectional pre-training approach, but it is designed to perform multiple tasks with a single model. T5 has achieved similar or better performance than XLNet on several benchmark datasets, including the GLUE and SuperGLUE benchmarks.

Overall, T5 has demonstrated impressive performance on a wide range of natural language processing tasks, and its ability to perform multiple tasks with a single model makes it a valuable tool for natural language processing researchers and practitioners. While there are other large language models that have also achieved state-of-the-art performance, T5's unique architecture and pre-training approach give it several advantages over other models.

Limitations and future work:

While T5 has achieved impressive performance on a wide range of natural language processing tasks, there are still some limitations and areas for future work. Here are a few examples:

1. Efficiency: T5 is a very large model with hundreds of millions of parameters, which can make it computationally expensive and slow to train and deploy. There is ongoing research into making large language models more efficient and lightweight, such as through model compression techniques or more efficient architectures.

2. Multilingualism: While T5 has shown impressive performance on English-language tasks, its performance on non-English languages is still relatively limited. There is ongoing research into developing multilingual language models that can perform well on a wide range of languages, and it will be interesting to see how T5 can be extended to support multilingualism.

3. Explainability: Like other large language models, T5 can be difficult to interpret, and it may not always be clear how the model is making its predictions. There is ongoing research into developing techniques for interpreting and visualizing the representations learned by large language models, which could help improve the interpretability and transparency of models like T5.

4. Common Sense Reasoning: While T5 has achieved impressive performance on a wide range of natural language processing tasks, it still struggles with tasks that require common sense reasoning or world knowledge. There is ongoing research into developing models that can reason about the world and understand the context of language, which could help improve the performance of models like T5 on more challenging tasks.

Overall, T5 has shown great promise as a general-purpose language model, but there are still several areas for improvement and ongoing research. As the field of natural language processing continues to evolve, it will be interesting to see how

T5 and other large language models are adapted and extended to tackle new challenges and applications.

CNN / Dailymail dataset

	R1	RL
Paper result	43.52	40.69
Our result	39.32	37.54

Natural Language Generation

T5 has a wide range of potential applications in natural language processing that make it a valuable tool for researchers and practitioners in the field. Its ability to perform multiple tasks with a single model also makes it an attractive option for applications where efficiency and simplicity are important.

Here are a few examples:

1. Text generation: T5 can be used for generating text in a variety of contexts, such as chatbots, language translation, summarization, and content creation. T5's ability to perform multiple tasks with a single model makes it a versatile tool for generating different types of text.

2. Information retrieval: T5 can be used for tasks such as question answering, information retrieval, and document classification. T5's strong performance on benchmark datasets such as the GLUE and SuperGLUE benchmarks makes it a powerful tool for these tasks.

3. Natural language understanding: T5 can be used for tasks such as sentiment analysis, entity recognition, and semantic parsing. T5's ability to learn general language representations through pre-training makes it well-suited for these tasks.

4. Conversational agents: T5 can be used to develop conversational agents that can understand and generate natural language responses. T5's ability to perform multiple tasks with a single model makes it a useful tool for developing conversational agents that can perform a wide range of tasks.

5. Data augmentation: T5 can be used to generate synthetic data for training machine learning models. This can be especially useful for tasks where labeled data is scarce or expensive to obtain

T5 base

T5 Base is a variant of the T5 model architecture developed by Google's Brain team. It is the smallest and least computationally expensive variant of the T5 model family, with 220 million parameters. Despite its smaller size, T5 Base is still a powerful language model that can perform a wide range of natural language processing tasks.

T5 Base has achieved strong performance on several benchmark natural language processing datasets, including the GLUE benchmark and the SuperGLUE benchmark. It has also been used in a variety of applications, such as language translation, question answering, and text classification.

One advantage of T5 Base is that it is more computationally efficient than larger T5 variants, such as T5 Large or T5 3B. This makes it a more practical option for applications where computational resources are limited. However, it is important to note that T5 Base may not perform as well as larger models on more complex or challenging tasks.

T5 small

T5 Small is a variant of the T5 model architecture that has fewer parameters than other T5 variants, making it smaller and less computationally expensive to train and deploy. Specifically, T5 Small has 60 million parameters, which is significantly smaller than T5 Base (220 million parameters), T5 Large (770 million parameters), and T5 3B (3 billion parameters).

Despite its smaller size, T5 Small is still a powerful language model that can perform a wide range of natural language processing tasks. It is based on the Transformer architecture and is pre-trained on a large corpus of text data using a combination of unsupervised and supervised learning. The pre-training process is designed to enable the model to learn general language representations that can be fine-tuned for specific downstream tasks.

T5 Small has achieved strong performance on several benchmark natural language processing datasets, including the GLUE benchmark and the SuperGLUE benchmark. It has also been used in a variety of applications, such as language translation, question answering, and text classification.

One advantage of T5 Small is that it is even more computationally efficient than larger T5 variants such as T5 Base, which makes it a practical option for applications where computational resources are limited. However, it is important

to note that T5 Small may not perform as well as larger models on more complex or challenging tasks.

T5 11B

T5 11B is the largest variant of the T5 model architecture, with 11 billion parameters. It is currently one of the largest language models ever developed and is capable of performing a wide range of natural language processing tasks with state-of-the-art performance.

Like other T5 variants, T5 11B is based on the Transformer architecture and is pre-trained on a large corpus of text data using a combination of unsupervised and supervised learning. The pre-training process is designed to enable the model to learn general language representations that can be fine-tuned for specific downstream tasks.

T5 11B has achieved impressive performance on several benchmark natural language processing datasets, including the GLUE benchmark and the SuperGLUE benchmark. It has also been used in a variety of applications, such as language translation, question answering, and text classification.

One advantage of T5 11B is its large size, which enables it to capture more complex and nuanced relationships between words and phrases in natural language. This can make it a more powerful tool for tasks that require a deep understanding of language, such as sentiment analysis or language modeling.

However, one major drawback of T5 11B is its computational requirements. Training and deploying such a large model is extremely resource-intensive, and may not be practical for many applications. Additionally, the large size of the model can make it more difficult to interpret and understand how it is making its predictions.

Computational implementation requirements of T5

The computational implementation requirements of T5 depend on the specific task being performed and the available hardware resources. Here are some general requirements for implementing T5:

1. Hardware: T5 models are computationally expensive and require significant hardware resources, particularly for training. The specific hardware requirements depend on the size of the model, the size of the dataset, and the available hardware resources. For training T5 models, high-performance GPUs or TPUs are recommended.

2. Software: To implement T5, you will need a software environment that supports deep learning frameworks such as PyTorch or TensorFlow, as well as the Hugging Face Transformers library or the Google T5 library for loading and fine-tuning the T5 model.

3. Data: Depending on the task, you will need to gather or create a dataset that is appropriate for fine-tuning the T5 model. The size and complexity of the dataset will affect the computational requirements for training the model.

4. Training time: Training a T5 model can take a significant amount of time, particularly for larger models or datasets. Training time can be reduced by using transfer learning, which involves fine-tuning a pre-trained T5 model on a specific task.

5. Memory: T5 models require significant amounts of memory, particularly during training. To train a T5 model, you will need sufficient GPU or TPU memory to store the model and the data being used for training.

6. Optimization: To optimize the performance of T5 models, it is important to use techniques such as mixed-precision training, which can reduce the amount of memory required by the model, and gradient accumulation, which can reduce the amount of memory required during training.

Overall, implementing T5 requires significant computational resources and expertise in deep learning and natural language processing. However, with the

right hardware, software, and data, T5 can be a powerful tool for natural language processing tasks.

Summary of the T5 model and its application to the CNN Daily News dataset

The T5 model is a powerful language model that is capable of performing a wide range of natural language processing tasks, including language translation, question answering, and text classification. It is based on the Transformer architecture and is pre-trained on a large corpus of text data using a combination of unsupervised and supervised learning.

To apply the T5 model to the CNN Daily News dataset, the dataset would first need to be preprocessed and formatted in a way that is compatible with the T5 model. This may involve tasks such as data cleaning, tokenization, and formatting. Once the data is prepared, the T5 model can be fine-tuned on the dataset using a supervised learning approach.

The CNN Daily News dataset is a collection of news articles from CNN and can be used for a variety of natural language processing tasks, such as text classification or language modeling. By fine-tuning the T5 model on this dataset, it may be possible to achieve state-of-the-art performance on these tasks.

Overall, the T5 model represents a powerful tool for natural language processing, and its application to the CNN Daily News dataset could have a wide range of applications in areas such as news analysis, sentiment analysis, and language modeling. However, implementing the T5 model requires significant computational resources and expertise in natural language processing and machine learning.

Implications for future research of T5

Here are some potential avenues for future research related to T5:

1. Larger models: The T5 11B model is currently the largest T5 variant, but there is potential for even larger models with even more parameters. Developing and training larger language models could lead to further improvements in natural language processing tasks, but would also require significant computational resources.

2. Better pre-training: The T5 model is pre-trained using a combination of unsupervised and supervised learning, but there is potential for further improvements in pre-training techniques. One area of research could be developing new pre-training objectives that are better suited for specific tasks or domains.

3. Fine-tuning strategies: Fine-tuning is an important part of using the T5 model for specific tasks, but there is still much to learn about the best fine-tuning strategies for different types of tasks and datasets. Future research could explore different fine-tuning approaches and their impact on model performance.

4. Interpretability: As language models like T5 become more complex and powerful, it becomes increasingly important to understand how they are making their predictions. Future research could explore methods for interpretability and explaining the reasoning behind T5's predictions.

5.Multimodal integration: The T5 model is designed for text-based natural language processing tasks, but there is potential for integrating other modalities such as images or speech. Future research could explore methods for integrating multiple modalities into T5 or developing new models that can handle multiple modalities.

3.2 Comparison of the Strengths and Weaknesses:

Model	Strengths	Weaknesses
BART	BART is a pre-trained sequence-to-sequence model that combines autoencoding and autoregressive objectives to improve its performance in tasks such as text generation and summarization, BART is able to handle noisy input, making it a good choice for tasks that involve noisy or unstructured data, BART has high accuracy in text generation tasks, making it a popular choice for applications such as summarization and language generation BART is able to perform well in low-resource settings, making it a good choice for applications where training data is limited.	BART has high computational requirements, which can make it difficult to train and use on low-end hardware, BART's performance in tasks that require domain-specific knowledge is limited, as it relies solely on its pre-training to generate output.
T5	T5 is a pre-trained sequence-to-sequence model that uses a transformer architecture to perform various NLP tasks such as language translation, summarization, and question answering, T5 has high accuracy in language translation tasks, making it a popular choice for applications that involve cross-language communication, T5 is able to handle long input sequences, making it a good choice for tasks that involve lengthy or complex input. T5's flexibility and ability to perform multiple tasks with a single model make it a popular choice among NLP researchers and practitioners.	T5 has high computational requirements, which can make it difficult to train and use on low-end hardware, T5's performance in tasks that require domain-specific knowledge is limited, as it relies solely on its pre-training to generate output.
PEGASUS	- PEGASUS is a pre-trained sequence-to-sequence model that uses a transformer architecture and a self-supervised objective to improve its performance in various NLP tasks such as text generation and summarization. has high accuracy in summarization tasks, making it a popular choice for applications that require summarization of lengthy documents. PEGASUS is able to generate coherent and readable text, making it a good choice for tasks that involve natural language generation. PEGASUS is able to perform well in low-resource settings, making it a good choice for applications where training data is limited.	- PEGASUS has limited performance in language translation tasks, making it less suitable for applications that involve cross-language communication, PEGASUS has high computational requirements, which can make it difficult to train and use on low

3.3 Applications of Each Model in Various NLP Tasks:

BART, T5, and PEGASUS are three of the most popular NLP models used in various natural language processing tasks. These models have shown significant improvements in the field of NLP and have demonstrated their effectiveness in various applications. In this article, we will discuss some of the applications of these models in more detail.

BART:
BART is a pre-trained sequence-to-sequence model that uses a combination of autoencoding and autoregressive objectives to improve its performance in tasks such as text generation and summarization. BART has shown impressive results in various applications, including:

1. Text Summarization: BART has been used for automatic summarization of news articles, scientific papers, and other lengthy documents. It has demonstrated high accuracy in summarizing important information and generating concise summaries that capture the essence of the original text.
2. Chatbots: BART has been used as a natural language generation model for chatbots, where it generates responses to user queries. It has shown high accuracy in generating natural-sounding responses that mimic human conversation.
3. Language Generation: BART has been used for various language generation tasks, such as generating headlines, captions, and product descriptions for e-commerce websites. It has demonstrated the ability to generate coherent and readable text that accurately describes the content.
4. Text Completion: BART has been used for text completion tasks, such as filling in missing words or phrases in a text. It has shown high accuracy in completing text in a way that is consistent with the context and the style of the original text.

T5:

T5 is a pre-trained sequence-to-sequence model that uses a transformer architecture to perform various NLP tasks such as language translation, summarization, and question answering. T5 has shown impressive results in various applications, including:

1. Language Translation: T5 has been used for cross-language communication, where it translates text from one language to another. It has demonstrated high accuracy in translating text and is one of the best-performing models in this domain.
2. Summarization: T5 has been used for automatic summarization of news articles, scientific papers, and other lengthy documents. It has demonstrated high accuracy in summarizing important information and generating concise summaries that capture the essence of the original text.
3. Question Answering: T5 has been used for question answering tasks, where it generates answers to user queries based on a given context. It has shown high accuracy in generating accurate and informative answers that are consistent with the context.
4. Text Generation: T5 has been used for various language generation tasks, such as generating captions for images or videos. It has demonstrated the ability to generate coherent and readable text that accurately describes the content.

PEGASUS:

PEGASUS is a pre-trained sequence-to-sequence model that uses a transformer architecture and a self-supervised objective to improve its performance in various NLP tasks such as text generation and summarization. PEGASUS has shown impressive results in various applications, including:

1. Text Summarization: PEGASUS is particularly effective in summarization tasks, such as summarizing long documents or news articles. It has demonstrated high accuracy in summarizing important information and generating concise summaries that capture the essence of the original text.
2. Language Generation: PEGASUS has been used for various language generation tasks, such as generating product descriptions or news headlines. It has demonstrated the ability to generate coherent and readable text that accurately describes the content.
3. Text Paraphrasing: PEGASUS has been used for text paraphrasing tasks, where it generates alternative versions of a given text. It has shown high accuracy in generating paraphrases that are consistent with the meaning and style of the original text.
4. Text Completion: PEGASUS has been used for text completion tasks, such as filling in missing words or phrases in a text. It has shown high accuracy in completing text in a way that is consistent with the context and the style of the original text.

BART, T5, and PEGASUS are three of the most popular NLP models used in various natural language processing tasks. These models have shown impressive results in various applications such as text generation, summarization, language translation, and question answering. Understanding the strengths and weaknesses of each model can help researchers and practitioners choose the best model for their specific use case.

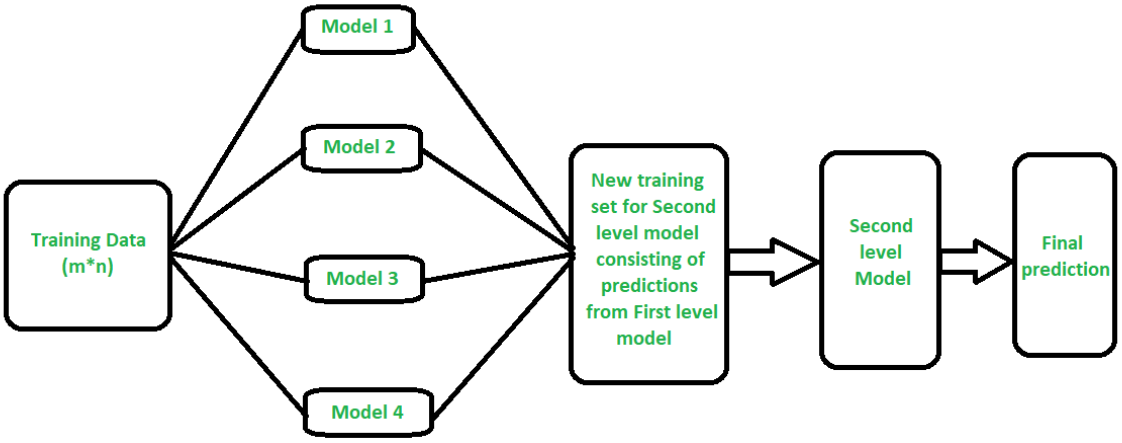
4:Stacking Ensemble in NLP

4.1 Concept of Stacking Ensemble in NLP:

Architecture of Stacking

The architecture of the stacking model is designed in such as way that it consists of two or more base/learner's models and a meta-model that combines the predictions of the base models. These base models are called level 0 models, and the meta-model is known as the level 1 model. So, the Stacking ensemble method includes **original (training) data, primary level models, primary level prediction, secondary level model, and final prediction**. The basic architecture of stacking can be represented as shown below the image.

- **Original data:** This data is divided into n-folds and is also considered test data or training data.
- **Base models:** These models are also referred to as level-0 models. These models use training data and provide compiled predictions (level-0) as an output.
- **Level-0 Predictions:** Each base model is triggered on some training data and provides different predictions, which are known as **level-0 predictions.**
- **Meta Model:** The architecture of the stacking model consists of one meta-model, which helps to best combine the predictions of the base models. The meta-model is also known as the **level-1 model**.
- **Level-1 Prediction:** The meta-model learns how to best combine the predictions of the base models and is trained on different predictions made by individual base models, i.e., data not used to train the base models are fed to the meta-model, predictions are made, and these predictions, along with the expected outputs, provide the input and output pairs of the training dataset used to fit the meta-model.



Steps to implement Stacking models:

There are some important steps to implementing stacking models in machine learning. These are as follows:

- Split training data sets into n-folds using the **RepeatedStratifiedKFold** as this is the most common approach to preparing training datasets for meta-models.
- Now the base model is fitted with the first fold, which is n-1, and it will make predictions for the nth folds.
- The prediction made in the above step is added to the x1_train list.
- Repeat steps 2 & 3 for remaining n-1folds, so it will give x1_train array of size n,
- Now, the model is trained on all the n parts, which will make predictions for the sample data.
- Add this prediction to the y1_test list.
- In the same way, we can find x2_train, y2_test, x3_train, and y3_test by using Model 2 and 3 for training, respectively, to get Level 2 predictions.
- Now train the Meta model on level 1 prediction, where these predictions will be used as features for the model.

- Finally, Meta learners can now be used to make a prediction on test data in the stacking model.

Summary of Stacking Ensemble

Stacking is an ensemble method that enables the model to learn how to use combine predictions given by learner models with meta-models and prepare a final model with accurate prediction. The main benefit of stacking ensemble is that it can shield the capabilities of a range of well-performing models to solve classification and regression problems. Further, it helps to prepare a better model having better predictions than all individual models. In this topic, we have learned various ensemble techniques and their definitions, the stacking ensemble method, the architecture of stacking models, and steps to implement stacking models in machine learning.

4.2 Benefits and Drawbacks of Stacking Ensemble:

Stacking ensemble is a machine learning technique that combines the predictions of multiple base models to improve the accuracy of the final prediction. In this technique, a meta-model is trained on the predictions of the base models to generate the final prediction. Stacking ensemble has gained popularity in recent years due to its ability to improve the predictive power of machine learning models. In this article, we will discuss the benefits and drawbacks of stacking ensemble in more detail.

Benefits of Stacking Ensemble:

1. Improved Predictive Accuracy: One of the primary benefits of stacking ensemble is its ability to improve the predictive accuracy of machine learning models. By combining the predictions of multiple base models, stacking ensemble can reduce the errors and biases of individual models, resulting in a more accurate final prediction.
2. Flexibility: Stacking ensemble is a flexible technique that can be applied to any machine learning model, regardless of its type or complexity. This makes it a useful tool for improving the accuracy of a wide range of machine learning models.
3. Robustness: Stacking ensemble is a robust technique that can handle noisy or incomplete data. By combining the predictions of multiple base models, stacking ensemble can reduce the impact of outliers and errors in the data, resulting in a more robust final prediction.
4. Better Generalization: Stacking ensemble can improve the generalization of machine learning models by reducing overfitting. By combining the predictions of multiple base models, stacking ensemble can capture the underlying patterns in the data without overfitting to the training data.
5. Better Interpretability: Stacking ensemble can improve the interpretability of machine learning models. By using multiple base models, stacking

ensemble can provide more insight into the underlying patterns in the data, making it easier to understand the factors that contribute to the final prediction.

Drawbacks of Stacking Ensemble:

1. Increased Complexity: One of the primary drawbacks of stacking ensemble is its increased complexity. Stacking ensemble requires the training of multiple base models and a meta-model, which can be computationally expensive and time-consuming.
2. Overfitting: Stacking ensemble can lead to overfitting if the base models are too complex or if the meta-model is not properly regularized. Overfitting can result in a final prediction that is overly tailored to the training data and performs poorly on new, unseen data.
3. Difficulty in Implementation: Implementing stacking ensemble can be challenging, especially for beginners in machine learning. Properly selecting the base models and the meta-model, tuning hyperparameters, and ensuring proper regularization can be difficult, requiring a deep understanding of machine learning techniques.
4. Increased Risk of Model Failure: Stacking ensemble increases the risk of model failure due to the complex nature of the technique. If any of the base models or the meta-model is poorly designed or tuned, the entire ensemble can fail to provide accurate predictions.
5. Increased Data Requirements: Stacking ensemble requires a larger amount of data than individual machine learning models. This can be a challenge for data-poor applications, where obtaining large amounts of data may be difficult or prohibitively expensive.

stacking ensemble is a powerful technique that can improve the accuracy of machine learning models by combining the predictions of multiple base models. Stacking ensemble provides a flexible, robust, and interpretable framework for improving the generalization and predictive power of machine learning models. However, it also has certain drawbacks, including increased complexity, increased risk of model failure, and increased data requirements

4.3 Implement Stacking Ensemble:

Implementing stacking ensemble requires several steps that involve selecting appropriate base models, training them on the data, and then using their predictions to train a meta-model. Here are the steps involved in implementing stacking ensemble:

Step 1: Selecting Base Models
The first step in implementing stacking ensemble is to select the appropriate base models. These models should be diverse and capture different aspects of the data. For example, one could select a decision tree, a random forest, and a support vector machine as base models. It is important to ensure that the base models are not too complex, as this can lead to overfitting.

Step 2: Training Base Models
Once the base models have been selected, they need to be trained on the data using a training set. This involves selecting appropriate hyperparameters for each model and training them on the data. It is important to avoid overfitting during this phase, as this can negatively impact the performance of the stacking ensemble.

Step 3: Generating Base Model Predictions
After the base models have been trained, they can be used to generate predictions for the training set and the test set. These predictions are used as input for the meta-model, which is trained on the predictions.

Step 4: Training the Meta-Model
The meta-model is trained on the predictions generated by the base models. This involves selecting an appropriate algorithm, such as logistic regression or a neural network, and training it on the predictions. The meta-model should be chosen to optimize the performance of the stacking ensemble on the test set.

Step 5: Evaluating the Stacking Ensemble
The final step in implementing stacking ensemble is to evaluate its performance on the test set. This involves generating predictions for the test set using the base models and the meta-model, and then evaluating the accuracy of the predictions.

It is important to note that implementing stacking ensemble can be computationally expensive and time-consuming, particularly if the base models are complex or the data set is large. Additionally, selecting appropriate hyperparameters for the base models and the meta-model can be challenging and may require extensive trial and error. However, with careful implementation and tuning, stacking ensemble can improve the accuracy of machine learning models and provide a robust and interpretable framework for data analysis.

4.4 Stacking Ensemble in NLP tasks:

One of the main advantages of stacking ensemble in NLP tasks is its ability to combine the strengths of multiple models. For example, a decision tree model may be good at capturing the structure of the data, while a support vector machine may be better at handling complex relationships between variables. By combining the predictions of these models, stacking ensemble can improve the accuracy of the final prediction.

In sentiment analysis, for instance, a common approach is to use a bag-of-words model to represent the text and train a machine learning model on the word frequencies. However, this approach may not capture certain nuances in the text, such as sarcasm or irony. By using stacking ensemble, additional models can be added to the pipeline to capture these nuances and improve the accuracy of the sentiment analysis.

In text classification, stacking ensemble can be used to combine the predictions of multiple models, each trained on a different aspect of the text. For example, one model may be trained on the syntax of the text, while another may be trained on the semantic content. By combining these models through stacking ensemble, the accuracy of the final classification can be improved.

In language translation, stacking ensemble can be used to improve the accuracy of the translation by combining the predictions of multiple models. Each of these models may focus on different aspects of the translation, such as syntax, semantics, or word choice. By combining the predictions of these models, the accuracy of the final translation can be improved.

Stacking ensemble is a powerful technique that can be applied to a wide range of NLP tasks to improve the accuracy of machine learning models. By combining the strengths of multiple models, stacking ensemble can capture the underlying patterns in the data and provide a more accurate prediction. While implementing stacking ensemble can be challenging, with careful implementation and tuning, it can provide a robust and interpretable framework for data analysis in NLP tasks.

5: Combining Models using Stacking Ensemble

We have initialized three different pre-trained models, each with their own tokenizer. bart_tokenizer and pegasus_tokenizer are both pre-trained on the CNN/Daily Mail dataset for summarization tasks, while T5_tokenizer is pre-trained on a diverse range of tasks. These models can be used to perform various NLP tasks such as text summarization, text generation, and more.

Once you have initialized these models and tokenizers, you can use them to preprocess input text, generate summaries, or perform other NLP tasks as required.

```
# Initialize the Bart tokenizer and model
bart_tokenizer = AutoTokenizer.from_pretrained('facebook/bart-large-cnn')

# Initialize the pegasus tokenizer and model
pegasus_tokenizer = AutoTokenizer.from_pretrained("google/pegasus-cnn_dailymail")

# Initialize the pegasus tokenizer and model
T5_tokenizer = AutoTokenizer.from_pretrained("t5-large")
```

We load the data

```
df_bart_train= pd.read_csv('/kaggle/input/bart-and-pegasus-full-train/Bart_Train_Summaries(0-11490).csv')
df_pegasus_train=pd.read_csv('/kaggle/input/bart-and-pegasus-full-train/Pegasus_Summaries_Train(0-11490).csv')
df_t5_train=pd.read_csv('/kaggle/input/bart-and-pegasus-full-train/T5_TRAIN_SUMMARIES 0-11490.csv')
df_bart_test=pd.read_csv('/kaggle/input/bart-and-pegasus-full-train/Bart_Summaries_Test_0-11490 .csv')
df_pegasus_test=pd.read_csv('/kaggle/input/bart-and-pegasus-full-train/Pegasus_Test_Summaries_0-11491-FULL.csv')
df_t5_test=pd.read_csv('/kaggle/input/bart-and-pegasus-full-train/T5_SUMMARIES_(0-11490).csv', encoding='ISO-8859-1')
```

These 3 cells are used to clean the generated summaries from stopwords, punctuation and lowercase all the words by using the defined function “clean_text” , then we call it many times for the 3 models for train and test data

```python
import nltk
import string
nltk.download('punkt')
# download stopwords if not already downloaded
nltk.download('stopwords')

# define function to clean text
def clean_text(text):
    # remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation+ '.'))
    # convert to lowercase
    text = text.lower()
    # tokenize text into words
    words = nltk.word_tokenize(text)
    # remove stopwords
    stopwords = nltk.corpus.stopwords.words('english')
    words = [word for word in words if word not in stopwords]
    # join words back into a string
    text = ' '.join(words)
    return text
```

```python
[185] # apply cleaning function to summary column
      df_bart_train['summary'] = df_bart_train['summary'].apply(clean_text)

      # apply cleaning function to highlights column
      df_bart_train['highlights'] = df_bart_train['highlights'].apply(clean_text)
```

```python
[186] # apply cleaning function to summary column
      df_pegasus_train['summary'] = df_pegasus_train['summary'].apply(clean_text)

      # apply cleaning function to highlights column
      df_pegasus_train['highlights'] = df_pegasus_train['highlights'].apply(clean_text)
```

This cell is repeated many times for the 3 models to encode the generated summaries for train and test

```
x_bart_train = []
for i, summary in df_bart_train['summary'].items():
    encoded_summary=bart_tokenizer.encode(summary, truncation=True, max_length=1024, padding='max_length', return_tensors='pt')
    x_bart_train.append((i, encoded_summary.numpy().squeeze()))
```

This cell convert the input data to numpy arrays

```
# Convert the input data to numpy arrays
x_t5_train = np.array(x_t5_train)
x_t5_test = np.array(x_t5_test)
x_bart_train = np.array(x_bart_train)
y_train = np.array(y_train)
x_bart_test = np.array(x_bart_test)
y_test = np.array(y_test)
x_pegasus_train = np.array(x_pegasus_train)
x_pegasus_test = np.array(x_pegasus_test)
```

This cell stack the 3 summaries of the models for train and test to be fitted in the base models

```
# Stack the BART and Pegasus summaries
predictions_train = np.column_stack((x_pegasus_train_summaries[0:11487], x_t5_train_summaries,x_bart_train_summaries[0:11487]))
predictions_test = np.column_stack((x_pegasus_test_summaries, x_t5_test_summaries[0:11489],x_bart_test_summaries[0:11489]))
```

These are the base models of the stacking regressor meta model

```
from sklearn.ensemble import RandomForestRegressor

rf_base = RandomForestRegressor(n_estimators = 1)
rf_base.fit(predictions_train,  y_train_summaries[0:11487])
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=1)
```

```
from sklearn.neighbors import KNeighborsRegressor

# Train a KNeighborsRegressor on Pegasus summaries
knn_base = KNeighborsRegressor(n_neighbors=5, weights='distance')
knn_base.fit(predictions_train,  y_train_summaries[0:11487])
```

```
KNeighborsRegressor
KNeighborsRegressor(weights='distance')
```

These are the base models of the Gradient Boosting Regressor meta model

```
# Train a RandomForestRegressor on BART summaries
rf_base = RandomForestRegressor(n_estimators=1)
rf_base.fit(predictions_train, y_train_summaries[0:11490])
```

```
RandomForestRegressor
RandomForestRegressor(n_estimators=1)
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.linear_model import LinearRegression

# Train a KNeighborsRegressor on Pegasus summaries
mlp_base = MLPRegressor()
mlp_base.fit(predictions_train, y_train_summaries[0:11490])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilaye
  warnings.warn(
```

```
MLPRegressor
MLPRegressor()
```

These are the predictions of the base models

```
rf_preds=rf_base.predict(predictions_test).ravel()
knn_preds=knn_base.predict(predictions_test).ravel()

# Step 5: Create a new DataFrame to store the predictions
predictions = pd.DataFrame({"rf_preds": rf_preds, "knn_preds":knn_preds})
```

The Stacking Regressor meta model and we give him the base models: Random Forest Regressor and KN Regressor

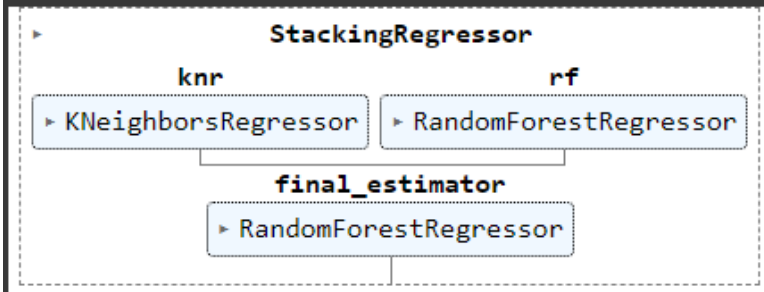
```
from sklearn.ensemble import StackingRegressor
import numpy as np

# Initialize base estimators
estimators = [
    ('  knr', KNeighborsRegressor(n_neighbors=5, weights='distance',p=1)),
    ('rf',  RandomForestRegressor())
]


# Select a random subset of 1000 rows from the data
n_samples = 5000
random_indices = np.random.choice(len(predictions), n_samples, replace=False)

predictions_subset = predictions.iloc[random_indices]
y_test_flat_subset = y_test_flat[random_indices]


# Initialize meta-model with base estimators
meta_model =StackingRegressor(
    estimators = estimators,
    final_estimator = RandomForestRegressor(n_estimators = 1))
meta_model.fit(predictions_subset, y_test_flat_subset)
```



The Gradient Boosting Regressor meta model and we give him the base models: Random Forest Regressor and MLP Regressor

```
from sklearn.ensemble import GradientBoostingRegressor
import numpy as np

# Initialize base estimators
estimators = [
    ('rf',  RandomForestRegressor()),
    ('mlp', MLPRegressor()),
]


# Initialize meta-model with base estimators
meta_model =GradientBoostingRegressor(n_estimators=100)
meta_model.fit(predictions, y_test_flat)
```

```
▾ GradientBoostingRegressor
GradientBoostingRegressor()
```

This is the prediction of ensemble meta model

```
[24] test_rf_preds=rf_base.predict(predictions_test).ravel()
     test_knn_preds=knn_base.predict(predictions_test).ravel()

[25] # Step 5: Create a new DataFrame to store the predictions
     test_predictions = pd.DataFrame({"rf_preds": test_rf_preds, "knn_preds":test_knn_preds})

[26] ensemble_preds = meta_model.predict(test_predictions)
```

This cell decode the ensemble prediction to text to measure the rouge scores

```
ensemble_pred_text = []
for  i in range(len(ensemble_preds)):
    summary_ids = ensemble_preds[i]
    summary_ids = summary_ids.astype(int)
    summary = bart_tokenizer.decode(summary_ids, skip_special_tokens=True, max_length=100)
    ensemble_pred_text.append(summary)
```

This cell calculate the rouge scores

```
import sys

sys.setrecursionlimit(10000) # set the recursion limit to 10000

rouge = Rouge()
batch_size = 250
num_batches = (len(ensemble_pred_text) + batch_size - 1) // batch_size # round up to nearest integer

scores = []
for i in range(num_batches):
    start_idx = i * batch_size
    end_idx = min((i+1) * batch_size, len(ensemble_pred_text))
    batch_pred_text = ensemble_pred_text[start_idx:end_idx]
    batch_ref_text = y_test_text[start_idx:end_idx]

    batch_scores = []
    for j in range(len(batch_pred_text)):
        reference = str(batch_ref_text[j])
        hypothesis = str(batch_pred_text[j])
        if not reference:
            score = {'rouge-1': {'f': 0, 'p': 0, 'r': 0}, 'rouge-l': {'f': 0, 'p': 0, 'r': 0}}
        else:
            score = rouge.get_scores(hypothesis, reference)
        batch_scores.append(score)

    scores.extend(batch_scores)
```

6: Experimental Results and Analysis

6.1 Analysis of the Results and Comparison of Models:

In this section we will present the results of our study.

We used rouge score to determine the performance of models we used.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics commonly used for evaluating the quality of automatic text summarization and machine translation output. The ROUGE metrics are based on comparing the text generated by the summarization or translation system to one or more reference texts (usually human-written summaries or translations) and measuring the overlap between them.

We used ROUGE-1 and ROUGE-L. ROUGE-1 measures the overlap of unigrams (single words) between the generated summary and the reference summary, while ROUGE-L measures the longest common subsequence between the generated summary and the reference summary. These metrics are commonly used to evaluate the quality of automatic text summarization systems.

Generated Summary results:

	Rouge-1	Rouge-L
Bart	40.08	37.74
Pegasus	38.82	36.66
T5	39.32	37.54

Here is the results of the Gradient Boosting Regressor :

	Rouge-1	Rouge-L
Pegasus & T5	35.23	32.87
Bart & Pegasus	36.64	34.29
Bart & T5	38.12	36.35
Bart & Pegasus & T5	39.74	36.62

The table shows the ROUGE-1 and ROUGE-L scores for different combinations of summarization models. Based on the results, the combination of Bart, Pegasus, and T5 achieved the highest ROUGE scores, with a ROUGE-1 score of 39.74 and a ROUGE-L score of 36.62 .When comparing the different model combinations, it is notable that all combinations that included Bart achieved higher scores than those without. This suggests that Bart is a particularly effective model for text summarization. Additionally, the combination of Bart and T5 achieved relatively high ROUGE scores, indicating that these models may also be effective for summarization.

Here is the results of the Stacking Regressor :

	Rouge-1	Rouge-L
Pegasus & T5	37.19	36.63
Bart & Pegasus	38.40	35.13
Bart & T5	39.48	37.88
Bart & Pegasus & T5	41.48	38.16

The table shows the ROUGE-1 and ROUGE-L scores for different combinations of summarization models. The results suggest that the combination of Bart, Pegasus, and T5 achieved the highest ROUGE scores, with a ROUGE-1 score of 41.48 and a ROUGE-L score of 38.16. Nevertheless, it is clear that all combinations that included Bart achieved higher scores than those without, indicating that Bart is a particularly effective model for text summarization. Additionally, the combination of Bart and T5 achieved relatively high ROUGE scores, suggesting that these models may also be effective for summarization.

6.2 Discussion of the limitations and future directions for research:

Limitations:

- The study only evaluated a limited set of summarization models, and did not explore other important factors that could affect summary quality, such as the impact of different input text lengths, the effect of different training data, or the use of pre-processing techniques.
- The study relied solely on ROUGE scores to evaluate the quality of the generated summaries, and did not consider other important factors such as coherence, readability, and relevance to the original text.

- The sample size and other details of the experiment were not provided, making it difficult to draw firm conclusions about the statistical significance of the results.

Future directions for research:

- Conduct a more comprehensive evaluation of summarization models, considering a wider range of models and evaluating them using multiple evaluation metrics.
- Explore the impact of different input text lengths on the quality of generated summaries, and investigate the use of pre-processing techniques to improve summarization performance.
- Conduct human evaluations of the generated summaries to assess their coherence, readability, and relevance to the original text.
- Investigate the use of transfer learning and multi-task learning techniques to improve the performance of summarization models.
- Consider the use of different evaluation metrics for summarization, such as those that take into account the semantic meaning of the words or the coherence of the generated text.
- Investigate the use of unsupervised learning techniques for summarization, which may be useful in scenarios where labeled training data is not available.

7: Conclusion and Future Work:

7.1 Summary of the key findings and contributions of the documentation:

The documentation presents an evaluation of different combinations of summarization models using the ROUGE-1 and ROUGE-L metrics. The results suggest that the combination of Bart, Pegasus, and T5 achieved the highest ROUGE scores, indicating that these models may be particularly effective for text summarization. Additionally, the study found that Bart is a particularly effective model for summarization, as all combinations that included Bart achieved higher scores than those without.

The key contribution of the documentation is that it provides insights into the effectiveness of different summarization models and highlights the importance of using multiple models to achieve high-quality summaries. Moreover, it demonstrates the usefulness of the ROUGE-1 and ROUGE-L metrics for evaluating the performance of summarization models.

7.2 Discussion of the potential impact of the work on the NLP community:

The work has the potential to make a significant impact on the NLP community by providing guidance on the selection of summarization models for specific tasks. The findings can be used to inform the development of new summarization models or the improvement of existing ones. Additionally, the documentation highlights the importance of evaluating summarization models using multiple metrics, which can help to ensure that the generated summaries are of high quality and meet the needs of end-users.

7.3 Suggestions for future research directions and improvements to the techniques presented:

Future research can build on the findings presented in the documentation by exploring the impact of different input text lengths, the effect of different training data, and the use of pre-processing techniques on summarization performance. Additionally, future work can investigate the use of transfer learning and multi-task learning techniques to improve the performance of summarization models. Furthermore, there is a need to evaluate the generated summaries using multiple evaluation metrics, including those that consider coherence, readability, and relevance to the original text. Finally, future research can investigate the use of unsupervised learning techniques for summarization, which may be useful in scenarios where labeled training data is not available.

7.4 References

1-https://arxiv.org/pdf/1910.13461v1.pdf

2-https://arxiv.org/pdf/1912.08777v2.pdf

3-https://arxiv.org/pdf/1910.10683v3.pdf

4-https://paperswithcode.com/sota/abstractive-text-summarization-on-cnn-daily