



SRH University Heidelberg

Master's Thesis

Plagiarism Detection and Paraphrase Based on Generative
Artificial Intelligence

Author:	Birwadkar, Rahul
Matriculation Number:	11037364
Address:	Bonhoefferstraße 13 69123 Heidelberg Germany
Email Address:	11037364@stud.hochschule- heidelberg.de
Study Programme:	M.E in Information Technology
1 st Supervisor:	Prof. Dr. A. Gottscheber
2 nd Supervisor:	Prof. Dr.-Ing. Milan Gnjatovic'
Begin:	01. October 2024
End:	01. March 2025

Acknowledgement

I would like to express my sincere gratitude to everyone who has supported me throughout the journey of completing this master's thesis, "***Plagiarism Detection and Paraphrase Based on Generative Artificial Intelligence***".

First and foremost, I extend my deepest appreciation to my supervisor, **Prof. Dr. A. Gottscheber**, for his invaluable guidance, encouragement, and expertise throughout the research process. His constructive feedback and insightful discussions have played a crucial role in shaping the direction of this work.

I am especially grateful to my second supervisor, **Prof. Dr.-Ing. Milan Gnjatović**, whose mentorship and support have been instrumental in refining my research approach. His expertise and insightful suggestions greatly contributed to the depth and clarity of my work.

I would also like to extend my heartfelt thanks to **SRH Hochschule Heidelberg**, its faculty, and staff for providing me with the resources and a conducive academic environment to conduct my research. The support from the **Information Technology department** has been invaluable.

On a personal level, I am deeply thankful to my family and friends for their unwavering encouragement, patience, and emotional support during this journey. Their belief in me has been a source of motivation during challenging times.

Lastly, I would like to express special gratitude to **Prof. Milan Gnjatović**, whose guidance and support went beyond academics, helping me navigate various aspects of my research with clarity and confidence.

This thesis is a reflection of the collective support and encouragement I have received from many individuals, and I remain deeply grateful to each one of them.

Rahul Birwadkar

Heidelberg, March 2025

Declaration

I, **Rahul Birwadkar**, hereby declare that this master's thesis, titled "**Plagiarism Detection and Paraphrase Based on Generative Artificial Intelligence**," is my own original work and has been carried out independently under the supervision of **Prof. Dr. A. Gottscheber** and co-supervision of **Prof. Dr.-Ing. Milan Gnjatović**.

I confirm that I have appropriately cited and referenced all sources used in this thesis in accordance with academic integrity guidelines. Any content taken from external sources, including text, figures, or data, has been duly acknowledged and credited.

I further declare that this thesis has not been submitted, in whole or in part, to any other academic institution for the purpose of obtaining a degree or qualification.

I understand that any form of academic dishonesty, including plagiarism, will result in appropriate disciplinary action as per the regulations of **SRH Hochschule Heidelberg**.

Heidelberg, March 2025

Rahul Birwadkar

Matriculation Number: 11037364

Abstract

Plagiarism detection is an important aspect of research, publication, and academic writing. Every year, numerous research papers, books, journals, and novels are published. If someone steals this content and republishes it as their own creation, it becomes a serious issue. While copying and repurposing content may seem easy, it significantly impacts the original author or owner. When writers publish their work, it should be secured as their intellectual property by respective authorities. To ensure content originality, plagiarism detection becomes essential.

Plagiarism detection is automated through specialized software, with various tools available for content verification. In this thesis, I study and demonstrate plagiarism detection and paraphrasing using generative artificial intelligence. When researchers express their work in writing, they must communicate clearly and comprehensibly. Sometimes research works address similar topics from different perspectives, requiring alternative words or synonyms. In these cases, paraphrasing tools help writers create content while preserving the original meaning.

Artificial intelligence offers a practical solution for writers and researchers, aiding in both paraphrasing during content generation and plagiarism detection before publication. Natural language processing, a subset of AI, will be extensively used throughout this thesis.

Diving deeper into the topic, I have developed an algorithm for plagiarism detection that analyses both syntactic and semantic similarity between input content and a comprehensive dataset. This dataset contains extensive content for verification, and the similarity score determines the content's originality.

For paraphrasing, I have leveraged pre-trained generative AI models fine-tuned on relevant datasets to effectively generate alternative versions of input content while maintaining its original meaning. Additionally, multiple pre-trained models have been employed to provide diverse paraphrased outputs, ensuring flexibility and accuracy.

By combining plagiarism detection and paraphrasing through generative AI, this research not only ensures content originality but also facilitates ethical content creation. The integration of AI-driven methods enhances the accuracy and efficiency of plagiarism detection while providing meaningful paraphrased alternatives for researchers and writers. This thesis contributes to the growing field of natural language processing by demonstrating a practical approach to maintaining academic integrity and promoting responsible content generation.

Kurzfassung

Die Erkennung von Plagiaten ist ein wichtiger Aspekt der Forschung, der Veröffentlichung und des akademischen Schreibens. Jedes Jahr werden zahlreiche Forschungsarbeiten, Bücher, Fachzeitschriften und Romane veröffentlicht. Wenn jemand diese Inhalte stiehlt und sie als seine eigene Schöpfung veröffentlicht, wird dies zu einem ernsten Problem. Das Kopieren und Wiederverwenden von Inhalten mag zwar einfach erscheinen, hat aber erhebliche Auswirkungen auf den ursprünglichen Autor oder Eigentümer. Wenn Autoren ihre Werke veröffentlichen, sollten sie von den zuständigen Behörden als ihr geistiges Eigentum geschützt werden. Um die Originalität der Inhalte zu gewährleisten, ist die Erkennung von Plagiaten unerlässlich.

Die Erkennung von Plagiaten wird durch spezialisierte Software automatisiert, wobei verschiedene Tools für die Überprüfung von Inhalten zur Verfügung stehen. In dieser Arbeit untersuche und demonstriere ich die Plagiatserkennung und Paraphrasierung mit Hilfe generativer künstlicher Intelligenz. Wenn Forscher ihre Arbeit schriftlich niederlegen, müssen sie klar und verständlich kommunizieren. Manchmal behandeln Forschungsarbeiten ähnliche Themen aus verschiedenen Blickwinkeln, was alternative Wörter oder Synonyme erfordert. In diesen Fällen helfen Paraphrasierungswerkzeuge den Autoren, Inhalte zu erstellen und dabei die ursprüngliche Bedeutung beizubehalten.

Künstliche Intelligenz bietet eine praktische Lösung für Autoren und Forscher, die sowohl bei der Paraphrasierung während der Erstellung von Inhalten als auch bei der Erkennung von Plagiaten vor der Veröffentlichung helfen kann. Die Verarbeitung natürlicher Sprache, ein Teilbereich der KI, wird in dieser Arbeit ausgiebig verwendet.

Zur Vertiefung des Themas habe ich einen Algorithmus zur Plagiatserkennung entwickelt, der sowohl die syntaktische als auch die semantische Ähnlichkeit zwischen den eingegebenen Inhalten und einem umfassenden Datensatz analysiert. Dieser Datensatz enthält umfangreiche Inhalte zur Überprüfung, und der Ähnlichkeitswert bestimmt die Originalität des Inhalts.

Für die Paraphrasierung habe ich vortrainierte generative KI-Modelle eingesetzt, die anhand relevanter Datensätze feinabgestimmt wurden, um effektiv alternative Versionen von Eingabeinhalten zu generieren, ohne die ursprüngliche Bedeutung zu verändern. Darüber hinaus wurden mehrere vortrainierte Modelle eingesetzt, um verschiedene paraphrasierte Ausgaben zu liefern und so Flexibilität und Genauigkeit zu gewährleisten.

Durch die Kombination von Plagiatserkennung und Paraphrasierung mittels generativer KI stellt diese Forschung nicht nur die Originalität der Inhalte sicher, sondern erleichtert auch die ethische Erstellung von Inhalten. Die Integration von KI-gesteuerten Methoden verbessert die Genauigkeit und Effizienz der Plagiatserkennung und bietet gleichzeitig sinnvolle Paraphrasierungsalternativen für Forscher und Autoren. Diese Arbeit leistet einen Beitrag zum wachsenden Bereich der Verarbeitung natürlicher Sprache, indem sie einen praktischen Ansatz zur Wahrung der akademischen Integrität und zur Förderung verantwortungsvoller Inhaltserstellung aufzeigt.

Contents

Acknowledgement.....	2
Declaration.....	3
Abstract.....	4
Kurzfassung	5
Contents.....	6
List of Figures	8
List of Tables	8
Chapter 1 : Introduction.....	9
4.1 Literature Review	10
Chapter 2 : Natural Language processing (NLP).....	14
2.1 What is Natural Language processing (NLP).....	14
2.2 Natural Language processing (NLP) Pipeline.....	14
2.2.1 Data Acquisition and Cleaning.....	15
2.2.2 Preprocessing	15
2.2.3 Feature Engineering	15
2.2.4 Modelling for Plagiarism Detection	16
2.2.5 Evaluation Metrics.....	16
2.2.6 Deployment	16
2.2.7 Monitoring and Model Updating	17
Chapter 3 : Phase I.....	18
3.1 Dataset Selection	18
3.2 Similarity Calculations.....	20
3.2.1 Cosine Similarity.....	20
3.2.2 L2 Normalization.....	21
3.2.3 Hybrid Similarity	24
3.3 Model Selection for Plagiarism Detection.....	25
3.3.1 BERT : The Foundation of MiniLM	25
3.3.2 (LLM) Large Language Model Distillation	26
3.3.3 MiniLM Model.....	29
3.3.4 SBERT: Sentence-BERT.....	30
3.3.5 all-MiniLM-L6-v2: A Sentence Transformer Model	31
3.4 Model Fine Tuning Process	32
Chapter 4 : Phase II.....	39

4.1	Dataset Selection	39
4.2	Transformer	41
4.2.1	Encoder	43
4.2.2	Decoder	60
4.3	Model Selection for Paraphrase Generation	66
4.4	Model Fine Tuning Process	73
4.5	Evaluation Metrics for Paraphrasing Quality Assessment.....	75
Chapter 5	: Phase III.....	79
5.1	Web Scraping for Plagiarism Detection Using BeautifulSoup	79
5.2	Legality of Web Scraping	80
5.3	Implementation of Web Scraping for Plagiarism Detection	81
5.4	Deployment.....	82
Chapter 6	Result and Analysis	84
6.1	Plagiarism Detection Model Evaluation and Performance Analysis.....	84
6.2	Web Scraping Similarity Index Results	92
6.3	Paraphrase Model Evaluation and Performance Analysis	93
Conclusion.....		96
Future Work		97
Bibliography		98

List of Figures

Figure 2.1: NLP Pipeline	14
Figure 3.1: XML File 1.....	19
Figure 3.2: XML File 2.....	19
Figure 3.3: Student-Teacher Model	29
Figure 3.4: MiniLM Model.....	30
Figure 4.1: Transformer Architecture (Vaswani et al., 2017).....	43
Figure 4.2: Encoder	44
Figure 4.3: Example of Attention(Q,K,V)	52
Figure 4.4: Decoder	63
Figure 4.5: BART Architecture.....	68
Figure 6.1: Pre-Train Model Result (Threshold = 0.4).....	84
Figure 6.2: Pre-Train Model Similarity Score Result (Threshold = 0.4)	85
Figure 6.3: Pre-Train Model Result (Threshold = 0.8).....	86
Figure 6.4: Sentence Transformer-based Fine-Tune Model Result (Threshold = 0.4) ..	87
Figure 6.5 Sentence Transformer-based Fine-Tune Model Result (Threshold = 0.8)....	88
Figure 6.6: Direct Fine-Tuning with MiniLM Model Result (Threshold =0.4)	89
Figure 6.7: Direct Fine-Tuning with MiniLM Model Result (Threshold =0.8)	90
Figure 6.8: Web Scraping Similarity Index Results 1	92
Figure 6.9: Web Scraping Similarity Index Results 2	92
Figure 6.10: Pre-Train BART Performance.....	93
Figure 6.11: Fine-Tune BART Performance.....	94
Figure 6.12: DeepSeek Model Performance.....	95

List of Tables

Table 3.1: Example TF-IDF vectors.....	21
Table 3.2: TF-IDF value before Normalization.....	23
Table 3.3: : TF-IDF value After Normalization	23
Table 3.4: Popular Distilled Models	28
Table 4.1: Example BoW.....	46
Table 4.2: Example Embedding Table.....	54
Table 4.3: Contextual Embeddings Table 1	55
Table 4.4: Contextual Embeddings Table 2.....	55
Table 4.5: Final Contextualized Embedding.....	56
Table 4.6: Difference between Encoder and Decoder	65
Table 4.7: Deepseek Model Paraphrase Examples.....	72
Table 4.8: Fine-Tune BART Training & validation Loss.....	75
Table 5.1: List of Scraped Websites	82
Table 6.1: : Plagiarism Detection Models Comparison	91
Table 6.2: Paraphrase Generation Models Comparison	95

Chapter 1 : Introduction

Plagiarism is a growing concern in academic writing, research, and content creation. With the vast amount of digital information available, ensuring originality has become more challenging. Plagiarism occurs when someone copies another person's work without giving credit, which can have serious consequences, such as academic penalties or legal issues. To prevent this, plagiarism detection tools are used to check whether content is original or copied from existing sources.

Traditionally, plagiarism detection relied on basic text-matching methods, which could only identify exact copies of content. However, people often modify or rewrite text to avoid detection, making traditional methods less effective. With advancements in Artificial Intelligence (AI) and Natural Language Processing (NLP), modern tools can now detect not just exact matches but also reworded or paraphrased content.

Paraphrasing plays an important role in academic writing, allowing researchers and writers to express ideas in their own words while keeping the original meaning. AI-powered paraphrasing tools help rewrite content in a clear and natural way, ensuring originality without altering the intent. In this thesis, I explore how AI can be used for both plagiarism detection and paraphrasing to improve content integrity and assist writers.

Objectives of the Study

This research has two key objectives:

1. Detect plagiarism using AI to compare text with existing content and identify copied material.
2. Paraphrase content using AI-based models to rewrite text while preserving its original meaning.

4.1 Literature Review

For the purposes of natural language processing (NLP) and academic integrity, two of the most important areas of study are the detection of plagiarism and the development of paraphrases. The expansion of digital material has brought to an increase in issues over the originality of content and the attribution of authorship. The early approaches of detecting plagiarism relied mostly on string-matching algorithms and lexical similarity measures. These methods were restricted to finding precise text matches, but they were not able to identify semantic copying. The development of machine learning has led to the emergence of increasingly complex models that make use of statistical and probabilistic methods. These models were then followed by deep learning-based approaches that considerably improved performance.

In the same way, the way paraphrases are made has changed over time. It used to be based on rule-based techniques that used predefined syntactic transformations and thesaurus-based word replacements. Now, statistical machine translation (SMT) methods use probabilistic phrase alignments to make paraphrases. By adding memory mechanisms to maintain contextual information throughout sequences, the introduction of recurrent neural networks (RNNs) and its variations, such as Long Short-Term Memory (LSTM) networks, signified a significant change in the creation of text. This transition was brought about by the incorporation of memories. On the other hand, these methods encountered considerable difficulties in terms of training efficiency, scalability, and the management of long-range relationships. In more recent times, transformer-based generative artificial intelligence models, which include BART, T5, and GPT-based architectures, have revolutionized the area by providing results that are considered to be state-of-the-art in terms of paraphrase creation and plagiarism detection.

This survey of the relevant literature investigates the most recent and cutting-edge methods that are used in the detection of plagiarism and the development of paraphrases. It takes a look at both classic and contemporary approaches. It draws attention to the gaps in research that currently exist, provides comparative assessments, and discusses possible future approaches in the subject.

1.1.1 Plagiarism Detection: Traditional and Modern Approaches

Two groups that can be used to organize different types of theft detection systems are intrinsic and secondary detection. One of the main goals of intrinsic plagiarism detection is to find mistakes within a single text. It finds the strange changes in style that point to information that has been copied using this method. When looking for secondary plagiarism, on the other hand, you compare your work to a set of outside sources to find plagiarized parts.

1.1.2 Early Approaches: Lexical and Statistical Methods

The first methods for finding plagiarized work were mostly based on word similarity metrics. These included methods like. These methods divide text into groups of words that overlap (n-grams) or characters that overlap (character n-grams) so that patterns can be found. In this group, n-grams and overlapping sequences are two examples. Jaccard similarity and cosine similarity were often used to figure out how similar two works were. In the early forms of programs that found copying, these two methods were used a lot.

TF-IDF, which stands for "Term Frequency–Inverse Document Frequency," is a mathematical method used to rate the importance of words by comparing how often they appear in a document to how many times they appear in complete texts. The TF-IDF algorithm has trouble finding modified copying because it relies on word repetition. However, it is good at finding similarity based on keywords

These methods worked well for finding plagiarism that was copied exactly the same, but they often missed cases of meaning similarity, rewriting, and rearranging the structure of words.

1.1.3 Traditional Machine Learning Approaches

Get around the problems with lexical-based methods, experts started using machine learning models to find copying. These models used feature engineering and classification algorithms, such as

Support Vector Machines (SVMs): These models used language characteristics, like word frequency patterns and changes in sentence structure, to tell the difference between copied and original text (Awale et al., 2020).

Naive Bayes and Decision Trees: These models used statistical language models to sort text into groups by how likely each word was to be used and how it was put together (Kalleberg, 2015).

The Jaccard and Cosine Similarity with Feature Engineering: Feature selection methods were used to improve similarity calculations, which led to more accurate spotting (Al-Jibory & Al-Tamimi, 2021).

Traditional machine learning models had a lot of problems, like needing a lot of feature engineering, being depending on datasets, and not being able to understand what words mean.

1.1.4 Neural Network-Based Approaches and Their Limitations

Before transformer models came along, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks were commonly used to find copying and produce new phrases. These models read text in a certain order and kept a secret state that saved data from earlier sources (Sutskever et al., 2014). But the fact that they relied on sequential data handling caused a number of problems:

Gradient Problem: Standard RNNs had trouble learning long-range relationships because gradients got smaller as sequences got longer (Bengio et al., 1994).

Inefficient use of computers: training RNN and LSTM models took a lot longer and used more computers because the text had to be handled one step at a time (Cho et al., 2014).

Limited Parallelization: LSTMs and RNNs couldn't handle text easily in parallel like modern transformer-based systems could, which raised worries about scalability (Bahdanau et al., 2015).

Even though LSTMs were better than regular RNNs, they weren't as good as current deep learning methods because they needed to be tuned in by hand for hyperparameters and weren't good at catching global relationships.

1.1.5 Modern Deep Learning Approaches for Plagiarism Detection

Deep learning and NLP embeddings are used in modern methods for finding copying, which greatly improves performance:

Latent Semantic Analysis (LSA) and Word Embeddings: These methods, such as Word2Vec and GloVe, made identification more accurate by looking at the context of words instead of just matching exact words (Mikolov et al., 2013).

Transformers and Embedding Sentences: Transformer-based models, such as BERT, SBERT, and MiniLM, store the meaning of sentences, which makes it easier to find copying in paraphrased text (Kalleberg, 2015).

Hybrid NLP Approaches: Some systems use methods like POS tagging, dependency parsing, and contextual embeddings along with machine learning and NLP preparation (Al-Jibory & Al-Tamimi, 2021).

These deep learning models work better than older ones, but they need a lot of structured data and a lot of computing power.

1.1.6 Paraphrase Generation: From Rule-Based to Generative Models

Similar to how copying is found, techniques for paraphrasing have evolved over time, moving from rule-based approaches to statistical models, LSTMs, and transformers.

Early approaches based on rules and statistics:

- **Word Substitutions Based on Thesauri:** Simple methods of rewriting depended on replacing words with synonyms, which didn't always make sense grammatically (Lin & Pantel, 2001).
- **PB-SMT stands for Phrase-Based Statistical Machine Translation.** PB-SMT was first created to translate languages. It matched sentences probabilistically to make paraphrases (Quirk et al., 2004).

1.1.7 RNN and LSTM-Based Paraphrase Models

Sequ2Seq LSTM Models: These models took in text, mapped it to a secret, compressed state, and made output that was paraphrased. Unfortunately, they had exposure bias and did worse on long text sequences (Sutskever et al., 2014).

Attention-Based LSTMs: Adding attention processes helped models focus on important input words, which increased consistency (Luong et al., 2015). Even with these changes, LSTMs were not scalable and needed too many computing resources.

3.3 Transformer-Based Paraphrase Models

Modern models built on transformers are better than older ones because they are more aware of context, can work in parallel, and are more efficient:

Lewis et al. (2020) talk about BART (Bidirectional and Auto-Regressive Transformers).

- **T5 (Text-to-Text Transfer Transformer)** (Raffel et al., 2020).

- Training DeepSeek and GPT Models on a variety of data made paraphrase even more accurate.

RNNs and LSTMs used to be the most popular NLP models, but transformers have mostly taken their place because they are more accurate, can be used on larger datasets, and understand semantics better. To move the field forward, more study should be done on mixed detection systems, international theft detection, and ethical AI uses.

Chapter 2 : Natural Language processing (NLP)

2.1 What is Natural Language processing (NLP)

Natural Language Processing (NLP) enables machines to understand and generate human language. Modern NLP models use deep learning techniques, particularly transformer-based architectures, to improve text analysis tasks such as plagiarism detection and paraphrasing.

Transformer models like T5 (Text-to-Text Transfer Transformer) and BART (Bidirectional Auto-Regressive Transformer) have been widely used for text generation and paraphrasing. In this research:

- T5 was used for paraphrasing because of its ability to generate reworded text while maintaining the original meaning.
- BART was explored for paraphrasing as it is effective in text rewriting tasks with minimal loss of context.
- TF-IDF and Cosine Similarity were used for plagiarism detection to compare input text against a reference dataset.

These pre-trained models were chosen for their efficiency, eliminating the need for training large models from scratch. By leveraging these state-of-the-art NLP techniques, this research improves plagiarism detection and assists writers in rewording content effectively.

2.2 Natural Language processing (NLP) Pipeline

Large-scale databases and several methods are needed to accomplish plagiarism detection. The work is incorporated in the first phase using a Natural Language Processing (NLP) pipeline as seen in the following ordered workflow:

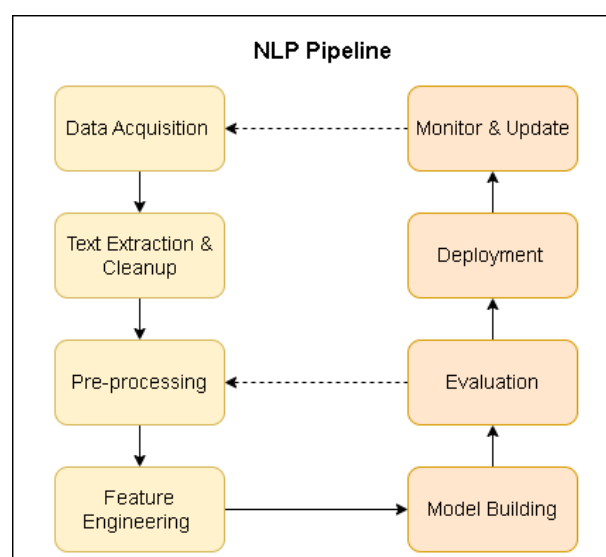


Figure 2.1: NLP Pipeline

2.2.1 Data Acquisition and Cleaning

Plagiarism detection requires access to extensive textual data. The following are the main data sources for this task:

- PAN Plagiarism Corpus 2011 (a reference dataset for plagiarism identification)
- Web Scraping (data extraction from web sources with BeautifulSoup)

The PAN plagiarism corpus from 2011 served as the main dataset for this investigation. This corpus consists of:

- Source files, or original material
- Files (materials including either manually or automatically introduced plagiarism) that seem suspicious

Extensive cleaning is not required as the dataset is well-structured and devoid of noisy or pointless material. Cleaning techniques include deleting HTML elements, stop words, and special characters are used, nevertheless, when extracting material from the web.

2.2.2 Preprocessing

In machine learning and natural language processing (NLP), preprocessing is a very important step that makes sure the data is organized in a way that makes it easy to analyse. During this stage,

- Tokenization is the process of breaking text into smaller pieces, like words or sentences.
- Stop word Removal: Eliminating common words like "the," "is," and "and" that don't add much sense is called "stop word removal."
- Lowercasing: Case-sensitivity problems can be avoided by making all text lowercase.
- Lemmatization/Stemming: Converting words to their root forms (e.g., "running" → "run").
- Removing Special Characters and Punctuation: cleaning up text to make it easier to find patterns.

These steps help to normalize the text, which makes the feature extraction and pattern recognition steps that follow work better.

2.2.3 Feature Engineering

Feature engineering is the process of converting raw text input into numerical representations for machine learning models. Key strategies include:

- TF-IDF (Term Frequency-Inverse Document Frequency): Determines word significance based on frequency in a document vs occurrence in the overall dataset.

- Word Embeddings (Word2Vec, GloVe, or BERT embeddings): Identifies semantic links between words.
- N-grams: N-grams are text sequences that collect contextual information.

These methods are critical for properly recognizing similarities and variances in textual information.

2.2.4 Modelling for Plagiarism Detection

Machine learning and deep learning models are used to compare papers and identify plagiarism. Some popular techniques include:

- Cosine Similarity: Determines the similarity of two document vectors.
- Jaccard Similarity: Measures text overlap between word sets.
- Deep Learning Models (BERT, RoBERTa, or SBERT) use transformer-based models to discover semantic similarity.

These models assist in determining if a particular work is plagiarized, partly plagiarized, or original.

2.2.5 Evaluation Metrics

To assess the accuracy and effectiveness of plagiarism detection methods, the following essential metrics are used:

- Precision: The percentage of accurately recognized plagiarized cases among all instances reported as plagiarism.
- Recall: Assesses the model's capacity to recognize all true plagiarism instances while minimizing false negatives.
- The F1-score is the harmonic mean of accuracy and recall, which provides a fair evaluation of model performance.
- Confusion Matrix: A complete visual representation of true positives, false positives, true negatives, and false negatives for analyzing model misclassifications.

A well-tuned model for finding plagiarism should try to get a high recall (catch most cases of plagiarism) while reducing false positives to avoid giving the wrong impression of plagiarism.

2.2.6 Deployment

The following methods can be used to deploy the plagiarism detection model once it has been successfully trained and assessed:

- Web-Based Applications: Integrated using Flask or Django, enabling API-based access.

- Cloud Platforms: Deployed on AWS, Google Cloud, or Firebase for scalability and remote access.
- Real-Time Processing: Optimized for instant plagiarism checks, enhancing efficiency.

2.2.7 Monitoring and Model Updating

Plagiarism detecting tools need to be updated often to keep up with how online content changes. During this time,

- Retraining with New Data: Getting used to new ways of plagiarizing.
- Updating Scraping Mechanisms: Making sure that users can get to the newest web papers.
- Monitoring False Positives and Negatives: Making sure that identification is still accurate.

These changes make sure that the system stays dependable, expandable, and strong against advanced copying techniques.

Chapter 3 : Phase I

3.1 Dataset Selection

This research uses the PAN 2011 dataset as the primary resource for plagiarism detection. The PAN (Plagiarism Detection) shared task is a recognized benchmark in text investigation, including structured datasets intended to assess plagiarism detection systems. The PAN 2011 dataset has a varied collection of plagiarism instances, from exact duplication to extensively paraphrased material, making it appropriate for assessing both conventional and sophisticated machine learning-based plagiarism detection methods.

3.1.1 PAN 2011 Dataset – External Plagiarism Detection

The PAN 2011 external plagiarism dataset compares a group of questionable documents to a collection of source documents in order to test how well plagiarism detection works. In real life, theft happens when content is taken from outside sources, with or without changes. This dataset models those situations. The dataset is organized into text files and XML metadata files that go with them. The metadata files give important details about cases of copying.

3.1.2 Structure of PAN 2011 External Dataset:

There are two main types of files in the PAN 2011 dataset: text files and XML information files. The way these files are set up makes it possible to add detailed notes that can be used to check for copying.

A. Text Files

The dataset includes a collection of plain text files that serve as:

- Suspicious documents – These contain potential instances of plagiarism.
- Source documents – These are original texts from which plagiarism may have been committed.

The dataset includes various degrees of text alterations, including precise copy-pasting, moderate paraphrasing, and extensive complication, therefore replicating authentic instances of plagiarism.

B. XML Metadata and Labels

Each suspicious document has a corresponding XML annotation file, which provides essential details about plagiarism occurrences:

- Start and end character positions – Indicating the exact location of plagiarized text within the suspicious document.
- Reference to the source document – Linking the plagiarized text to the original document.
- Type of plagiarism – Identifying whether the text is copied direct, lightly modified, or paraphrased.

- Perplexity level – Describing the extent to which the text has been altered from its original form.

These structured annotations allow for precise evaluation of plagiarism detection models by enabling both supervised learning approaches and rule-based detection systems.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<document reference="suspicious-document00001.txt">
  <feature name="about" authors="Wiggin, Kate Douglas Smith" title="Mother Carey's Chickens"
    lang="en"/>
  <feature name="md5Hash" value="727144157faa71b184b81ba92da52692"/>
</document>
```

Figure 3.1: XML File 1

This Figure 3.1 illustrates the Metadata for the document titled “*suspicious-document00001.txt*”. Based on the provided XML annotations, this document is classified as non-plagiarized, as no plagiarism type is indicated. There are no tags in the information that deal with theft, which means that the document does not contain any stolen or paraphrased text from other sources. This classification shows how important information is for telling the difference between copied and original material in the PAN 2011 dataset.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<document reference="suspicious-document00005.txt">
  <feature name="about" authors="Saint-Simon, Louis de Rouvroy, duc de" title="Memoirs of Louis
    XIV and His Court and of the Regency – Volume 06" lang="en"/>
  <feature name="md5Hash" value="ae971a081edc198d6a1132420419020f"/>
  <feature name="plagiarism" type="artificial" obfuscation="low" this_language="en"
    this_offset="19254" this_length="1557" source_reference="source-document00178.txt"
    source_language="en" source_offset="3835" source_length="1560"/>
</document>
```

Figure 3.2: XML File 2

This Figure 3.2 illustrates the Metadata for the document titled “*suspicious-document00005.txt*”. The “plagiarism” term in the information shows that this document has been copied from other sources based on the XML tags. “Artificial” copying means that the text has been changed in some way, and “low” obfuscation means that there have been few changes to the original content.

The metadata provides additional information, such as:

- The source document from which plagiarism has been detected (“source-document00178.txt”)
- The text position offsets, specifying the exact plagiarized portion in both the suspicious and source documents.
- The language of the text, which is English (en).

This structured metadata helps in systematically identifying plagiarism cases, allowing automated systems to detect and evaluate textual similarities efficiently.

3.1.3 Significance of Natural Labels in Model Training and Evaluation

The dataset's natural labels are very important for teaching machine learning models how to find plagiarism. These marks show if a paper has plagiarism and give information like the type, amount, and source of the theft. With these labeled examples, the model can learn to spot trends that are linked to various amounts of text similarity and change.

These names also make the review process easier because they provide a basis for judging how well the model works. By analyzing the model's forecasts to the real labels, you can find metrics like F1-score, recall, accuracy, and precision. This makes sure that the learned model can tell the difference between copied and original content and also be able to spot different ways that text has been changed.

So, having well-annotated labels in the dataset not only makes training more effective, but it also makes model evaluation more reliable. This makes sure that the system works well for real-life tasks that need to find plagiarism.

3.2 Similarity Calculations

3.2.1 Cosine Similarity

Cosine similarity is a way to find out how similar two non-zero vectors are to each other. It is widely used in machine learning and data analysis. In fact, it finds the cosine of the angle between two vectors. This gives us an idea of how far the two vectors point in the same direction, no matter how big or small they are. This word is often used in text analysis tasks, like comparing how similar two papers are, in search requests, and even in recommendation systems that fit user interests.

A similarity measure is a distance between two data objects in a dataset that is based on traits of those objects. If the distance is small, the degree of similarity will be high. If the distance is big, the degree of similarity will be low.

Cosine Similarity is defined as:

$$\cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||}$$

Where:

- $A \cdot B \rightarrow$ The dot product of two vectors
- $||A|| \rightarrow$ The magnitude (length) of vector A
- $||B|| \rightarrow$ The magnitude (length) of vector B
- $\theta \rightarrow$ The angle between the two vectors

Understanding the formula

- If $\cos(\theta) = 1$: The two vectors are identical (High Similarity)
- If $\cos(\theta) = 0$: The two vectors are completely different (No Similarity)

- If $\cos(\theta) = -1$: The two vectors are opposite

Example :

- Cosine Similarity = 1.0 \rightarrow "Music therapy is beneficial" and "Therapy using music is helpful" (very similar).
- Cosine Similarity = 0.3 \rightarrow "Music therapy helps" and "I enjoy playing sports" (low similarity).

Cosine Similarity in Text Processing :

Before calculating cosine similarity, text data needs to be converted into numerical vectors using techniques like:

- Bag of Words (BoW)
- TF-IDF (Term Frequency-Inverse Document Frequency)
- Word Embeddings (Word2Vec, BERT, GloVe)

Once the text is transformed into vectors, we can calculate cosine similarity by comparing these vectors.

Example : We have two sentences

1. Music therapy is beneficial for mental health.
2. Therapy using music helps to improve mental wellbeing.

Step 1: Convert sentences into TF-IDF vectors.

Words	Music	Therapy	Beneficial	Mental	Health	Helps	Improve	Well-being
Sentence 1	0.45	0.60	0.70	0.50	0.40	0	0	0
Sentence 2	0.50	0.55	0	0.50	0	0.60	0.70	0.40

Table 3.1: Example TF-IDF vectors

Step 2: Compute Cosine Similarity

Using the Formula:

$$\cos(\theta) = \frac{(0.45 \times 0.50) + (0.60 \times 0.55) + (0.70 \times 0) + (0.50 \times 0.50) + (0.40 \times 0) + (0 \times 0.60) + (0 \times 0.70) + (0 \times 0.40)}{\sqrt{(0.45^2 + 0.60^2 + 0.70^2 + 0.50^2 + 0.40^2)} \times \sqrt{(0.50^2 + 0.55^2 + 0^2 + 0.50^2 + 0^2 + 0.60^2 + 0.70^2 + 0.40^2)}}$$

If the result is Cosine Similarity = 0.85, it means the two sentences are 85% similar.

3.2.2 L2 Normalization

L2 Normalization (also known as Euclidean Normalization) is a mathematical approach for scaling vectors such that their length (or magnitude) equals 1 while keeping their directional information.

It is widely used in machine learning, deep learning, NLP (Natural Language Processing), and computer vision to standardize data, increase model performance, and prevent one characteristic from dominating another.

Need of L2 Normalization?

- Minimizes the dominance of large feature values in TF-IDF vectorization or word embeddings. Frequent words may have greater TF-IDF values. Normalization guarantees that each word contributes evenly to similarity computations.
- Ensures Fair Vector Comparisons: Cosine Similarity compares document similarity based on angle, not magnitude. L2 normalization ensures that document length does not influence the outcomes.
- Enhances Machine Learning Model Stability: Facilitates gradient-based optimization in deep learning, preventing big values from generating instability during training.

L2 Normalization Calculation:

L2 normalization rescales a vector by dividing each element by its L2 norm (Euclidean norm).

Formula for L2 Normalization:

$$X_{Normalized} = \frac{X}{||X||_2}$$

When $||X||_2$ is the L2 norm (Euclidean norm):

$$||X||_2 = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

Each element x_i in the vector is transformed as:

$$x'_i = \frac{x_i}{||X||_2}$$

This ensures that the sum of sequence of all elements in the vector equals 1:

$$\sum x_i'^2 = 1$$

Example of L2 Normalization:

Let's vector $X = [3,4]$

I. Calculate L2 norm:

$$||X||_2 = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

II. Normalize each element:

$$X_{normalized} = \left[\frac{3}{5}, \frac{4}{5}\right] = [0.6, 0.8]$$

The new vector points in the same direction, but its magnitude is now 1.

L2 Normalization in Cosine Similarity

L2 Normalization is often used before calculating Cosine Similarity, which determines the angle between two vectors.

- Cosine Similarity only considers the angle of vectors, not their length.
- Without normalization, lengthier documents may have higher TF-IDF values, which might influence similarity ratings.
- Normalization guarantees that text similarity comparisons are fair.

Example : Without Normalization:

- Doc 1: [3, 4]
- Doc 2: [6, 8]

Cosine Similarity:

$$\cos(\theta) = \frac{(3 \times 6) + (4 \times 8)}{\sqrt{3^2 + 4^2} \times \sqrt{6^2 + 8^2}} = \frac{18 + 32}{\sqrt{25} \times \sqrt{100}} = \frac{50}{5 \times 10} = 1.0$$

Example : With Normalization:

- Normalized Doc 1: [0.6, 0.8]
- Normalized Doc 2: [0.6, 0.8]

Now, Cosine Similarity is still 1.0, but the vector magnitudes are standardized.

L2 Normalization in TF-IDF and NLP

L2 Normalization is often used in TF-IDF vectorization to standardize text representations.

Example: TF-IDF Values Before Normalization

Word	Doc 1	Doc 2
Music	3.0	6.0
Therapy	4.0	8.0

Table 3.2: TF-IDF value before Normalization

After L2 Normalization

Word	Doc 1 (Normalized)	Doc 2(Normalized)
Music	0.6	0.6
Therapy	0.8	0.8

Table 3.3: : TF-IDF value After Normalization

3.2.3 Hybrid Similarity

In this thesis, Hybrid Similarity is employed as a comprehensive approach to compare text by integrating multiple similarity measures. Rather than depending just on one approach, it combines semantic similarity methods, including Neural Network Embeddings, SBERT, and Word2Vec, which capture deeper contextual meanings, with lexical similarity techniques, including TF-IDF and n-grams, which identify direct word matches and structural patterns. This dual technique guarantees a more exact and sophisticated identification of plagiarism, paraphrasing, and reworded text, hence improving the plagiarism detecting system's efficacy. This work intends to overcome the difficulties of identifying semantically identical but lexically changed material by using hybrid similarity, therefore improving content originality evaluation. There are two main parts to hybrid similarity:

A. Syntactic Similarity (TF-IDF)

Term Frequency-Inverse Document Frequency (TF-IDF) is used to measure the importance of words in a given text relative to a larger document collection (corpus). It focuses on how words appear across different documents and assigns higher weights to words that are significant in a specific text but less common in the entire dataset.

- **Vector Representation:** Text is transformed into numerical vectors, with each value indicating the importance of a word.
- **Cosine Similarity:** The similarity between two TF-IDF vectors is determined by measuring the angle between them—smaller angles indicate higher similarity.
- **N-gram Approach:** To detect minor textual variations, unigrams (single words) and bigrams (two-word combinations) are used.

B. Semantic Similarity (Sentence Transformer - SBERT)

While TF-IDF focuses on word-level comparisons, Sentence-BERT (SBERT) enhances similarity detection by capturing the contextual meaning of sentences. Instead of merely identifying exact word matches, SBERT represents entire sentences as high-dimensional embeddings, allowing for a deeper semantic understanding of text.

- **Sentence Embedding:** Each document is transformed into a dense numerical vector.
- **Cosine Similarity on Embeddings:** The alignment between two document embeddings is measured to determine their semantic closeness.
- **Context Capture:** SBERT effectively identifies paraphrased content, synonyms, and semantically similar expressions, making it suitable for detecting subtle textual modifications.

Hybrid Similarity: Combining TF-IDF and Semantic Similarity

To maximize accuracy, this thesis employs a **hybrid similarity approach** that balances both syntactic and semantic methods. The final similarity score is computed as:

$$\text{Hybrid Similarity Score} = (\text{TF-IDF Similarity} + \text{Semantic Similarity}) / 2$$

This approach ensures **word-level precision** through **TF-IDF**, which detects exact matches and text structure, while **SBERT** enhances **contextual understanding** by identifying paraphrased and meaning-based similarities. Combining both methods results in a more accurate and robust plagiarism detection system

3.3 Model Selection for Plagiarism Detection

3.3.1 BERT : The Foundation of MiniLM

The BERT (Bidirectional Encoder Representations from Transformers) model, which was presented by Devlin et al. (2018), is a deep bidirectional transformer model that has been pre-trained with the intention of providing contextual word embeddings. BERT, in contrast to prior natural language processing models, is capable of simultaneously capturing both left and right contexts, which has the effect of increasing its effectiveness in understanding details of actual language.

BERT is a transformer-based model that was developed by Google with the purpose of enhancing natural language processing tasks. These tasks include text categorization, question answering, named entity identification, and machine translation. It has been one of the most widely used models in modern natural language processing (NLP) research and applications, and it has also been responsible for the establishment of new standards in natural language understanding (NLU as well).

Key Characteristics of BERT

1. Bidirectional Training:

- Conventional models, such as LSTMs and GPT, analyze text in a sequential manner, moving from left to right (or vice versa), which restricts their ability to understand the meaning of whole phrases.
- On the other hand, BERT analyzes words in relation to their whole contextual surroundings, which results in a more comprehensive knowledge of the meanings of words.

2. Masked Language Modelling (MLM):

- The pre-training step of BERT involves the random masking of certain words inside a sentence so that it may learn to anticipate such words.
- It is because of this that the model needs to develop a comprehensive contextual representation of words, rather than relying just on symbols that are next to one another.

3. Next phrase Prediction (NSP):

- By being trained on pairs of phrases, BERT is able to learn the capacity to predict if a forthcoming phrase logically follows the one that came before it.

- The purpose of this project is to improve BERT's ability to perceive sentence connections, which is necessary for responding to questions, summarizing information, and producing discourse.

4. BERT is pre-trained on large datasets:

- Books Corpus, which has 800 million words, and Wikipedia, which contains 2.5 billion words.
- Because it has received so extensive pre-training, BERT is able to generalize well across a wide range of NLP tasks.

5. Fine-Tuning Capability:

- BERT may be improved for certain natural language processing tasks by utilizing supplemental labeled data once it has been pre-trained.
- When compared to training from the ground up, fine-tuning BERT often requires less resources; yet, it still makes use of its superior contextual understanding.

Evolution of BERT: The Need for Smaller, Faster Models

BERT's limitations led to the development of more efficient models that retain BERT's powerful language understanding while being more computationally viable. Some of these models include:

- DistilBERT (40% smaller, 60% faster, retains 97% of BERT's performance)
- TinyBERT (lightweight version optimized for mobile and edge devices)
- ALBERT (parameter-sharing approach reducing model size while improving efficiency)
- MiniLM (distilled model maintaining near-BERT performance at a fraction of the size)

These optimized versions have helped make BERT more accessible for real-world applications where speed and efficiency are critical.

3.3.2 (LLM) Large Language Model Distillation

A Large Language Model (LLM) is a neural network that uses deep learning and was taught on a huge amount of text data to understand, create, and handle human language. LLMs use transformer designs, which lets them do great at many natural language processing (NLP) jobs, like creating text, translating it, summarizing it, and answering questions. Some well-known LLMs are GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), and T5 (Text-to-Text Transfer Transformer). They all work at the highest level in a wide range of NLP tasks.

Key Characteristics of LLMs

3. **Massive Pre-training:** LLMs are trained on huge amounts of data, like books, papers, Wikipedia, and online material, which lets them generalize across many areas.
4. **Transformer-Based Architecture:** To figure out how words relate to each other in context, they use self-attention systems and deep neural networks.
5. **Transfer Learning:** LLMs can be fine-tuned on specific tasks with small datasets after being pre-trained on huge amounts of unstructured text.
6. **Multimodal Capabilities:** Some new LLMs, like GPT-4 and PaLM, can handle pictures, code, and music alongside text to make them more useful.
7. **Generative Skills:** LLMs can write text that sounds like it was written by a person, answer complicated questions, and do thinking tasks, which makes them necessary for AI-driven apps.

Process of Model Distillation

Model distillation follows a structured process that involves training a large teacher model and extracting its knowledge to train a smaller student model. The typical process includes:

Step 1: Train a Large Model (Teacher Model)

- The original large model (BERT, GPT, T5) is trained on massive datasets using self-supervised learning.
- This model captures complex language representations and patterns.

Step 2: Extract Knowledge from the Teacher Model

- The teacher model generates soft labels or probabilistic outputs (logits) for training data.
- In some methods, attention distributions, hidden states, or intermediate layer representations are also transferred.

Step 3: Train a Smaller Model (Student Model)

- A smaller model (DistilBERT, TinyBERT, MiniLM) is trained to mimic the teacher's behavior.
- Instead of training from scratch, the student learns directly from the teacher's outputs.
- The goal is to achieve similar accuracy with fewer parameters.

Types of Knowledge Transfer in Distillation

There are several ways to transfer knowledge from the teacher model to the student model:

Logit-Based Distillation (Soft Targets)

- The teacher model provides soft probability distributions rather than hard labels.

- The student model is trained to match these probabilities using KL Divergence Loss.

Feature-Based Distillation (Intermediate Layer Knowledge)

- Intermediate hidden states from the teacher model are transferred to the student model.
- Helps capture the depth of word representations while keeping the model small.

Attention-Based Distillation

- The student model learns from the attention weights of the teacher model.
- Effective for retaining contextual dependencies between words.

Multi-Task Distillation

- The student model is trained on multiple tasks using the same distilled knowledge.
- Example: A distilled BERT model trained on both sentiment analysis & question answering.

Popular Distilled Models

Several smaller models have been successfully developed using model distillation techniques:

Distilled Model	Original Model	Size Reduction	Speedup
DistilBERT	BERT	40%	60%
TinyBERT	BERT	75%	75%
MiniLM	BERT	60%	50%
ALBERT	BERT	90% (via parameter-sharing)	70%

Table 3.4: Popular Distilled Models

These models are significantly lighter and faster, making them ideal for production use cases.

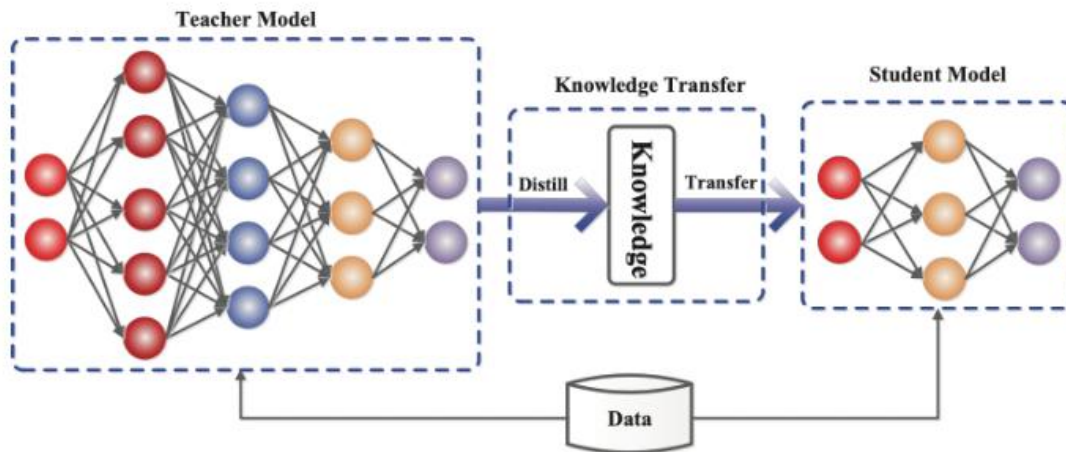


Figure 3.3: Student-Teacher Model

3.3.3 MiniLM Model

MiniLM (Wang et al., 2020) is a streamlined variant of BERT optimised for accelerated inference with a reduced number of parameters. MiniLM is intended to condense extensive pre-trained Transformer models, such as BERT or RoBERTa, into more compact and efficient variants with little performance degradation. This is accomplished using a method known as knowledge distillation, whereby a smaller model (the pupil) learns to emulate the behaviour of a bigger, well-trained model (the instructor).

Key Factors in MiniLM Distillation

1. **Self-Attention Mechanism:** This part of Transformer models lets the model decide which words in a sentence are the most important when it encodes a certain word. This feature is very important for finding environmental connections in the data.
2. **Distilling Deep Self-Attention:** MiniLM doesn't focus on taking information from all the layers of the teacher model; instead, it only looks at the self-attention modules in the last Transformer layer of the teacher. The student model is taught to copy this self-attention behaviour's which helps it quickly gather important environmental information.
3. **Value-Relation Transfer:** MiniLM doesn't just copy attention patterns (like the links between searches and keys); it also brings the idea of moving the links between values in the self-attention system. This uses the scaled dot-product between value vectors to help the student model understand more complex conceptual connections..
4. **Teacher Assistant Strategy:** To bridge the gap between a large teacher model and a much smaller student model, an intermediate-sized model, termed the teacher assistant, is used. There are two steps in the distillation process: first, the teacher tells the teacher assistant what to do, and then the teacher assistant helps the student. Using this staged method makes it easier to share information.

Feature	BERT-base	MiniLM
Layers	12	6
Parameters	110M	33M
Training Speed	Slow	Faster
Inference Speed	High Latency	Optimized for Efficiency

Figure 3.4: MiniLM Model

MiniLM distills the self-attention layers, making it lighter and faster while preserving accuracy.

3.3.4 SBERT: Sentence-BERT

SBERT (Sentence-BERT) is a more advanced version of BERT (Bidirectional Encoder Representations from Transformers) that was made to do jobs that compare sentences. BERT has changed Natural Language Processing (NLP) by giving contextually rich word embeddings. However, it has trouble understanding whole sentences and needs pairwise comparisons, which are very time-consuming to compute.

SBERT, created by Reimers and Gurevych (2019), gets around these problems by improving BERT with Siamese and Triplet Networks to create fixed-length sentence embeddings. These embeddings make it possible to compare things quickly and on a large-scale using similarity measures like cosine similarity. This makes SBERT a good tool for tasks like semantic search, text grouping, paraphrase detection, and copy detection.

The most important new thing about SBERT is that it does away with the need for BERT's computationally expensive pairwise comparisons. SBERT doesn't run each pair of sentences through the model separately; instead, it handles each sentence on its own and creates embeddings that can be directly compared. This makes sentence similarity calculations much faster, which makes SBERT a powerful tool for NLP tasks that need to compare and retrieve big amounts of text.

Even though BERT understands context very well, it is not best for sentence matching tasks for the following reasons:

- Pairwise Comparison is Hard to Do on a Computer: BERT needs to do a separate forward pass for each sentence pair, which makes it quadratic in complexity ($O(n^2)$) for big datasets.
- No Direct Sentence Embeddings: BERT yields contextualised word embeddings instead of direct sentence embeddings.
- Not Made for Quick Retrieval: BERT-based models have to compare each set of sentences one at a time, which is slow for large-scale tasks like semantic search, finding duplicates, and grouping.

E.g. Imagine looking at 10,000 words side by side to see which ones are most alike:

- BERT: Needs $(10,000 \times 10,000) / 2 = 50$ million comparisons.

- SBERT: Makes one embedding for each line and quickly compares them using cosine similarity.

3.3.5 all-MiniLM-L6-v2: A Sentence Transformer Model

What is all-MiniLM-L6-v2?

all-MiniLM-L6-v2 is a sentence transformer model based on MiniLM with 6 layers Version 2, designed for semantic similarity and retrieval tasks.

Why all-MiniLM-L6-v2?

- Lightweight: Faster inference.
- Efficient Sentence Embeddings: Optimized for semantic search.
- Pretrained & Ready to Use: Available in sentence-transformers.

Model Architecture

- Base Model: The model is based on Microsoft's MiniLM architecture, specifically the MiniLM-L6-H384-uncased model.
- Layers: It comprises 6 Transformer layers, which is denoted by "L6" in its name.
- Hidden Size: Each layer has a hidden size of 384 dimensions.
- Output Embeddings: The model maps input sentences or paragraphs to a 384-dimensional dense vector space.

Training and Fine-Tuning

- Pre-training: Initially, the model was pre-trained on large-scale datasets using self-supervised learning objectives.
- Fine-Tuning: It was further fine-tuned on over 1 billion sentence pairs using a contrastive learning objective. In this setup, given a sentence from a pair, the model predicts which out of a set of randomly sampled sentences was actually paired with it in the dataset.

3.4 Model Fine Tuning Process

Fine-tuning is a transfer learning technique where a pre-trained model is trained further on a specific dataset to adapt it to a specialized task. In this case, we use MiniLM-L6-v2, a compact Transformer-based model trained for sentence embeddings, and fine-tune it on the PAN Dataset for plagiarism detection.

Overview of the Fine-Tuning Process

The fine-tuning process involves:

- Loading the PAN dataset (Suspicious and Source documents).
- Preprocessing the dataset (Extracting text, tokenization).
- Creating a dataset class to handle input pairs.
- Defining a model architecture with MiniLM.
- Training the model on plagiarism detection.
- Evaluating the model's performance.
- Saving the model for deployment.

Fine-Tuning all-MiniLM-L6-V2 for Plagiarism Detection on the PAN Dataset

This section details the methodology used to fine-tune all-MiniLM-L6-V2 on the PAN dataset, a corpus specifically designed for plagiarism detection. The primary objective is to enable the model to identify textual similarities between suspicious documents and their potential source documents. This is accomplished through fine-tuning a pre-trained language model, optimizing it for cosine similarity-based classification.

1. Introduction to Fine-Tuning

Fine-tuning refers to the process of adapting a pre-trained language model to a specific task by training it on a domain-specific dataset. In this study, all-MiniLM-L6-V2, a lightweight transformer-based model, was fine-tuned using the PAN dataset to improve its ability to detect plagiarism. Instead of training a model from scratch, fine-tuning leverages pre-existing knowledge from a pre-trained model, significantly reducing computational costs and training time.

2. Dataset Overview

The dataset used for fine-tuning was sourced from the PAN plagiarism detection corpus, which contains pairs of suspicious and source documents.

The dataset is structured as follows:

- Suspicious documents: These texts are suspected of containing plagiarized content.
- Source documents: These are the original texts from which plagiarism may have occurred.

- Annotation files (XML format): Each suspicious document has an associated XML file detailing the plagiarized sections and their corresponding source documents.

Dataset Preprocessing Steps

To utilize the dataset effectively, the following preprocessing steps were performed:

1. Extracting and loading documents: The dataset was provided in compressed format (ZIP), containing suspicious and source documents, which were extracted for processing.
2. Parsing XML annotations: The XML files were analyzed to identify which sections of the suspicious documents were plagiarized and which source documents they corresponded to.
3. Generating labeled data: Each suspicious document was matched with a corresponding source document, with a binary label:
 - 1: Indicates plagiarism (strong textual similarity exists).
 - 0: Indicates no plagiarism (randomly paired documents).
4. Handling missing values: Since not all suspicious documents had clear source mappings, missing values were addressed by assigning "unknown" sources to non-plagiarized cases.

3. Model Selection: all-MiniLM-L6-V2

MiniLM (Minimal Language Model) is a transformer-based architecture optimized for sentence embeddings. It provides efficient text similarity detection while maintaining low computational requirements. The MiniLM-L6-V2 variant was chosen for this study due to:

- Compact size: It is a small yet powerful model optimized for similarity tasks.
- Pre-trained embeddings: The model was initially trained on millions of sentences for semantic search, making it a strong candidate for plagiarism detection.

Modification for Fine-Tuning

The original MiniLM model was modified to:

- Extract sentence-level embeddings from documents.
- Compute cosine similarity between suspicious and source text pairs.
- Use contrastive learning objectives to differentiate plagiarized vs. non-plagiarized text pairs.

4. Tokenization and Input Representation

To prepare text for the model, each suspicious and source document pair underwent tokenization using the MiniLM tokenizer:

- Padding: Ensured all sentences had the same length to optimize batch processing.

- Truncation: Removed extra words exceeding the model's token limit (512 tokens).
- Attention masks: Indicated which tokens were valid (to differentiate real text from padding).

Each document pair was represented as:

1. Suspicious Text Input: Tokenized representation of the suspicious document.
2. Source Text Input: Tokenized representation of the potential source document.
3. Binary Label: 1 (Plagiarism) or 0 (No Plagiarism).

5. Training Strategy

The fine-tuning process involved training the model on labeled plagiarism data to maximize its ability to detect textual similarity. The key training configurations were:

- Loss Function: Mean Squared Error (MSE Loss) was used to compare the cosine similarity score between document embeddings and the actual plagiarism labels.
- Optimization Algorithm: AdamW optimizer was selected due to its stability in fine-tuning large-scale language models.
- Learning Rate: A low learning rate (1e-6) was used to prevent overfitting while fine-tuning.
- Batch Size: A batch size of 16 was selected to balance performance and memory usage.
- Number of Training Epochs: The model was fine-tuned over 5 epochs, allowing it to generalize well without excessive computational cost.

6. Model Training

During training, the model was provided with pairs of documents along with their similarity labels. The MiniLM encoder was used to extract embeddings for each text:

1. Sentence Embedding Extraction:
 - The MiniLM model processes both texts and generates dense vector embeddings.
 - These embeddings capture semantic meaning beyond just lexical similarity.
2. Cosine Similarity Computation:
 - A cosine similarity function was applied to measure the closeness between the two embeddings.
 - A similarity score closer to 1 indicates high similarity (potential plagiarism).
 - A similarity score closer to 0 indicates dissimilarity.

3. Loss Optimization:

- The MSE loss function was used to ensure that the model outputs higher similarity scores for plagiarized text pairs and lower scores for non-plagiarized pairs.
- To improve model generalization, dropout layers were introduced to prevent overfitting.

7. Evaluation Metrics

To assess the performance of the fine-tuned model, standard classification metrics were used:

1. Accuracy: Measures how often the model correctly classifies plagiarism cases.
2. Precision: Evaluates how many of the predicted plagiarism cases were actually correct.
3. Recall: Measures how well the model identifies actual plagiarism cases.
4. F1 Score: Provides a balance between precision and recall.
5. Confusion Matrix: Visualizes the classification performance, showing true positives, false positives, true negatives, and false negatives.

The evaluation process involved:

- Generating embeddings for validation samples.
- Computing cosine similarity scores between document pairs.
- Applying a threshold (0.5) to classify pairs as plagiarized or non-plagiarized.
- Comparing predictions against true labels to compute evaluation metrics.

Fine-Tuning Approaches:

I. Direct Fine-Tuning with MiniLM:

This approach follows a low-level, explicit fine-tuning process where all key steps data preprocessing, embedding extraction, similarity computation, and optimization are explicitly defined.

Step-by-Step Process:

1. Dataset Preparation

- Suspicious and source documents are extracted from ZIP files.
- XML annotations are parsed to map plagiarized text pairs.
- The dataset is structured into input pairs: (suspicious document, source document, label).

2. Tokenization and Input Representation

- AutoTokenizer from transformers is used to tokenize both suspicious and source texts.
- Each document is tokenized into input IDs, attention masks, and truncated to 512 tokens.
- Tokenized inputs are explicitly managed before passing into the model.

3. Model Architecture

- AutoModel (all-MiniLM-L6-v2) is loaded directly for fine-tuning.
- Mean pooling is applied on transformer outputs to extract sentence embeddings.
- A dropout layer (0.3) is added for regularization and overfitting prevention.

4. Training Pipeline

- The model is trained using cosine similarity loss (to compare document pairs).
- Mean Squared Error (MSE) loss is used to fine-tune similarity scores based on ground truth labels.
- The optimizer used is AdamW with a low learning rate (1e-6).
- Automatic Mixed Precision (AMP) training is applied to reduce GPU memory usage.

5. Fine-Tuning Strategy

- The model is trained for 5 epochs with a batch size of 16.
- During training, sentence embeddings are extracted explicitly for both documents.
- Cosine similarity is computed between suspicious and source embeddings.
- The model learns to maximize similarity for plagiarized cases (label = 1) and minimize similarity for non-plagiarized pairs (label = 0).

6. Evaluation and Model Saving

- After training, the model is evaluated using accuracy, precision, recall, F1-score, and a confusion matrix.
- The model is saved using PyTorch's torch.save(), ensuring flexibility in future deployment.

II. SentenceTransformer-Based Fine-Tuning:

This approach simplifies fine-tuning by leveraging the SentenceTransformer library, which abstracts many internal processes. Instead of directly handling embeddings and tokenization manually, SentenceTransformer provides built-in methods to automate these steps.

Step-by-Step Process:

1. Dataset Preparation

- Similar to direct Fine-Tuning with MiniLM, the PAN dataset is extracted from ZIP files.
- XML annotations are parsed to create labeled text pairs (suspicious, source, label).
- The dataset is split into train and validation sets for training.

2. Sentence Encoding Instead of Tokenization

- Instead of manually tokenizing inputs, SentenceTransformer directly encodes documents.
- The `SentenceTransformer.encode()` method converts each text into a high-dimensional vector representation.
- Tokenization, truncation, and padding are handled internally.

3. Model Architecture

- The all-MiniLM-L6-v2 model is loaded as a SentenceTransformer model.
- Unlike direct Fine-Tuning with MiniLM, no additional dropout layers or embedding modifications are applied.
- The model generates sentence-level embeddings automatically.

4. Training Pipeline

- The model is trained using cosine similarity loss.
- Instead of explicitly computing embeddings, SentenceTransformer automatically calculates similarity scores.
- The optimizer used is AdamW with a learning rate of $1e-6$.

5. Fine-Tuning Strategy

- Suspicious and source texts are encoded into vector embeddings.
- Cosine similarity is computed between both embeddings.
- Loss function optimizes the model to increase similarity for plagiarized cases.

- Unlike direct Fine-Tuning with MiniLM, this process does not manually handle embedding pooling.

6. Evaluation and Model Saving

- The model is evaluated using accuracy, precision, recall, and F1-score.
- Instead of saving using PyTorch, the SentenceTransformer model is saved using the built-in `.save()` method.
-

This study successfully fine-tuned all-MiniLM-L6-V2 to enhance its plagiarism detection capabilities. By leveraging pre-trained sentence embeddings and optimizing them for cosine similarity classification, the model effectively differentiates between plagiarized and non-plagiarized text pairs. The fine-tuned model can now be deployed for real-world applications in academic and research integrity verification systems.

Chapter 4 : Phase II

4.1 Dataset Selection

4.3.1 PAWS Dataset

For the paraphrase generation task in this study, the PAWS (Paraphrase Adversaries from Word Scrambling) dataset was selected due to its high-quality, challenging sentence pairs, which emphasize structural and lexical similarity while maintaining distinct meanings. PAWS is particularly well-suited for training deep learning models like BART (Denosing Autoencoder for Pretraining Sequence-to-Sequence Models) because it contains adversarial paraphrase pairs that require sophisticated contextual understanding rather than simple word-level transformations. The dataset includes human-verified paraphrase and non-paraphrase pairs, making it an ideal choice for enhancing the robustness and generalization of the model in distinguishing between valid rephrasings and semantically altered text. By leveraging PAWS, this research aims to improve the paraphrasing capabilities of the generative AI model while minimizing semantic drift and preserving the original intent of the input text. This ensures that paraphrase-based plagiarism detection is more effective in distinguishing between genuine paraphrasing and text duplication.

Structure of the PAWS Dataset

The PAWS dataset is designed to include highly similar yet challenging paraphrase and non-paraphrase pairs, making it suitable for training models on paraphrase generation and detection. The dataset consists of the following key components:

1. **Sentence Pairs:** Each sample in the dataset contains two sentences (sentence1 and sentence2) that are either paraphrases or non-paraphrases.
2. **Label:** Each sentence pair is labeled as either:
 - 1 (Paraphrase): If the sentences are semantically equivalent despite structural variations.
 - 0 (Non-paraphrase): If the sentences have minor differences that alter their meaning.
3. **Adversarial Examples:** The dataset includes adversarial sentence pairs, generated using word swapping and back-translation to make paraphrase detection more challenging. This ensures that models learn to distinguish subtle semantic differences rather than relying on simple lexical overlap.
4. **Human-Annotated Subset:** A portion of the dataset is manually labeled to ensure high-quality ground truth for training and evaluation.
5. **Versions of PAWS:** The dataset is available in two major versions:
 - PAWS-Wiki: Constructed from Wikipedia, with adversarial sentence pairs generated using word scrambling and back-translation.
 - PAWS-QQP: Derived from the Quora Question Pairs (QQP) dataset, focusing on question paraphrases.

6. **Data Format:** The dataset is typically provided in a tabular format (CSV or TSV) and contains the following columns:
- sentence1: First sentence in the pair.
 - sentence2: Second sentence in the pair.
 - label: Binary label (1 for paraphrase, 0 for non-paraphrase).

This structured dataset provides a strong foundation for training the BART model for paraphrasing by ensuring exposure to diverse and linguistically complex sentence transformations. Its use in this research allows for improved model performance in paraphrase-based plagiarism detection, ensuring the generated paraphrases maintain the original intent of the text while introducing linguistic variations.

4.3.2 Custom Dataset

In addition to the PAWS dataset, a custom dataset was developed and used to evaluate the performance of different paraphrase generation models. This dataset consists of manually curated sentence pairs, designed to assess the quality of paraphrase generation in a real-world setting. Unlike PAWS, which includes adversarial examples generated through word scrambling and back-translation, the custom dataset focuses on natural language variations, synonym replacements, and structural modifications. This makes sure that the review is based on real-life situations where paraphrasing involves small but important changes to language, rather than examples that are meant to be negative.

Experimental Setup

To evaluate paraphrasing performance, three different models were tested on the custom dataset:

1. **Pretrained BART**
 - The original BART (facebook/bart-large) model was tested without fine-tuning to assess its out-of-the-box paraphrasing capabilities.
2. **Fine-tuned BART on PAWS**
 - The BART model was fine-tuned using the PAWS-Wiki (labeled_final) dataset to enhance its ability to handle challenging paraphrase pairs.
 - Fine-tuning aimed to improve semantic preservation, ensuring the generated paraphrases retain the original meaning while modifying structure and vocabulary.
3. **DeepSeek Model (deepseek-ai/deepseek-r1-distill-qwen-7b)**
 - The DeepSeek R1 Distill Qwen 7B model was also tested to compare its paraphrase generation capabilities with BART.

- This model is a distilled version of a large-scale transformer, optimized for efficient text generation while maintaining high-quality language understanding.

Custom Dataset Structure

Unlike PAWS, which contains sentence pairs with explicit paraphrase labels, the custom dataset consists of single, independent sentences. Each sentence serves as an input to the paraphrase models, which then generate multiple paraphrased versions.

- Sentence → A single sentence without predefined paraphrase pairs or labels.
- Model Output → The paraphrased version(s) generated by each model.

4.2 Transformer

Natural Language Processing (NLP) made a big step forward with the Transformer model (Vaswani et al., 2017). It added the self-attention mechanism, which lets models focus on the most important words in long text sequences. Traditional Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, which had trouble with disappearing gradients, were not as good as this one.

Before Transformers came out, RNNs and LSTMs were the main types of neural networks used in NLP models. However, these models had some problems, such as:

- Slow Training: RNNs had to understand words one at a time, which took a lot of time and computing power.
- Vanishing Gradient Problem: The models had trouble with long-range dependencies, which made it hard to figure out how words that were far apart in a series were related to each other.
- Limited Parallelization: RNNs can't be successfully parallelized on GPUs because they work in a sequential way.

Transformers do not use sequential processing like RNNs. Alternatively, they:

- Process the full phrase at once, making training and inference considerably more efficient.
- Use self-attention to concentrate on key words, regardless of where they occur in a phrase.
- Take use of parallel processing, which makes them ideal for large-scale NLP workloads.

Key Transformer-Based Models:

- GPT (Generative Pre-trained Transformer) – Generates human-like text, making it useful for chatbots, content creation, and language translation.
- BERT (Bidirectional Encoder Representations from Transformers) – Reads text in both directions (left to right and right to left), improving its ability to understand context.
- RoBERTa (Robustly Optimized BERT Pretraining Approach) – A faster and more efficient version of BERT with better accuracy.
- DistilBERT – A smaller and faster version of BERT, designed to reduce computational costs while maintaining high performance.

Transformer Innovations:

I. Positional Encoding.

Traditional NLP models read text word for word in order. However, Transformers process all words simultaneously. To guarantee that the model understands the sequence of words, it employs positional encoding, which assigns a unique number to each word depending on its place in a phrase.

For example, in the phrase "The cat sat on the mat," positional encoding assists the model in understanding that "The" comes before "cat" and "mat" is the last word.

II. Self-Attention Mechanism.

Instead of reading words in sequence, like humans do, the Transformer examines all words at once and chooses which ones are most significant for interpreting the meaning.

For example, in the phrase "She bought a book, and she loved it," the model employs self-attention to understand that "it" relates to "book", despite the presence of other words in between

Transformer Architecture:

There are two main parts to the Transformer:

- Encoder reads and processes the text that is given it, turning it into a set of meaningful numbers (embeddings).
- Decoder: This part of the system takes the information that was handled by the encoder and makes new text, answers questions, or sorts of data into groups.

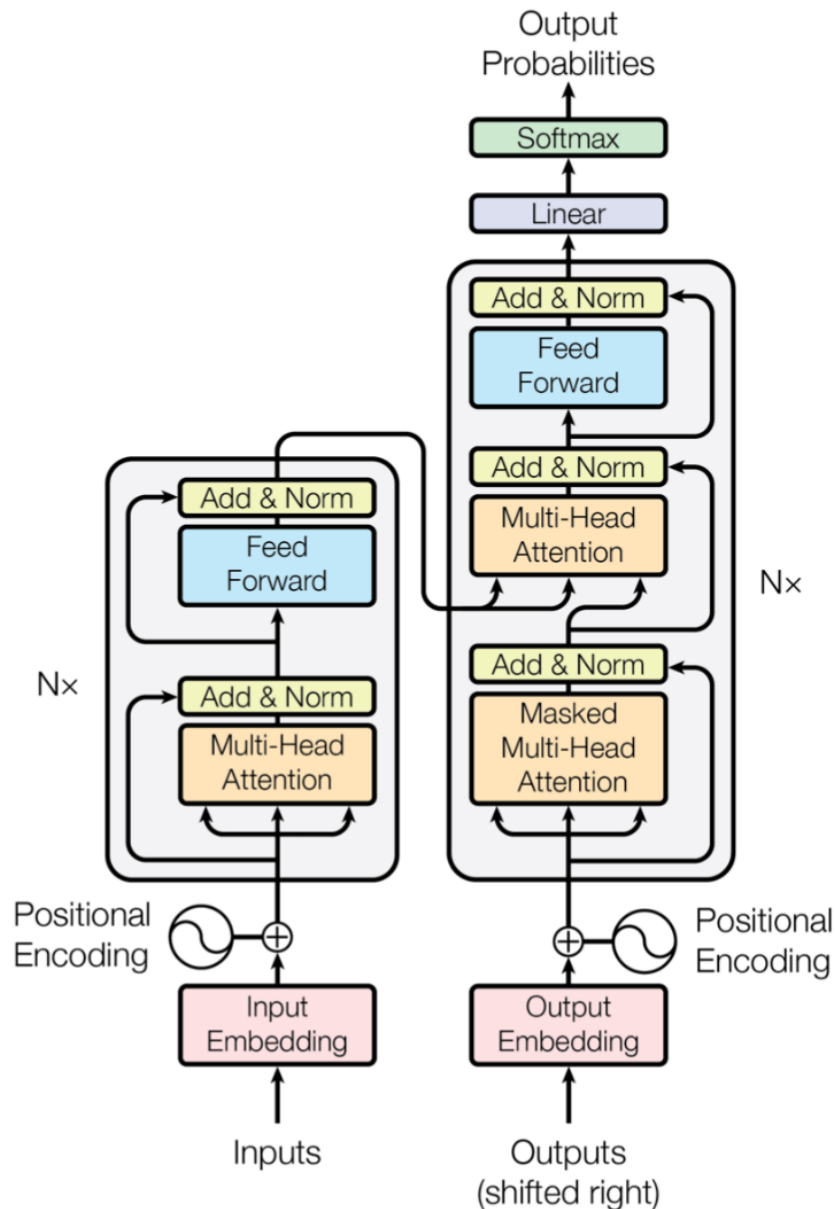


Figure 4.1: Transformer Architecture (Vaswani et al., 2017).

4.2.1 Encoder

The encoder is a key component of the Transformer model, processing input sequences and producing context-aware representations. It transforms raw input tokens (words, subwords, or characters) into high-dimensional numerical representations that convey semantic meaning. These embeddings allow a variety of downstream Natural Language Processing (NLP) activities, such as text categorization, similarity detection, machine translation, summarization, and named entity recognition (NER).

Traditional NLP models, such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, processed words sequentially, restricting their capacity to grasp connections between distant words in sentences. Transformers address this restriction by using an encoder that processes all input words concurrently with self-attention.

This enables the encoder to:

- Understand the context of each word by evaluating the whole phrase at once.
- Recognize long-range dependencies and grasp word connections regardless of distance.
- Process inputs in parallel, resulting in much faster calculations than sequential models.

Encoder Architecture:

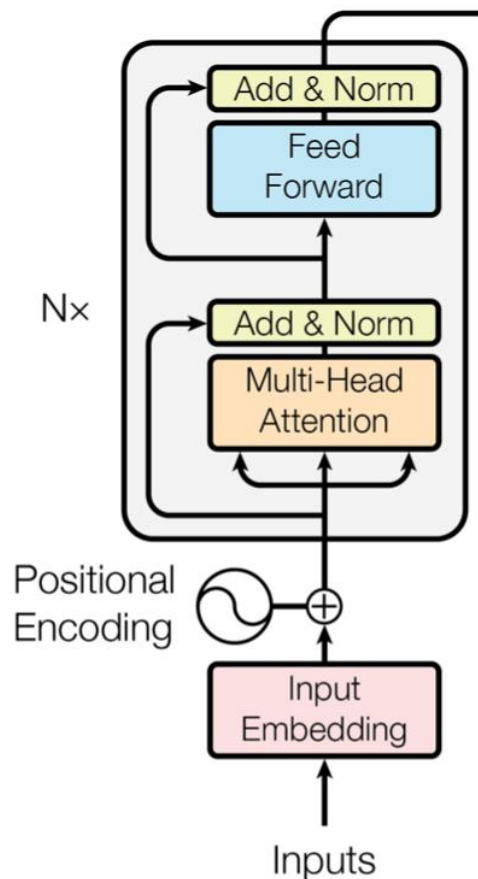


Figure 4.2: Encoder

The encoder consists of multiple identical layers stacked N times, with each layer containing these sub-modules:

- Input Embedding
- Positional Encoding
- Multi-Head Self-Attention
- Add & Norm (Residual Connection & Layer Normalization)
- Feed-Forward Neural Network (FFN)
- Add & Norm (Residual Connection & Layer Normalization)

Each of these components serves an essential function in converting raw text into context-aware numerical representations.

4.2.1.1 Input Embedding

Input embedding consists of Tokenization, Vectorization and word embeddings and their types.

1.1. Tokenization

In Natural Language Processing (NLP), tokenization is a basic technique that includes breaking down a stream of text into smaller units called tokens. Tokens are the units that are used in tokenization. In accordance with the amount of granularity that is necessary, these tokens might be anything from single letters to whole sentences or phrases. By converting text into these manageable chunks, machines can more effectively analyze and understand human language. Through the use of punctuation and spaces, this technique divides text into individual words. Because it simplifies the process of understanding and processing language for machines, tokenization is an essential procedure because it breaks down complicated material into smaller chunks. In most cases, the tokens that are produced are used as input for further processing processes, such as vectorization, which involves transforming the tokens into numerical representations that may be utilized by machine learning models.

Example:

- Input → "A dog barked loudly at night."
- Tokenization → ["A", "dog", "barked", "loudly", "at", "night"]

1.2. Sub- word Tokenization

Sub-word tokenization is an essential method that is used in modern natural language processing models, particularly Transformer-based models like as BERT, GPT, BART, and T5. In contrast to the conventional method of word tokenization, which divides text into individual words, the sub-word tokenization method divides words into more manageable and significant components.

Through the recognition of word roots, prefixes, and suffixes, this method assists in the reduction of vocabulary size, the handling of unfamiliar words, and the preservation of semantic meaning over time.

The need of sub-words

- Handling Out-of-Vocabulary (OOV) Words
 - The conventional word-based approaches are ineffective especially when faced with unique or uncommon terms.
 - As an example, a model is unable to interpret the phrase "happiest" if it has never encountered it before.
 - Tokenization of subwords guarantees that the model will continue to learn certain components of the term.
- Reducing Vocabulary Size
 - A word-based vocabulary requires hundreds of thousands of unique words.
 - Sub-word tokenization allows a smaller vocabulary, making models more efficient.

- Recognizing Word Relationships
 - example → “unhappiness” → [“un”, “happiness”]

2.1. Vectorization

Text data is transformed into numbers so that computers, specifically machine learning and deep learning models, can understand and handle it. This is called vectorization. Vectorization is an important step in making text information machine-readable because computers don't naturally understand human language. Words and lines make sense to humans based on their meaning, structure, and the rest of the phrase. Computers, on the other hand, work with numbers and logical patterns. As much of the sense of words as possible is kept when they are turned into numbers through vectorization.

Types of Vectorization Methods

Text can be turned into numbers using a number of different methods. These methods range from simple ones that use word frequency to more complex ones that use context.

Basic Count-Based Methods: These methods focus on counting words or their importance across documents.

A. Bag-of-Words (BoW)

The Bag-of-Words model represents text as an unordered collection (bag) of words, ignoring grammar and word order. It constructs a word-document matrix, where:

- Rows represent documents.
- Columns represent unique words.
- Each cell contains the frequency of a word in a document.
- Example: Consider two documents:
 - "Apple is a fruit, and I like apple."
 - "Banana is also a fruit."

Words	Apple	Banana	Fruit	Is	Like	Also	And
Doc 1	2	0	1	1	1	0	1
Doc 2	0	1	1	1	0	1	0

Table 4.1: Example BoW

B. Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF improves upon BoW by assigning different importance (weights) to words based on their frequency and uniqueness.

- Term Frequency (TF): Measures how often a word appears in a document.

$$TF = \frac{\text{Number of times the word appears in a document}}{\text{Total words in the document}}$$

- Inverse Document Frequency (IDF): Measures how unique a word is across multiple documents.

$$IDF = \log \left(\frac{\text{Total words in the document}}{\text{No. of documents containing the word}} + 1 \right)$$

- TF-IDF Score: A word's importance is determined by multiplying $TF \times IDF$

$$TF - IDF = TF \times IDF$$

Example: Consider three documents:

- "Apple is a fruit."
- "Banana is also a fruit."
- "Apple and banana are delicious."

TF Calculation:

- "Apple" appears once in Doc 1 (out of 4 words) → $TF = 1/4 = 0.25$
- "Banana" appears once in Doc 2 (out of 5 words) → $TF = 1/5 = 0.20$

IDF Calculation:

- "Apple" appears in two documents → $IDF = \log(3/2) = 0.18$
- "Banana" appears in two documents → $IDF = \log(3/2) = 0.18$

TF-IDF Scores:

- Apple (Doc 1) = $0.25 \times 0.18 = 0.045$
- Banana (Doc 2) = $0.20 \times 0.18 = 0.036$

3.1. Word Embedding

When it comes to natural language processing (NLP), which is a subfield of machine learning, word embeddings are an essential point of discussion. Textual data, which machine learning algorithms are unable to interpret, is transformed into a numerical form that can be understood by these algorithms via the use of word embeddings. In addition, they may be used to capture the core of the context in which words are employed, as well as the semantic and syntactic similarities between words, as well as the relationship between words.

Word Embedding Methods

These methods map words into dense vector spaces, capturing semantic relationships.

a) Word2Vec

- It is Introduced by Google.
- Creates dense word embeddings by predicting words in a given context.
- Uses two architectures:
 - CBOW (Continuous Bag-of-Words): Predicts a target word given surrounding word.

- Skip-gram: Predicts surrounding words given a target word.
- Example: "King" - "Man" + "Woman" \approx "Queen"

b) GloVe (Global Vectors for Word Representation)

- Developed by Stanford.
- Uses co-occurrence statistics to learn word embeddings.
- Embeddings capture linear relationships between words.

c) FastText

- Developed by Facebook.
- Improves Word2Vec by considering subword information (character n-grams).
- Helps in handling out-of-vocabulary (OOV) words.
- Example: "unhappiness" \rightarrow ["un", "hap", "pine", "ness"]

Contextualized Word Embeddings

These methods generate dynamic embeddings, meaning a word's representation changes based on its context.

(a) BERT (Bidirectional Encoder Representations from Transformers)

- Developed by Google.
- Uses self-attention and transformers to understand context.
- Word embeddings vary based on sentence structure.

Examples:

- "He went to the bank to withdraw money." (bank = financial institution)
- "He sat on the bank of the river." (bank = riverbank)

(b) GPT (Generative Pretrained Transformer)

- Used for text generation tasks.
- Generates word embeddings that adapt dynamically based on context.

4.2.1.2 Position Embedding

A fundamental idea in transformers and other deep learning models handling sequential input like text, audio, and time-series data is position embedding—or positional encoding. Position embeddings are included to provide positional information into the model as transformers lack a built-in knowledge of order unlike RNNs or CNNs with sequential designs.

Transformers such as BERT and GPT handle input sequences in parallel using self-attention techniques. Transformers handle input tokens as a "bag of words" without order unlike recurrent neural networks (RNNs) or convolutional neural networks (CNNs), which naturally learn sequence order due to their design.

However, in many real-world applications, the order of elements in a sequence matters:

- In natural language processing (NLP), "I love music" is different from "Music love I."
- In speech processing, the order of audio frames defines meaning.
- In time-series forecasting, past events influence future predictions.

Position embedding solves this by encoding position information numerically and adding it to token embeddings.

Types of Position Embeddings: There are two main types of position embeddings:

A. Absolute Positional Embedding

- Each position in the sequence is assigned a unique vector.
- This vector is learned during training.
- It allows the model to differentiate between positions but does not directly capture relationships between them.

B. Relative Positional Embedding

- Instead of encoding absolute positions, the model learns the relative position between tokens.
- More useful in tasks where relationships between words matter more than their absolute positions.

Implementation of position embedding

Before the token embeddings are handled by the transformer, position embeddings are vectors that are added to them. This makes sure that positional information is included in the self-attention process.

A. Learned Positional Embeddings

- Just like word embeddings, position embeddings can be trained.
- While the model is being trained, these embeddings are learned by the model, and then they are optimized for the particular job.
- Used in models such as the BERT model.

B. Sinusoidal Positional Encoding Used in the Transformer

- Instead of learning embeddings, a fixed mathematical function assigns position embeddings.
- Uses sine and cosine functions of different frequencies to encode position.

Mathematical Formula:

$$PE(pos, 2i) = \sin(pos/10000^{2i/d_{model}})$$

$$PE(pos, 2i + 1) = \cos(pos/10000^{2i/d_{model}})$$

Where:

- pos = tokens position in the sequence.
- i = dimension index.
- d_{model} = total embedding dimension (e.g. 512 in transformers).
- 10000 = a constant to normalize value.

Example : Mathematical Calculation of Position Embeddings Using a Sentence. Let's manually compute the positional encoding for each word in the sentence:

- Sentence : "I love music a lot"
- Sentence length: 5 words
- Embedding dimension: 4

Step by step calculation for each word:

- i. For the word "I" at position $pos = 0$

$$PE(0,0) = \sin\left(\frac{0}{10000^{\frac{0}{4}}}\right) = \sin(0) = 0$$

$$PE(0,1) = \cos\left(\frac{0}{10000^{\frac{0}{4}}}\right) = \cos(0) = 1$$

$$PE(0,2) = \sin\left(\frac{0}{10000^{\frac{2}{4}}}\right) = \sin(0) = 0$$

$$PE(0,3) = \cos\left(\frac{0}{10000^{\frac{2}{4}}}\right) = \cos(0) = 1$$

Positional Embedding for "I" : [0,1,0,1]

- ii. For the word "love" at position $pos = 1$

$$PE(1,0) = \sin\left(\frac{1}{10000^{\frac{0}{4}}}\right) = \sin(1) \approx 0.8415$$

$$PE(1,1) = \cos\left(\frac{1}{10000^{\frac{0}{4}}}\right) = \cos(1) \approx 0.5403$$

$$PE(1,2) = \sin\left(\frac{1}{10000^{\frac{2}{4}}}\right) = \sin(0.01) \approx 0.0100$$

$$PE(1,3) = \cos\left(\frac{1}{10000^{\frac{2}{4}}}\right) = \cos(0.01) \approx 0.99995$$

Positional Embedding for "love" : [0.8415, 0.5403, 0.0100, 0.99995]

- iii. For the word "music" at position $pos = 2$

$$PE(2,0) = \sin\left(\frac{2}{10000^{\frac{0}{4}}}\right) = \sin(2) \approx 0.9093$$

$$PE(2,1) = \cos\left(\frac{2}{10000^{\frac{0}{4}}}\right) = \cos(2) \approx -0.4161$$

$$PE(2,2) = \sin\left(\frac{2}{10000^{\frac{2}{4}}}\right) = \sin(0.02) \approx 0.0200$$

$$PE(2,3) = \cos\left(\frac{2}{10000^{\frac{2}{4}}}\right) = \cos(0.02) \approx 0.9998$$

Positional Embedding for “ music ” : [0.9093, -0.4161, 0.0200, 0.9998]

iv. For the word “ a ” at position $pos = 3$

$$PE(3,0) = \sin\left(\frac{3}{10000^{\frac{0}{4}}}\right) = \sin(3) \approx 0.1411$$

$$PE(3,1) = \cos\left(\frac{3}{10000^{\frac{0}{4}}}\right) = \cos(3) \approx -0.9900$$

$$PE(3,2) = \sin\left(\frac{3}{10000^{\frac{2}{4}}}\right) = \sin(0.03) \approx 0.0300$$

$$PE(3,3) = \cos\left(\frac{3}{10000^{\frac{2}{4}}}\right) = \cos(0.03) \approx 0.99955$$

Positional Embedding for “ a ” : [0.1411, -0.9900, 0.0300, 0.99955]

v. For the word “ lot ” at position $pos = 4$

$$PE(4,0) = \sin\left(\frac{4}{10000^{\frac{0}{4}}}\right) = \sin(4) \approx -0.7568$$

$$PE(4,1) = \cos\left(\frac{4}{10000^{\frac{0}{4}}}\right) = \cos(4) \approx -0.6536$$

$$PE(4,2) = \sin\left(\frac{4}{10000^{\frac{2}{4}}}\right) = \sin(0.04) \approx 0.0400$$

$$PE(4,3) = \cos\left(\frac{4}{10000^{\frac{2}{4}}}\right) = \cos(0.04) \approx 0.9992$$

Positional Embedding for “ a ” : [-0.7568, -0.6536, 0.0400, 0.9992]

4.2.1.3 Multi-Head Self-Attention

Self-Attention Mechanism (Single-Head)

Self-attention, also known as intra-attention, is a process that uses the relationship between distinct points in a single sequence to calculate a representation for that sequence. In layman's words, self-attention enables a model to concentrate on important elements of the input while processing a particular token, word, or piece in the sequence.

Self-attention is critical in recognizing interconnections between words that are far apart in a phrase. Instead of processing the sequence in order, self-attention allows the model to "attend" to the full sequence at once and prioritize the most significant pieces for the current job.

The Self-Attention Formula

Self-attention computes a weighted sum of the values (V) where the weights are calculated by comparing the query (Q) with the corresponding keys (K). The formula for self-attention is:

$$Attention(Q, K, V) = softmax\left(\frac{(Q \times K^T)}{\sqrt{d_k}}\right) \times V$$

Where:

- *Q* (Query): The token or word being processed.
- *K* (Key): Other words in the sequence that the current word is compared against.
- *V* (Value): The vector representation of the corresponding words.
- *d_k* : Dimensionality of the keys, used for scaling to prevent large dot products.

```
Word: '[CLS]'
Query (Q): [-0.08137083053588867, -0.019340967759490013, -0.11453711241483688, 0.13843244314193726, -0.03894498571753502]...
Key (K): [-0.09859864413738251, 0.11907981336116791, -0.006296854466199875, -0.09057803452014923, 0.1766502410173416]...
Value (V): [-0.14585432410240173, 0.05925733596086502, -0.06560222804546356, 0.047010913491249084, -0.06170424818992615]...

Word: 'transformers'
Query (Q): [0.0029927343130111694, 0.02160189300775528, -0.043171659111976624, 0.0017787497490644455, -0.020153746008872986]...
Key (K): [-0.044451888650655746, -0.08433902263641357, -0.03519744426012039, 0.010243680328130722, -8.825259283185005e-05]...
Value (V): [-0.00584967527538538, 0.013789396733045578, 0.020967941731214523, -0.021595409139990807, 0.009503227658569813]...

Word: 'have'
Query (Q): [-0.01740225777029991, 0.0103981401771307, -0.045080866664648056, -0.04121097922325134, 0.05770525336265564]...
Key (K): [-0.09257259964942932, -0.00056655769944191, -0.06590491533279419, -0.07862921059131622, 0.0221097469329834]...
Value (V): [-0.006870499812066555, 0.09804168343544006, 0.029116783291101456, 0.03759155794978142, -0.03612659126520157]...

Word: 'revolution'
Query (Q): [-0.008569331839680672, -0.014059850946068764, -0.006205983459949493, -0.048115309327840805, -0.03239266574382782]...
Key (K): [-0.11630650609731674, -0.022513747215270996, 0.05103020742535591, -0.04967394098639488, 0.011849334463477135]...
Value (V): [-0.005915079265832901, 0.045383937656879425, -0.05995199456810951, 0.021632298827171326, -0.01161014474928379]...

Word: '####ized'
Query (Q): [-0.06266915053129196, 0.0014453902840614319, -0.03320736438035965, -0.05162417143583298, 0.01934102177619934]...
Key (K): [-0.08523015677928925, 0.008468426764011383, -0.0067598046734929085, 0.02701178751885891, 0.006084628403186798]...
Value (V): [0.0065290131606161594, 0.020315563306212425, -0.01494109258055687, -0.008277462795376778, 0.0031079258769750595]...
```

Figure 4.3: Example of Attention(Q,K,V)

Self-attention operation break down in the steps:

1. Input Embedding: The input sequence, i.e., a sentence or a phrase, undergoes a transformation process where it is converted into embeddings. These embeddings are dense vector representations that capture the meaning and details of the individual words contained in that sequence.

2. **Query, Key, Value Vectors:** For every single word that exists within the context of the sequence which we are considering, we generate a total of three unique vectors: i.e., the Query vector, the Key vector, and the Value vector. These individual vectors are obtained from the input embeddings which we first obtain by using learned weight matrices that have been optimized during the training process.
3. **Dot Product Attention:** The current word's query vector is compared elementwise with each individual word's key vector in the sequence to calculate attention scores. The scores generated represent the level of relevance, where the score being higher means that the particular word is more relevant to the current word being considered.
4. **SoftMax Normalization:** The attention scores that have been computed are normalized by applying the SoftMax function, which makes the resulting weights fall within a specific range between 0 and 1
5. **Weighted Sum:** It computes the weighted sum of the value vectors based on the attention weights to decide the output that is particular for that word.

This specific process is done over and over again for every single word that occurs in the sequence, which in turn allows the model to generate representations that are contextually aware.

Multi-Head Self-Attention

Multi-Head Self-Attention (MHSA) is one of the most important parts of Transformer designs like BERT and GPT. It's better than single-head attention because it lets a model focus on different parts of an input stream at the same time.

This mechanism is particularly useful in natural language processing (NLP) and computer vision (CV) because:

- it shows how different words in a sentence are related to each other.
- It avoids losing information due to averaging (like in single-head attention).
- It improves parallelism, which speeds up processes.

The multi-head attention system employs numerous self-attention heads in combination, as opposed to relying on a single attention mechanism. A variety of ways to represent the input are learned by each individual head.

- There are a variety of heads that concentrate on various component of the phrase.
- Words can capture many connections with one another.
- It prevents the loss of information, which is a problem since averaging in single-head attention may lead to the loss of context.

Multi-Head Self-Attention Formula

Each head has its own set of W_Q, W_K, W_V matrices. We apply multiple self-attention computations in parallel.

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W_o$$

Where:

- Each head compute attention(Q_i, K_i, V_i)
- The outputs are concatenated and projected using W_o

Detailed explanation with example:

- Sentence : "I love music a lot"

Step 1: Tokenization

The first step in converting the sentence "I love music a lot" into embeddings is tokenization that is breaking the sentence into individual words:

- Tokens=["I","love","music","a","lot"]

Depending on the tokenizer, words might be broken further into subwords, especially for unknown words (e.g., "loving" might be split into "lov" and "ing").

Step 2: Converting Tokens to Vectors

Each token is mapped to a numerical vector using a pre-trained embedding model (e.g., Word2Vec, GloVe, BERT). These vectors capture semantic relationships between words.

Let's assume we are using Word2Vec trained on a large corpus, and each word is represented as a 4-dimensional vector.

For example, an embedding table (simplified):

Word	Embedding (4D Example)
I	[0.1, 0.3, 0.2, 0.5]
love	[0.7, 0.5, 0.6, 0.9]
music	[0.8, 0.6, 0.7, 0.3]
a	[0.3, 0.4, 0.2, 0.1]
lot	[0.5, 0.2, 0.4, 0.6]

Table 4.2: Example Embedding Table

Each word is now represented as a vector in a continuous space.

This forms the input matrix :

$$X = \begin{bmatrix} 0.1 & 0.3 & 0.2 & 0.5 \\ 0.7 & 0.5 & 0.6 & 0.9 \\ 0.8 & 0.6 & 0.7 & 0.3 \\ 0.3 & 0.4 & 0.2 & 0.1 \\ 0.5 & 0.2 & 0.4 & 0.6 \end{bmatrix}$$

Step 3: Contextual Embeddings (Transformer-based Models)

If we use Transformer-based embeddings (e.g., BERT, GPT), the embeddings depend on the context.

For instance, in BERT, the embedding for "music" in:

- "I love music a lot" will be different from "Music is my passion."
- This is because BERT considers surrounding words when computing embeddings.

Word	Embedding (4D)
I	(0.1, 0.3, 0.2, 0.5)
love	(0.7, 0.5, 0.6, 0.9)
music	(0.8, 0.6, 0.7, 0.3)
a	(0.3, 0.4, 0.2, 0.1)

Table 4.3: Contextual Embeddings Table 1

Word	Embedding (4D)	Positional Encoding (4D)	Final Input
I	(0.1, 0.3, 0.2, 0.5)	(0.00, 1.00, 0.00, 1.00)	(0.1, 1.3, 0.2, 1.5)
love	(0.7, 0.5, 0.6, 0.9)	(0.84, 0.54, 0.91, 0.41)	(1.54, 1.04, 1.51, 1.31)
music	(0.8, 0.6, 0.7, 0.3)	(0.90, 0.44, 0.99, 0.14)	(1.7, 1.04, 1.69, 0.44)
a	(0.3, 0.4, 0.2, 0.1)	(0.91, 0.41, 1.00, 0.08)	(1.21, 0.81, 1.2, 0.18)
lot	(0.5, 0.2, 0.4, 0.6)	(0.95, 0.31, 1.00, 0.02)	(1.45, 0.51, 1.4, 0.62)

Table 4.4: Contextual Embeddings Table 2

Multi-Head Self-Attention (Refining Contextual Meaning). Now, the model applies multi-head self-attention, which re-weights how each word interacts with the others. Compute Query, Key, and Value Matrices

$$Q = XW_Q, K = XW_K, V = XW_V,$$

where:

- W_Q, W_K, W_V are trainable weight matrices.
- Q, K, V capture different aspects of word relationships.

For simplicity, let's assume we get the following matrices:

$$Q = \begin{bmatrix} 0.4 & 0.5 & 0.3 & 0.7 \\ 1.1 & 1.0 & 0.8 & 1.2 \\ 1.3 & 1.2 & 0.9 & 0.6 \\ 0.7 & 0.8 & 0.5 & 0.4 \\ 0.9 & 0.6 & 0.7 & 1.0 \end{bmatrix} \quad K = \begin{bmatrix} 0.3 & 0.6 & 0.2 & 0.8 \\ 1.0 & 0.9 & 0.7 & 1.1 \\ 1.2 & 1.1 & 0.8 & 0.5 \\ 0.6 & 0.7 & 0.4 & 0.3 \\ 0.8 & 0.5 & 0.6 & 0.9 \end{bmatrix} \quad V = \begin{bmatrix} 0.5 & 0.7 & 0.4 & 0.9 \\ 1.2 & 1.1 & 0.9 & 1.3 \\ 1.4 & 1.3 & 1.0 & 0.8 \\ 0.8 & 0.9 & 0.5 & 0.6 \\ 1.0 & 0.7 & 0.8 & 1.1 \end{bmatrix}$$

Compute Attention Scores. The attention scores are calculated as:

$$Attention(Q, K, V) = softmax\left(\frac{(Q \times K^T)}{\sqrt{d_k}}\right) \times V$$

Where:

- Q (Query): The token or word being processed.
- K (Key): Other words in the sequence that the current word is compared against.
- V (Value): The vector representation of the corresponding words.
- d_k : Dimensionality of the keys, used for scaling to prevent large dot products.

Step 4: Compute Contextualized Embeddings

The final contextualized representation of each word is obtained by multiplying the attention weights with the V matrix:

$$Output = Attention\ Weights \times V$$

Now, each word embedding is refined based on context, meaning:

- "love" will have more attention towards "music".
- "lot" will focus more on "a" and "music".

Step 5: Final Representation (Context-Aware Embeddings)

At this point, we have contextualized embeddings that capture meaning based on the entire sentence.

Word	Final Contextualized Embedding
I	(0.98, 0.94, 1.09, 1.07)
love	(1.13, 1.08, 1.25, 1.22)
music	(1.12, 1.07, 1.24, 1.21)
a	(0.99, 0.95, 1.10, 1.08)
lot	(1.04, 1.00, 1.17, 1.14)

Table 4.5: Final Contextualized Embedding

4.2.1.4 Add & Norm (Residual Connection & Layer Normalization) in Transformers

The Add & Norm mechanism in Transformer models is a combination of Residual Connections (Add) and Layer Normalization (Norm). It plays a crucial role in stabilizing deep learning models, improving gradient flow, and ensuring efficient training. This mechanism is applied twice in each Transformer encoder and decoder block—once after the self-attention mechanism and once after the feed-forward network (FFN).

In Transformer models, Add & Norm is applied twice per encoder and decoder block:

- After the Multi-Head Self-Attention layer.
- After the Feed-Forward Network (FFN).

1. Residual Connection (Add)

Residual connections help prevent the vanishing gradient problem, which occurs when deep neural networks fail to update weights properly during training. Instead of passing only the transformed output to the next layer, a residual connection adds the original input back to the transformed output.

Mathematical Representation

If x is the input and $F(x)$ is the output of a transformation (like self-attention or a feed-forward network), then the residual connection is:

$$y = F(x) + x$$

This ensures that important features from the input are not lost during transformation. By maintaining this shortcut connection, the model can retain essential information from earlier layers while also learning new patterns.

Benefits of Residual Connections

- Prevents the loss of important information from the original input.
- Reduces the risk of vanishing gradients in deep networks.
- Allows easier training of very deep models by ensuring stable learning.

2. Layer Normalization (Norm)

After the residual connection, the model applies Layer Normalization to stabilize activations and speed up convergence. Unlike Batch Normalization, which depends on batch statistics, Layer Normalization works independently for each input sequence.

Formula for Layer Normalization

For an input $y = [y_1, y_2, y_3, \dots, y_n]$ the normalization is applied as:

$$\hat{y}_i = \frac{y_i - \mu}{\sigma}$$

where:

- $\mu = \frac{1}{n} \sum_{i=1}^n y_i$: Mean of the features.
- $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (y_i - \mu)^2$: Variance of the features.
- \hat{y}_i Normalized output.

After normalization, learnable parameters γ (*scale factor*) and β (*shift factor*) are applied:

$$z = \gamma \hat{y}_i + \beta$$

where γ and β allow the model to adjust the normalized values.

- Normalization Layer Ensures stable training by keeping activations within a controlled range.
- Works independently of batch size, unlike Batch Normalization.
- Helps with faster convergence and better generalization.

3. Complete Add & Norm Process

The full Add & Norm operation in a Transformer block follows these steps:

- Compute the transformed output $F(x)$ using self-attention or a feed-forward network.
- Apply residual connection: Add the original input x back to the transformed output.
- Perform layer normalization to stabilize the output.

Mathematical Formula

$$z = \text{Layer Norm}(x + F(x))$$

where:

- x is the input embedding.
- $F(x)$ is the output of the transformation (self-attention or FFN).
- Residual connection ensures that information is not lost.
- Layer Normalization standardizes the output.

4.2.1.5 Feed-Forward Network (FFN) in Transformer Encoder & Decoder Architecture:

The Feed-Forward Network (FFN) is a fundamental component of each encoder and decoder block in the Transformer architecture. Its primary function is to process and refine individual token representations independently, ensuring that each token undergoes additional transformation beyond what the Multi-Head Attention (MHA) mechanism provides.

Unlike MHA, which models relationships between tokens and enables the model to capture global dependencies, the FFN operates on each token separately, without considering other tokens in the sequence. This means that while MHA provides contextualized embeddings by attending to different parts of the sequence, the FFN is responsible for further enhancing each token representation independently. This step is crucial because it allows the Transformer model to introduce non-linearity, improve feature extraction, and increase the overall representational capacity of the network.

In a self-attention mechanism like MHA, each token in the sequence is represented as a weighted sum of all other tokens. While this contextualization is powerful, it is still linear in nature, meaning it does not introduce non-linearity or learn complex hierarchical features on its own. The FFN overcomes this limitation by applying a non-linear transformation to each token individually, making it capable of capturing abstract patterns and hierarchical relationships.

The primary role of the FFN can be summarized as follows:

- Enhancing token representations after attention mechanisms: While self-attention extracts dependencies between tokens, the FFN transforms each token's embedding into a higher-dimensional space before projecting it back.
- Introducing non-linearity into the model: The use of activation functions like ReLU (Rectified Linear Unit) or GELU (Gaussian Error Linear Unit) allows the model to capture complex interactions.
- Ensuring per-token processing: Unlike attention mechanisms that mix token information, FFN ensures that each token is processed independently, allowing for localized feature learning.

Each Transformer encoder and decoder block contains an FFN as its second major component, following the Multi-Head Attention (MHA) mechanism. The FFN processes the output from the attention layers separately for each token, refining the token embeddings before they are passed to the next layer.

Step-by-Step Breakdown of FFN Processing:

First Linear Transformation (Expanding Dimensions)

- The input token embedding x is passed through a fully connected linear transformation, where the embedding dimension is expanded (e.g., from 512 to 2048 in BERT).
- This increased capacity allows the network to learn a more expressive representation for each token

$$h = W_1 \cdot x + b_1$$

Here:

- W_1 is a learnable weight matrix that maps the input embedding to a higher-dimensional space.
- b_1 is a learnable bias.

Non-Linear Activation Function

- To introduce non-linearity, the output is passed through an activation function.
- Common choices include ReLU and GELU:
 - ReLU (Rectified Linear Unit):
 - $ReLU(h) = \max(0, h)$

GELU (Gaussian Error Linear Unit):

$$GELU(h) = h \cdot \Phi(h)$$

where $\Phi(h)$ is the standard Gaussian cumulative distribution function.

- Without activation, the FFN would just be a linear transformation, limiting its ability to learn complex hierarchical relationships.

- The activation function helps capture more intricate patterns within each token embedding.

Second Linear Transformation (Reducing Dimensions)

- The expanded representation is then projected back to the original token dimension using another linear layer.
- $y = W_2 \cdot h' + b_2$

Here, W_2 and b_2 are the second set of learnable parameters that map the higher-dimensional representation back to the original embedding size (e.g., $2048 \rightarrow 512$).

Final Add & Norm (Residual Connection + Layer Normalization)

- A residual connection is applied, where the original input x is added back to the FFN output y .
- This prevents vanishing gradients and allows gradients to flow efficiently during backpropagation.
- Layer Normalization is applied to stabilize training and improve convergence

$$z = \text{LayerNorm}(x + y)$$

4.2.2 Decoder

Introduction to the Transformer Decoder

The Transformer model, introduced by Vaswani et al. (2017), has revolutionized Natural Language Processing (NLP) by enabling fast and parallelized training for sequence-to-sequence tasks. The decoder is a fundamental component of the Transformer model, responsible for generating output sequences in applications such as:

- Machine Translation (e.g., English to French translation)
- Text Summarization
- Speech Recognition
- Dialogue Generation

Unlike Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTMs), which process sequences sequentially, the Transformer decoder leverages self-attention and cross-attention mechanisms to efficiently model relationships between words. This approach enables parallel computation, improving the model's ability to capture long-range dependencies while maintaining high training efficiency.

Functionality of the Transformer Decoder

The decoder receives contextualized input embeddings from the encoder and processes them through multiple layers to generate a structured and logical output sequence. In sequence-to-sequence tasks like machine translation, the decoder must not only attend to previously

generated tokens (self-attention) but also extract relevant information from the encoder's output (cross-attention).

Key Differences Between Encoder and Decoder

A key distinction between the encoder and decoder is the masked multi-head self-attention used in the decoder. While the encoder allows tokens to attend to all positions in the input, the decoder ensures causal dependency, meaning each token can only attend to previous tokens and not future ones. This is achieved using a masking mechanism, which sets attention scores for future tokens to negative infinity before applying the SoftMax function. This ensures that the model generates output in a left-to-right manner, preserving natural sentence structure during training.

Beyond self-attention, the decoder employs cross-attention, enabling it to interact with the encoder's output. This mechanism allows the decoder to query relevant information from the encoder-generated representations, ensuring that the generated sequence aligns with the input. For example, in machine translation, the decoder utilizes cross-attention to extract meaningful representations of the input sentence before constructing the translated output word by word.

To further refine token representations, each decoder layer contains a Feed-Forward Network (FFN), which applies additional transformations to each token's embedding. Unlike attention mechanisms, which model relationships across tokens, the FFN processes each token independently through a series of non-linear transformations. This enhances the decoder's ability to capture abstract features, improving the fluency and coherence of generated text.

Additionally, Add & Norm layers follow the FFN to stabilize training by incorporating residual connections and layer normalization, preventing gradient-related issues.

Advantages of the Transformer Decoder

One of the primary advantages of the Transformer decoder is its parallelization capability. Unlike RNNs and LSTMs, which process sequences step by step, the Transformer decoder computes attention scores for all tokens simultaneously, leading to significant improvements in computational efficiency. This makes the Transformer decoder well-suited for handling large-scale datasets and generating long sequences, as seen in models such as GPT (Generative Pre-trained Transformer) and T5 (Text-To-Text Transfer Transformer).

At the final stage of decoding, the processed representations are passed through a linear transformation and a SoftMax layer to produce output tokens. The linear layer maps the decoder's hidden states to a probability distribution over the vocabulary, determining the likelihood of each token appearing next in the sequence. The SoftMax function then normalizes these probabilities, ensuring that the model generates a well-formed output.

To improve text generation quality, beam search or greedy decoding is commonly used to select the most probable sequence of tokens.

Challenges and Future Improvements

Despite its advantages, the Transformer decoder faces challenges related to computational efficiency and memory usage. The attention mechanism requires storing large matrices

proportional to sequence length, which can become prohibitive when dealing with long documents or real-time applications.

To address these challenges, researchers have developed efficient Transformer variants such as:

- Reformer – Reduces memory usage using locality-sensitive hashing.
- Longformer – Uses dilated attention for better handling of long texts.
- Sparse Transformers – Introduces sparse attention mechanisms to improve efficiency.

Another area of improvement is the integration of pre-trained language models into the Transformer decoder. Modern architectures like GPT-3, T5, and BART extend the basic Transformer decoder structure by incorporating pre-trained embeddings, improving the model's ability to generate coherent text with minimal fine-tuning. This has led to advancements in few-shot learning, where models can perform well on new tasks with limited labelled data.

The Transformer decoder is a cornerstone of modern sequence generation models, enabling efficient and high-quality text production for various NLP applications. Its combination of masked self-attention, cross-attention, and feed-forward networks allows it to model complex dependencies, generate fluent sequences, and process large-scale data efficiently.

The flexibility of the decoder to integrate pre-trained models and fine-tuning techniques has significantly advanced automated text generation, machine translation, and conversational AI, solidifying its role as one of the most impactful innovations in deep learning.

Architecture of the Transformer Decoder

The decoder consists of multiple identical layers ($N \times$ layers), with each layer containing the following core components:

1. Input Embedding & Positional Encoding
2. Masked Multi-Head Self-Attention (prevents attending to future tokens)
3. Multi-Head Attention (Cross-Attention with Encoder Outputs)
4. Feed-Forward Network (FFN) (applies transformations to token embeddings)
5. Add & Norm Layers (Residual Connections and Layer Normalization)
6. Linear & SoftMax Layer for Final Predictions

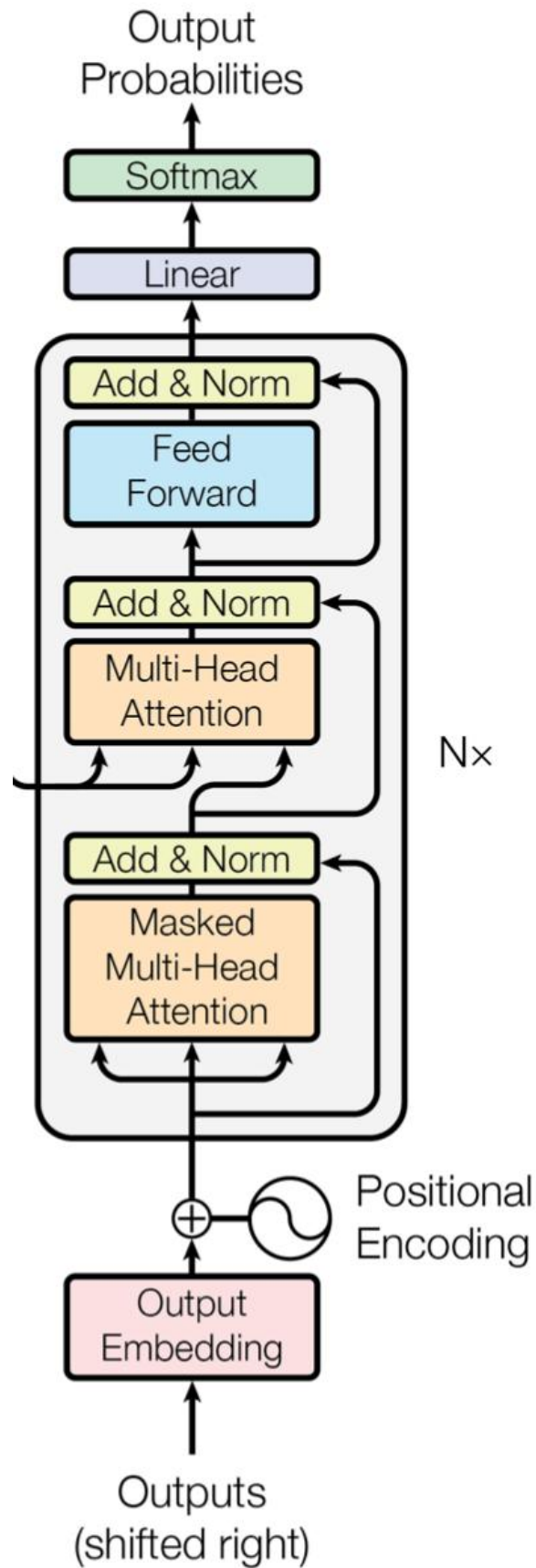


Figure 4.4: Decoder

Key Components of the Transformer Decoder

Input Embedding & Positional Encoding

- The decoder begins by embedding the input tokens (i.e., previously generated words) into dense vectors.
- Since transformers lack recurrence, positional encodings are added to retain the order of words in a sequence.
- Positional encoding is calculated using sinusoidal functions, ensuring unique representations for each position in the sequence.

Masked Multi-Head Self-Attention

- This layer allows the decoder to focus on relevant parts of the sequence while ensuring that future tokens remain unseen during training.
- A causal mask is applied so that each token can only attend to past tokens (i.e., already generated words) and not future ones.
- This prevents information leakage and enforces autoregressive decoding (i.e., generating text step-by-step).
- The multi-head attention mechanism enables the decoder to capture diverse contextual relationships efficiently.

Multi-Head Attention (Cross-Attention with Encoder Outputs)

- This is where the decoder interacts with the encoder outputs.
- It allows the decoder to selectively attend to relevant parts of the encoder's processed input representations.
- The query vectors come from the decoder, while the keys and values come from the encoder.
- This mechanism ensures that the decoder properly aligns generated words with the input sentence, improving translation and generation quality.

Feed-Forward Network (FFN)

- Each decoder layer contains a fully connected feed-forward network (FFN) that applies transformations to each token.
- It consists of Two linear layers with a ReLU activation function in between. The FFN is applied independently to each position in the sequence.
- This step improves the decoder's ability to model complex linguistic relationships.

Add & Norm Layers (Residual Connections and Layer Normalization)

- Residual connections are used after every attention and feed-forward layer to ensure gradient stability.

- Layer normalization is applied to stabilize training and enhance performance.

Linear & SoftMax Layer for Final Predictions

- The final decoder output is passed through a fully connected linear layer to generate a probability distribution over the vocabulary.
- The SoftMax function is applied to predict the next word in the sequence.
- The highest probability token is selected and fed back into the decoder for the next timestep.

Step-by-Step Workflow of the Decoder

1. The input embeddings are computed and combined with positional encodings.
2. The masked multi-head self-attention mechanism processes previous tokens while preventing future tokens from being visible.
3. Cross-attention allows the decoder to focus on relevant encoder outputs.
4. The feed-forward network (FFN) refines the output representations.
5. Residual connections and layer normalization ensure stability in training.
6. The final output is passed through a linear & SoftMax layer to produce probabilities over the vocabulary.
7. The most likely token is selected and used as input for the next decoding step.
8. The process repeats until an end-of-sequence ([EOS]) token is generated.

Difference between Encoder and Decoder:

Features	Encoder	Decoder
Input	Full input sequence	Previously generated words
Self-Attention	Standard multi-head attention	Masked multi-head attention
Cross-Attention	No cross-attention	Attends to encoder outputs
Output	Contextual representation of input	Final output sequence

Table 4.6: Difference between Encoder and Decoder

The masked self-attention mechanism is crucial in ensuring that:

1. The model does not cheat by looking at future tokens before they are generated.
2. The sequence is generated step-by-step, following an autoregressive approach.
3. The training process remains stable by preventing the decoder from making unnecessary assumptions about the next token.

4.3 Model Selection for Paraphrase Generation

6.1.1 BART: Bidirectional and Auto-Regressive Transformers

BART (Bidirectional and Auto-Regressive Transformers) is a sequence-to-sequence transformer model that uniquely combines bidirectional encoding with autoregressive decoding, making it highly effective for both text understanding and generation. Unlike traditional transformer models that either focus only on encoding (e.g., BERT) or only on decoding (e.g., GPT), BART integrates both functionalities, enabling it to handle complex language tasks such as text summarization, machine translation, and dialogue generation (Lewis et al., 2020).

1. Encoder: Understanding Context with Bidirectional Processing

The encoder in BART operates similarly to BERT, processing entire input sequences bidirectionally, meaning it can attend to all words in the input simultaneously. This allows it to capture intricate relationships between words, leading to better context understanding. The bidirectional nature of the encoder enables BART to extract deep contextual relationships between words, improving its ability to understand complex sentence structures, disambiguate meaning, and recognize dependencies across words (Lewis et al., 2020).

The encoder consists of multiple self-attention layers, allowing each token to attend to all other tokens in the input sequence. This mechanism helps BART recognize long-range dependencies, ensuring that information from earlier and later tokens is equally utilized during encoding. The self-attention weights also enable the model to assign different importance levels to words, ensuring that more relevant words influence the representation of others (Lewis et al., 2020).

A significant advantage of BART's encoder is its ability to handle corrupted text inputs during pretraining, where words are randomly deleted, shuffled, or replaced. This forces the model to learn how to reconstruct the original sentence, making it robust for real-world NLP applications where text may be incomplete, noisy, or grammatically inconsistent (Lewis et al., 2020).

2. Decoder: Generating Text Autoregressively

The decoder in BART follows an autoregressive approach, similar to GPT. Instead of processing all tokens simultaneously like the encoder, the decoder generates words one at a time while attending to previously generated tokens. This ensures that the model produces coherent and fluent text while maintaining logical sequencing (Lewis et al., 2020).

To prevent the decoder from seeing future tokens during training, BART applies masked self-attention to the decoder's input. This ensures that word generation happens sequentially, preventing the model from "cheating" by looking ahead. Each word prediction is influenced by previously generated words and the encoded input sequence, ensuring contextual consistency in text generation (Lewis et al., 2020).

The decoder also incorporates cross-attention layers, which allow it to attend to the encoder's output representations while generating new tokens. This mechanism helps BART effectively

integrate context from the input text, making it highly effective for translation, summarization, and paraphrasing tasks. Cross-attention enables the decoder to focus on important parts of the input, ensuring that relevant information is retained while generating output (Lewis et al., 2020).

Additionally, each decoder layer contains a Feed-Forward Network (FFN) that applies non-linear transformations to further refine token representations. This improves the quality of generated text by ensuring that words are generated based on both immediate context and broader sentence-level meaning (Lewis et al., 2020).

3. How BART's Architecture Enhances Performance

BART's hybrid architecture (a BERT-like encoder and a GPT-like decoder) gives it significant advantages over traditional transformer models. The bidirectional encoding ensures a deep understanding of input context, while the autoregressive decoding enables the model to generate human-like text fluently. Compared to standard sequence-to-sequence models, BART benefits from corruption-based pretraining, making it more robust to text manipulation, missing words, and noisy inputs (Lewis et al., 2020).

Another key advantage of BART is its scalability. Unlike older models that struggle with long-range dependencies, BART's self-attention mechanism allows it to process and generate long documents effectively without performance degradation. This makes it well-suited for tasks such as document summarization, legal text processing, and dialogue modeling, where handling large input-output sequences is crucial (Lewis et al., 2020).

Furthermore, BART can be fine-tuned for different NLP tasks with minimal modifications. By adding a simple task-specific classification or generation head, BART can be adapted for sentiment analysis, question-answering, and language modeling. This flexibility makes it a powerful choice for researchers and developers looking to apply deep learning to real-world language applications (Lewis et al., 2020).

4. BART's Key Features

- Bidirectional Encoding (BERT-like): Captures deep contextual relationships between words.
- Autoregressive Decoding (GPT-like): Generates coherent and fluent text.
- Masked Self-Attention in Decoder: Prevents future words from influencing predictions, ensuring logical sequence generation.
- Cross-Attention Mechanism: Ensures decoder extracts important information from encoder outputs.
- Pre-trained Using Text Corruption: Enhances robustness against incomplete, noisy, and shuffled text.
- Highly Adaptable for Multiple NLP Tasks: Supports summarization, translation, question-answering, and more.

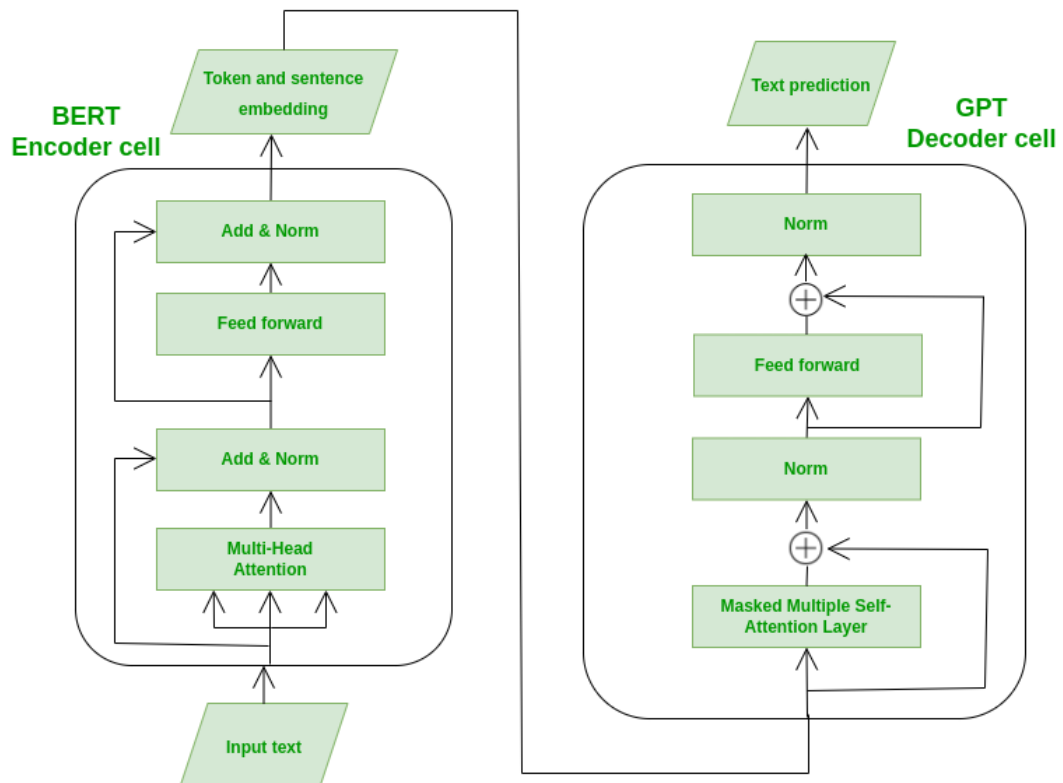


Figure 4.5: BART Architecture

This Figure 4.4 illustrates the BART (Bidirectional and Auto-Regressive Transformers) model's architecture, showing how it combines a BERT-like encoder with a GPT-like decoder

BART (Bidirectional and Auto-Regressive Transformers) is a sequence-to-sequence (Seq2Seq) model designed as a denoising autoencoder for natural language generation and understanding tasks. Unlike conventional transformer models that focus on either encoding (e.g., BERT) or decoding (e.g., GPT), BART integrates both bidirectional encoding and autoregressive decoding to improve performance on text generation tasks such as summarization, machine translation, question answering, and text reconstruction (Lewis et al., 2020). Its architecture consists of an encoder-decoder structure, where the encoder is bidirectional like BERT, allowing it to understand the full context of an input sequence, and the decoder is autoregressive like GPT, generating text in a left-to-right manner. This hybrid approach enables BART to effectively process both structured and noisy text inputs while ensuring fluent text generation.

The encoder in BART follows the same structure as BERT, employing self-attention mechanisms that allow each token to attend to every other token in the sequence. This bidirectional processing ensures that the model captures long-range dependencies and contextual relationships, making it particularly effective for understanding sentence structures and Clarifying meanings in complex texts. The encoder consists of multi-head self-attention layers, feed-forward networks (FFN), and layer normalization with residual connections, which stabilize training and enable deep contextual representations. Additionally, BART's encoder is pretrained using corrupted text reconstruction, meaning it learns to reconstruct input sequences that have been artificially modified by masking, deleting, commuting, or replacing

words. This denoising objective improves the model's robustness, allowing it to handle incomplete or noisy text data better than traditional transformer models.

In contrast, the decoder follows an autoregressive generation approach, meaning it generates text one token at a time, conditioned on previously generated tokens. Unlike the encoder, the decoder employs masked self-attention, preventing it from attending to future tokens in the sequence. This enforces causal dependency, ensuring that each generated word is influenced only by past words, similar to how GPT models operate. Additionally, the decoder includes cross-attention layers, which allow it to attend to the contextualized representations from the encoder. This mechanism ensures that the generated text remains faithful to the input, making BART particularly effective for sequence-to-sequence tasks such as translation and summarization. The decoder also contains feed-forward layers and normalization layers, which further refine token embeddings before they are passed through the final output layer, where a softmax function predicts the next token.

One of BART's most significant advantages over other transformer models is its pretraining objective, which relies on a denoising autoencoder approach rather than masked language modeling (BERT) or causal language modeling (GPT). During pretraining, BART applies multiple corruption techniques to input sequences, including token masking, token deletion, text infilling, sentence permutation, and document rotation. These transformations force the model to learn meaningful representations by reconstructing the original text from corrupted versions, making BART highly resilient to input noise. This property makes it particularly effective for tasks involving noisy text data, such as dialogue modeling, text correction, and abstractive summarization.

The information flow in BART starts with an input sequence being tokenized and passed through the encoder, where self-attention layers compute contextualized embeddings. These embeddings are then fed into the decoder, which applies masked self-attention to ensure sequential generation while using cross-attention layers to extract relevant context from the encoder. Finally, the output passes through a linear transformation and a softmax layer, which selects the most probable token for each step in the sequence generation process. This combination of bidirectional encoding, autoregressive decoding, and cross-attention makes BART one of the most effective sequence-to-sequence transformer models available.

Compared to other transformer architectures, BART provides several key advantages. The bidirectional encoder enables deeper contextual understanding, capturing complex dependencies across long texts, while the autoregressive decoder ensures fluent and coherent text generation. The cross-attention mechanism effectively links the encoder and decoder, ensuring that generated outputs remain accurate and contextually relevant. Furthermore, BART is highly adaptable; it can be fine-tuned for various NLP tasks with minimal architectural modifications. This flexibility allows BART to outperform many other transformer-based models in summarization, text generation, and translation tasks. Ultimately, BART's combination of bidirectional context encoding, autoregressive text generation, and denoising pretraining makes it a highly versatile and powerful model for natural language processing applications.

Process of BART

1. Preprocessing the Input

Before the model processes the text, the input undergoes tokenization and conversion into numerical representations.

Example Input: "I love music a lot."

1. Tokenization: The input text is split into subwords or tokens.

["I", "love", "music", "a", "lot", "."]

2. Conversion to IDs: Each token is mapped to a numerical representation using a vocabulary.

[101, 1045, 2293, 2189, 1037, 2843, 1012, 102]

3. Adding Special Tokens: The model adds <s> (start) and </s> (end) tokens.

["<s>", "I", "love", "music", "a", "lot", ".", "</s>"]

2. Encoding (Bidirectional Processing)

The encoder processes the input simultaneously in both directions, allowing it to understand context deeply.

1. Embeddings: Converts tokenized words into dense vectors.
2. Self-Attention Mechanism: Each token interacts with all other tokens in the sentence to understand relationships.
 - Example: "music" attends to both "love" and "a lot" for better meaning.
3. Feed-Forward Layers: Non-linear transformations refine word representations.
4. Output Representation: The encoder produces a contextualized vector representation for each word.

3. Decoding (Autoregressive Generation)

The decoder generates output text one word at a time, using previously generated words as context.

1. Input Initialization:
 - The decoder starts with a special <s> token.
 - It does NOT see future words (uses masked self-attention).
2. Word-by-Word Prediction:
 - The first token is generated: <s> → "I'm"
 - The next token is generated based on previous words: "<s> I'm" → "really"

- The process continues until <eos> is reached.
3. Cross-Attention Between Encoder & Decoder:
 - The decoder attends to encoder outputs to maintain meaning while rewording.
 4. Final Output Generation: "<s> I'm really passionate about music </s>"

4. Postprocessing & Output

Once the text generation is complete:

1. Detokenization: The numerical output is mapped back to words.
2. Removing Special Tokens: <s> and </s> are removed.
3. Final Output Text: "I'm really passionate about music."

6.1.2 DeepSeek Model

In this study on plagiarism detection and paraphrase generation using Generative AI, the DeepSeek R1 Distill Qwen 7B model was employed for paraphrase generation. This model, developed by DeepSeek-AI, is a distilled version of a large-scale transformer model designed for efficient text generation. Unlike traditional rule-based paraphrase detection approaches, DeepSeek leverages deep learning techniques to generate meaning-preserving textual variations, making it well-suited for applications in plagiarism detection. By employing context-aware paraphrase generation, this model enhances the ability to identify semantically equivalent but lexically diverse expressions, improving automated plagiarism detection systems.

DeepSeek-R1 is a reinforcement learning (RL)-based large language model (LLM) designed to improve reasoning capabilities without relying on supervised fine-tuning (DeepSeek-AI, 2024). Unlike traditional transformer-based models that undergo supervised learning with labeled datasets, DeepSeek-R1 applies reinforcement learning directly to the base model to enhance its ability to generate accurate and coherent text. This makes it a powerful tool for paraphrase generation in plagiarism detection, as it can produce semantically equivalent yet lexically distinct variations of input text.

DeepSeek Architecture and Training Approach

DeepSeek-R1 builds upon the transformer-based DeepSeek-V3-Base model and employs a multi-stage reinforcement learning pipeline to develop improved reasoning and text generation capabilities. The training methodology consists of three core phases:

1. DeepSeek-R1-Zero: Reinforcement Learning Without Supervision
 - The initial version, DeepSeek-R1-Zero, was trained using pure reinforcement learning without relying on any labeled datasets.
 - The model developed self-evolving reasoning behaviors, improving text coherence and paraphrasing ability through iterative RL steps.
 - However, R1-Zero faced issues such as poor readability and occasional language mixing, which were addressed in later iterations (DeepSeek-AI, 2024).
2. DeepSeek-R1: Cold-Start Training with Supervised Fine-Tuning (SFT)

- To enhance readability and language alignment, DeepSeek-R1 incorporated supervised fine-tuning on a cold-start dataset before undergoing RL.
 - This additional training step helped align the model's outputs with human preferences, ensuring fluent, high-quality paraphrases.
3. Distillation for Small Model Efficiency
- DeepSeek-R1's knowledge was further distilled into smaller dense models (1.5B to 70B parameters) to make it computationally efficient.
 - The distilled 7B model (DeepSeek-R1-Distill-Qwen-7B) outperformed larger baseline models like GPT-4o-0513 in certain reasoning benchmarks.

Paraphrase Generation Using DeepSeek

DeepSeek-R1 is particularly effective for paraphrase generation, which is critical in plagiarism detection systems. The model's reinforcement learning-based training allows it to:

- Generate semantically equivalent paraphrases with diverse wording.
- Ensure contextual consistency while varying syntax and structure.
- Avoid direct copying while maintaining fluency and coherence.

Example of DeepSeek Paraphrase Generation

Original Sentence	DeepSeek-R1 Paraphrase
"AI is transforming healthcare services."	"Healthcare is being revolutionized by artificial intelligence."
"The economy is facing inflation challenges."	"Rising inflation is impacting the economy."

Table 4.7: Deepseek Model Paraphrase Examples

DeepSeek's ability to preserve meaning while altering sentence structure makes it highly valuable for plagiarism detection systems, which often struggle to identify paraphrased text using conventional string-matching algorithms.

Reinforcement Learning for Optimized Paraphrasing

DeepSeek-R1's paraphrase generation capability is enhanced through reinforcement learning, which optimizes text output by:

- Incentivizing accurate and diverse reasoning: The model learns through self-reinforcement and refines its text generation ability over multiple iterations.
- Applying reward modeling: The model receives accuracy rewards for producing paraphrases that preserve original intent while being lexically distinct.
- Rejection sampling and supervised fine-tuning: Helps eliminate low-quality paraphrases and improves text coherence.

This RL-based training enables DeepSeek-R1 to surpass supervised learning models in paraphrasing tasks, making it ideal for academic integrity applications.

DeepSeek-R1 represents a significant advancement in reinforcement learning-based text generation, particularly in paraphrase generation for plagiarism detection. By leveraging self-evolving reasoning capabilities, the model can generate fluent, contextually accurate

paraphrases, making it a valuable tool for detecting paraphrased plagiarism in academic and professional settings (DeepSeek-AI, 2024).

4.4 Model Fine Tuning Process

Fine-Tuning the BART Model for Paraphrase Generation on PAWS

The Bidirectional and Auto-Regressive Transformer (BART) is a powerful sequence-to-sequence model that has been widely used for text generation tasks, including paraphrase generation, summarization, and translation. Fine-tuning BART on the PAWS dataset allows the model to learn effective paraphrasing patterns, improving its ability to generate semantically similar yet diverse sentences. The fine-tuning process involved dataset preparation, tokenization, model configuration, training optimization, and evaluation.

Dataset Preparation

For this task, we utilized the PAWS labeled_final dataset, which consists of sentence pairs (sentence1, sentence2) where sentence2 is either a paraphrase or a non-paraphrase of sentence1. The dataset was loaded directly from Hugging Face's datasets library, ensuring efficient access to predefined training and validation splits. Unlike a manual subset selection approach, we leveraged the entire dataset to provide the model with a broader range of paraphrasing patterns.

To ensure balanced training and mitigate potential biases in the dataset, the training and validation data were used as provided by Hugging Face. This approach preserved the natural distribution of paraphrase and non-paraphrase pairs, ensuring effective generalization.

Tokenization and Preprocessing

To process the dataset for training, we used the facebook/bart-large tokenizer, which converts text into numerical representations. Tokenization was applied to both input and target sentences with:

- Truncation to limit sequences to a maximum length of 128 tokens.
- Padding to ensure all inputs maintain a uniform size.
- Mapping sentence1 to the input and sentence2 to the target output.
- Formatting labels properly by replacing padding tokens with -100, allowing the model to ignore them during loss computation.

Tokenization was performed batch-wise, leveraging Hugging Face's dataset mapping functionality for efficiency.

Model Configuration and Training Setup

The facebook/bart-large model, which consists of 406 million parameters, was fine-tuned for paraphrase generation. The model was configured with regularization techniques such as dropout (0.3) to reduce overfitting. Training was carried out using Hugging Face's Trainer API,

optimizing computational efficiency through gradient accumulation and mixed-precision (fp16) training.

To ensure training stability and generalization, the following hyperparameters were used:

- Batch Size: 8 per device for both training and evaluation, optimizing for memory efficiency.
- Learning Rate: 5e-5, ensuring stable learning and avoiding large updates that could destabilize training.
- Epochs: 3, with validation after each epoch to monitor performance improvement.
- Weight Decay: 0.01, applied to prevent overfitting and ensure smooth parameter updates.
- Gradient Clipping: Implemented to stabilize training by preventing excessively large gradient updates.
- Warmup Steps: Automatically adjusted via a learning rate scheduler to facilitate better convergence.

For training monitoring and model checkpointing, logging was configured to occur every 500 steps, allowing continuous tracking of performance metrics. The model checkpointing strategy ensured that only two model checkpoints were stored, with the best-performing model based on validation loss being retained.

Evaluation and Performance Measurement

After fine-tuning, the model was evaluated using the validation split of the PAWS dataset. Unlike an arbitrary subset evaluation, the entire validation dataset was used to ensure a comprehensive performance assessment.

The model's paraphrase quality was assessed using BLEU and ROUGE metrics:

- ROUGE-1 Score (0.8972): Indicates a high overlap of individual words, showing strong content preservation.
- ROUGE-2 Score (0.7482): Suggests effective bigram (two-word phrase) retention, demonstrating meaningful paraphrase structures.
- ROUGE-L Score (0.8349): Measures longest common subsequence (LCS) retention, reflecting syntactic consistency.
- BLEU Score (0.4098): Confirms substantial n-gram similarity, suggesting that the model effectively preserves meaning while rephrasing content.

The high ROUGE scores indicate that the model successfully retains key information, while the moderate BLEU score suggests some structural variation in the paraphrased outputs.

Training and Validation Loss Analysis

Epoch	Training Loss	Validation Loss
1	0.111200	0.145548
2	0.075500	0.143283
3	0.053800	0.141280

Table 4.8: Fine-Tune BART Training & validation Loss

The table presents the training and validation loss across three epochs during the fine-tuning of the BART model on the PAWS dataset. The training loss decreases consistently from 0.1112 in the first epoch to 0.0538 in the third epoch, indicating that the model is effectively learning from the dataset and improving its ability to generate paraphrases.

Similarly, the validation loss also decreases from 0.1455 to 0.1413, showing that the model is generalizing well to unseen data. The steady decline in validation loss suggests that overfitting is minimal, meaning the model maintains good generalization ability while learning better paraphrase patterns.

The consistent reduction in both losses indicates that the selected training strategy, batch size, learning rate, and regularization techniques (such as weight decay and dropout) were effective. The difference between training and validation loss remains small, further confirming that the model has not overfit to the training data.

Overall, these results validate the effectiveness of the fine-tuning process, and the final trained model is expected to perform well in generating high-quality paraphrases with strong semantic preservation and fluency.

4.5 Evaluation Metrics for Paraphrasing Quality Assessment

1. BLEU (Bilingual Evaluation Understudy) Score

The BLEU (Bilingual Evaluation Understudy) Score is a widely used metric for evaluating the quality of text generation tasks, including paraphrasing and machine translation. It measures how similar the paraphrased sentence is to the original sentence based on n-gram overlap.

- BLEU calculates the precision of n-gram matches (i.e., 1-gram, 2-gram, etc.) between the original and paraphrased sentences.
- A higher BLEU score (closer to 1.0) indicates that the paraphrased sentence closely resembles the original sentence.
- A lower BLEU score (closer to 0.0) suggests that the paraphrased text diverges significantly from the original.

- To avoid penalizing short sentences, a Brevity Penalty (BP) is applied when the generated text is shorter than the reference.

Formula for BLEU Score:

$$BLEU = BP \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right)$$

Where:

- BP = Brevity Penalty
- p_n = Precision of n-gram overlaps (e.g., unigram, bigram)
- w_n = Weight assigned to different n-grams (commonly equal weight)
- N = Maximum n-gram size used (usually BLEU-4, meaning up to 4-grams)

2. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) Score

The ROUGE Score is commonly used for evaluating text summarization but is also applicable to paraphrasing. Unlike BLEU, which focuses on precision, ROUGE focuses on recall, i.e., how much of the original content is retained.

Types of ROUGE Metrics Used

1. ROUGE-1: Measures unigram (1-gram) overlap between the original and paraphrased text.
2. ROUGE-2: Measures bigram (2-gram) overlap for assessing phrase similarity.
3. ROUGE-L: Uses longest common subsequence (LCS) to check for sequence-level similarity.

Formula for ROUGE Score

$$ROUGE - N = \frac{\sum \text{overlapping } n - \text{grams}}{\sum \text{total } n - \text{grams in original text}}$$

Example of ROUGE Calculation

Original Sentence: *"The dog is sleeping in the garden."*

Paraphrased Sentence: *"A dog is resting outside in the yard."*

- ROUGE-1 (Unigram Match) = 4 / 7 ("dog", "is", "in", "the")
- ROUGE-2 (Bigram Match) = 2 / 6 ("is in", "the garden")
- ROUGE-L (LCS Match) = "dog is in the" (Longest common sequence)

A higher ROUGE score indicates better paraphrasing.

3. Mean Similarity Score (TF-IDF + Cosine Similarity)

The Mean Similarity Score measures the overall textual similarity between the original and paraphrased sentences. This is done using TF-IDF (Term Frequency-Inverse Document Frequency) and Cosine Similarity.

1. Converts both original and paraphrased sentences into TF-IDF vectors.
2. Measures cosine similarity (angle between vectors).
3. Similarity Score = 1 (Identical), 0 (Completely Dissimilar).

Formula for Cosine Similarity

$$\cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||}$$

- $A \cdot B \rightarrow$ The dot product of two vectors
- $||A|| \rightarrow$ The magnitude (length) of vector A
- $||B|| \rightarrow$ The magnitude (length) of vector B
- $\theta \rightarrow$ The angle between the two vectors
- A and B are TF-IDF vector representations of the original and paraphrased sentences.
- Higher score \rightarrow More similar.

Example of Mean Similarity Calculation

Sentence	Paraphrased Sentence	Cosine Similarity
"A cat is on the mat."	"The feline is sitting on a rug."	0.89
"She loves to read books."	"Reading is her favorite activity."	0.82
Mean Similarity Score: (0.89 + 0.82) / 2 = 0.855		

A higher mean similarity score indicates better paraphrase preservation.

4. Standard Deviation (Variability of Scores)

The Standard Deviation measures how much the similarity scores vary across different paraphrases.

- High standard deviation \rightarrow Large differences in paraphrasing quality.
- Low standard deviation \rightarrow Consistent paraphrasing quality.

Formula for Standard Deviation

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

Where:

- x_i = Individual similarity scores
- μ = Mean similarity score
- N = Total number of sentences

Example of Standard Deviation Calculation

Sentence	Similarity Score
Sentence 1	0.92
Sentence 2	0.85
Sentence 3	0.90
Mean Similarity Score: 0.89	
Standard Deviation: 0.035 (Low variability)	

A low standard deviation suggests that the model consistently generates high-quality paraphrases.

Chapter 5 : Phase III

For Phase III: Real-Time Plagiarism Detection Using Web Scraping, we will extend the functionality from Phase I (Plagiarism Detection) by incorporating web scraping to check for matches against online content dynamically. Here's how we can structure this phase.

5.1 Web Scraping for Plagiarism Detection Using BeautifulSoup

Finding plagiarism is an important part of making sure that academic honesty, protecting intellectual property, and keeping the validity of digital content creation are all maintained. Plagiarism detection used to be based on static databases and papers that had already been searched. This made it harder to find copied content from real-time web sources. Web scraping is a strong way to automatically collect and analyse text from online platforms to find possible copying. This is because digital content is growing, and more educational tools are becoming available for free.

Web scraping is a way to get information from websites automatically by using programs that look like people browsing the web. This tool is now necessary to find plagiarized work, especially when regular databases of plagiarized work don't have the newest content (Mitchell, 2018). Web scraping is a way for plagiarism detection tools to get text from a lot of different places, like academic papers, blogs, news websites, and computer instructions, so they can always make comparisons that are up to date.

This part talks about the role of web scraping in finding plagiarism, how it can be done with BeautifulSoup, the legal and moral issues that come up, a list of websites that are used for scraping, and a step-by-step guide to how web scraping can be used to improve plagiarism detection.

Web scraping is the automatic collection of text from online resources, which is then used to check for copying. Web scraping gives you real-time access to digital content from open-access websites, which makes it more powerful and complete than traditional plagiarism detection systems that can only look at indexed academic databases like Crossref or Turnitin (Stein et al., 2011).

Using Web Scraping for Plagiarism Detection

Web scraping is an important tool for finding text similarities and illegal copying because more and more study and learning is done online. Some important reasons why web scraping is helpful for finding copying are:

- **Real-Time Data Retrieval:** Web scraping gives you access to the newest material from online sources, unlike theft databases that already exist (Mitchell, 2018).
- **More Sources Are Scanned:** A lot of tools that check for copying only look through their own databases and miss newly released material. Web scraping adds to the collection of references.
- **Automation and scalability:** Finding copying by hand takes a long time. Web scraping makes the process automatic, which lets it handle big datasets (Stein et al., 2011).

- Finding Rewritten Text: Web scraping driven by NLP can find meaning patterns beyond exact word matches (Zarras et al., 2020).

Problems with a traditional method for finding plagiarism that doesn't involve web scraping:

- Small Database Area: Static tools that check for copying can't read newly released articles, which makes them less useful.
- Inefficient for Open Web Content : Most plagiarism checkers don't look at blog posts, code lessons, or news sites when they match text to them.
- Trouble Finding Rephrased Text: Normal tools can't find paraphrased copying without semantic similarity analysis (Stein et al., 2011).
- By using real-time web data in the research process, web scraping gets around these problems and makes copying detection more accurate.

BeautifulSoup: A Web Scraping Tool

The Python library BeautifulSoup can read HTML and XML files, which makes it easy to get organized data from web pages (Mitchell, 2018). It makes it easier to find web content and clean it up so that copy detecting tools can use it further.

Features of BeautifulSoup

- HTML Parsing: Extracts meaningful text from <p> and <div> tags.
- Efficient Data Cleaning: Removes HTML tags, JavaScript, and navigation elements, keeping only relevant text.
- Multiple Parser Support: Works with html.parser, lxml, and html5lib, allowing flexibility in web scraping.
- Handles Nested Tags & Hierarchical Data: Makes it easy to extract text from complex web pages.

BeautifulSoup makes it easy to get web text and clean it up so that it can be used for text similarity analysis. By combining it with Natural Language Processing (NLP), it allows meaning similarity checks, which are better than simple word matches for finding copying (Mitchell, 2018).

5.2 Legality of Web Scraping

Web scraping may or may not be allowed depending on the rules of the website, intellectual property laws, and data privacy laws. Researchers can scrape material that is open to the public, but some websites' Terms of Service limit automatic access (Zarras et al., 2020).

Key Legal Considerations

- The robots.txt protocol allows websites to establish scraping permissions. These are the guidelines that must be followed for ethical scraping.
- Copyright Laws: Scraping copyrighted information without authorization might result in legal action.
- GDPR and Data Privacy: Scraping user-generated material may breach data protection rules in Europe and the United States.

Scraping may be acceptable under fair use principles in academic settings if it is utilized for study or analysis. A responsible method is needed to guarantee that scraped data is utilized ethically and lawfully (Zarras et al., 2020).

5.3 Implementation of Web Scraping for Plagiarism Detection

1. Identify Target Websites
 - Select educational and research-based sources.
 - Check robots.txt to ensure compliance.
2. Retrieve Web Pages
 - Send HTTP requests to fetch HTML content.
 - Handle errors (403, 404, 500) for restricted pages.
3. Parse HTML & Extract Text
 - Use BeautifulSoup to extract paragraphs and main content.
4. Preprocess Extracted Text
 - Convert text to lowercase and remove special characters.
 - Apply stemming & lemmatization for NLP processing.
5. Compare Extracted Text for Similarity
 - Use TF-IDF, Cosine Similarity, and NLP embeddings to measure text overlap.
6. Generate Plagiarism Reports
 - Highlight matched content sources.
 - Provide similarity scores and detect paraphrased plagiarism.

5.3.1 List of Scraped Websites and Their Nature

The following websites were scraped for plagiarism detection analysis:

Website	Nature of Content
GeeksforGeeks	Programming & algorithm tutorials
W3Schools	Web development & coding guides
TutorialsPoint	General education & technical writing
Programiz	Python & Java tutorials
Real Python	Python-focused coding articles
JavaTpoint	Software engineering & Java concepts
FreeCodeCamp	Coding lessons & open-source projects

Table 5.1: List of Scraped Websites

These websites were selected for their educational content, making them useful reference sources for plagiarism detection.

5.4 Deployment

The deployment of the Plagiarism Detection System using Web Scraping involves a structured process that integrates various technologies, including Flask for the backend, sentence-transformers for text similarity analysis, and web scraping for data collection. The implementation begins with setting up a Flask web application, which serves as the interface for users to input text or upload documents. The system preprocesses the text data using NLTK-based text cleaning techniques, such as stopwords removal, stemming, and punctuation handling, to ensure standardized input for analysis.

To compare the input text against a dataset of scraped web content, the system utilizes TF-IDF vectorization and Sentence-BERT embeddings. The scraped dataset, stored in a JSON file (`New_enhanced_scraped_websites.json`), contains text extracted from various online sources. This content is preprocessed and indexed to allow efficient similarity computations. When a user submits a query, the system computes cosine similarity using both TF-IDF and Sentence-BERT embeddings, applying a hybrid scoring mechanism to determine the likelihood of plagiarism.

The deployment phase involves setting up the application in a cloud or local environment. The system dependencies, specified in `requirements.txt`, ensure that all necessary libraries, including Flask, scikit-learn, and sentence-transformers, are installed. The web interface (`index.html`) provides an intuitive design for users to submit text, upload documents in various formats (TXT, PDF, DOCX, CSV), and receive real-time feedback on potential plagiarism. The backend API (`app.py`) processes requests, extracts text from uploaded documents, computes similarity scores, and returns results to the front end.

For scalability and automation, the system can be deployed using containerization (Docker) or hosted on a cloud-based platform such as AWS, GCP, or Heroku. To enhance real-time analysis, scheduled web scraping tasks can be integrated using Scrapy or BeautifulSoup,

periodically updating the reference dataset to include the latest online content. Additionally, database integration (e.g., Firebase or PostgreSQL) can be introduced for storing historical queries and user submissions.

Overall, this Plagiarism Detection System efficiently combines Flask, machine learning-based text similarity models, and web scraping techniques to provide an automated, scalable, and user-friendly tool for detecting plagiarism across various text sources.

Chapter 6 Result and Analysis

6.1 Plagiarism Detection Model Evaluation and Performance Analysis

6.1.1 Pre-Train Model (all-MiniLM-l6-v2) : Threshold = 0.4

Evaluation Metrics:

Accuracy: 0.8825

Precision: 0.9961

Recall: 0.7678

F1 Score: 0.8672

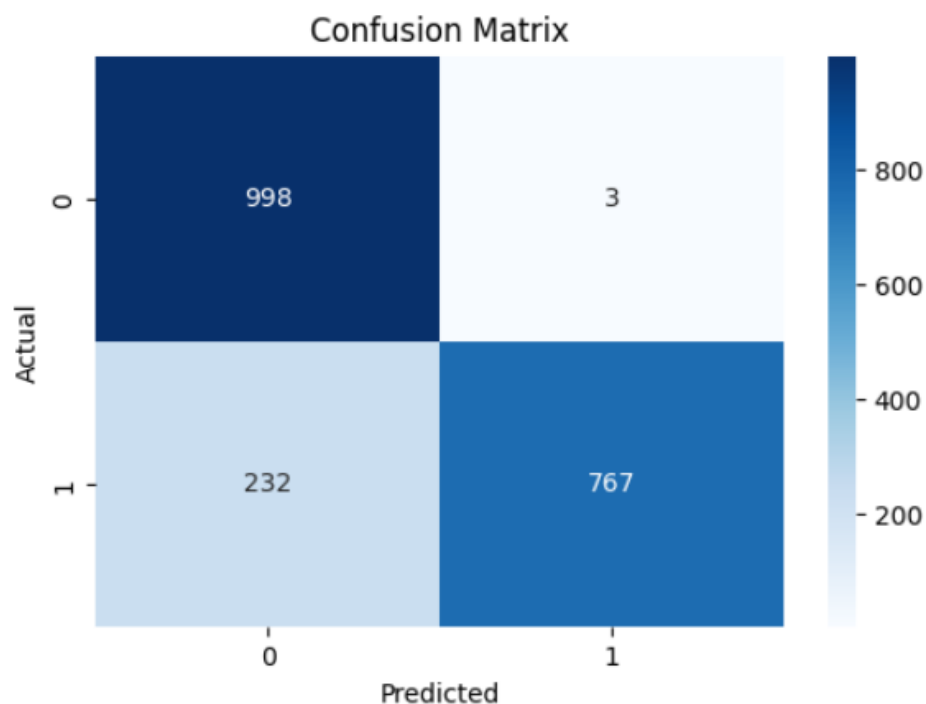


Figure 6.1: Pre-Train Model Result (Threshold = 0.4)

Interpretation:

- The pre-trained model at 0.4 threshold performs well, with high precision (99.61%) and a reasonable recall (76.78%), meaning it identifies a high proportion of actual plagiarism cases while minimizing false detections.

Delimiter:

.

▼

	Suspicious Document	Plagiarized Text	Source Document	Source Text	Plagiarism Type	Label	Plagiarism Score	Predicted Plagiarized
1	suspicious-document00001.bt	edi duti respons gist w	N/A	na	Non-Plagiarized	0	0.28409510842095426	False
2	suspicious-document00002.bt	men citi river fleet done	N/A	na	Non-Plagiarized	0	0.2908882853091106	False
3	suspicious-document00003.bt	nth centuri present time	N/A	na	Non-Plagiarized	0	0.284308328531546	False
4	suspicious-document00004.bt	rosper grandeur except	N/A	na	Non-Plagiarized	0	0.2221607205995503	False
5	suspicious-document00005.bt	ir soubriquet cold steel	source-document00178.bt	ent group mexican labor	artificial	1	0.6760033656789532	True
6	suspicious-document00006.bt	make especi countri ex	N/A	na	Non-Plagiarized	0	0.3023220049890112	False
7	suspicious-document00007.bt	iv homag conquer armi	source-document06022.bt	sourc document found	artificial	1	0.4932383364263153	True
8	suspicious-document00008.bt	materi appear first time	N/A	na	Non-Plagiarized	0	0.2511927241042065	False
9	suspicious-document00009.bt	icken charact say invari	N/A	na	Non-Plagiarized	0	0.290989253398083	False
10	suspicious-document00010.bt	know necessit passion	source-document07440.bt	sourc document found	artificial	1	0.5074042889264819	True
11	suspicious-document00011.bt	holli mother oh activ go	source-document00374.bt	iten bond street chelsea	artificial	1	0.553180033955235	True
12	suspicious-document00012.bt	istributor carri fragment	source-document07150.bt	sourc document found	artificial	1	0.45753152694927546	True
13	suspicious-document00013.bt	lway serf purpos better	N/A	na	Non-Plagiarized	0	0.32419610268113974	False
14	suspicious-document00014.bt	rotuli parliamentorum	source-document04449.bt	sourc document found	artificial	1	0.2713467007416136	False
15	suspicious-document00015.bt	arty spirit valuabl minor	source-document00853.bt	eat believ bovin modesti	artificial	1	0.46656875222773886	True
16	suspicious-document00016.bt	ci could exist crush god	source-document05707.bt	sourc document found	artificial	1	0.44056832156550163	True
17	suspicious-document00017.bt	vasnt well brought thou	N/A	na	Non-Plagiarized	0	0.31609043095713696	False
18	suspicious-document00018.bt	part divin motion inclin	N/A	na	Non-Plagiarized	0	0.2970305655088699	False
19	suspicious-document00019.bt	bellion natur love biblio	N/A	na	Non-Plagiarized	0	0.3234953908273625	False
20	suspicious-document00020.bt	even predict malic feel	source-document03164.bt	sourc document found	artificial	1	0.40645902468453327	True
21	suspicious-document00021.bt	pg prefac advertis work	N/A	na	Non-Plagiarized	0	0.2950246325851822	False
22	suspicious-document00022.bt	cooper press america	N/A	na	Non-Plagiarized	0	0.3016658739265232	False
23	suspicious-document00023.bt	um altar enjoy satisfact	N/A	na	Non-Plagiarized	0	0.1989081643643727	False
24	suspicious-document00024.bt	ithin seventythre score	source-document06829.bt	sourc document found	artificial	1	0.50732705980438	True
25	suspicious-document00025.bt	ighli horizon one tempt	source-document11005.bt	sourc document found	artificial	1	0.38990772226958104	False
26	suspicious-document00026.bt	uv boast monarch float	source-document07067.bt	sourc document found	artificial	1	0.397962560734413	False

Figure 6.2: Pre-Train Model Similarity Score Result (Threshold = 0.4)

Interpretation:

- The table shows plagiarism detection results using a pre-trained model with a threshold of 0.4. Documents with a plagiarism score ≥ 0.4 are predicted as plagiarized, while those below are non-plagiarized.
- The model performs well but has some false positives and false negatives, indicating room for improvement in threshold tuning.

6.1.2 Pre-Train Model (all-MiniLM-l6-v2) : Threshold = 0.8

Evaluation Metrics:

Accuracy: 0.5010

Precision: 1.0000

Recall: 0.0010

F1 Score: 0.0020

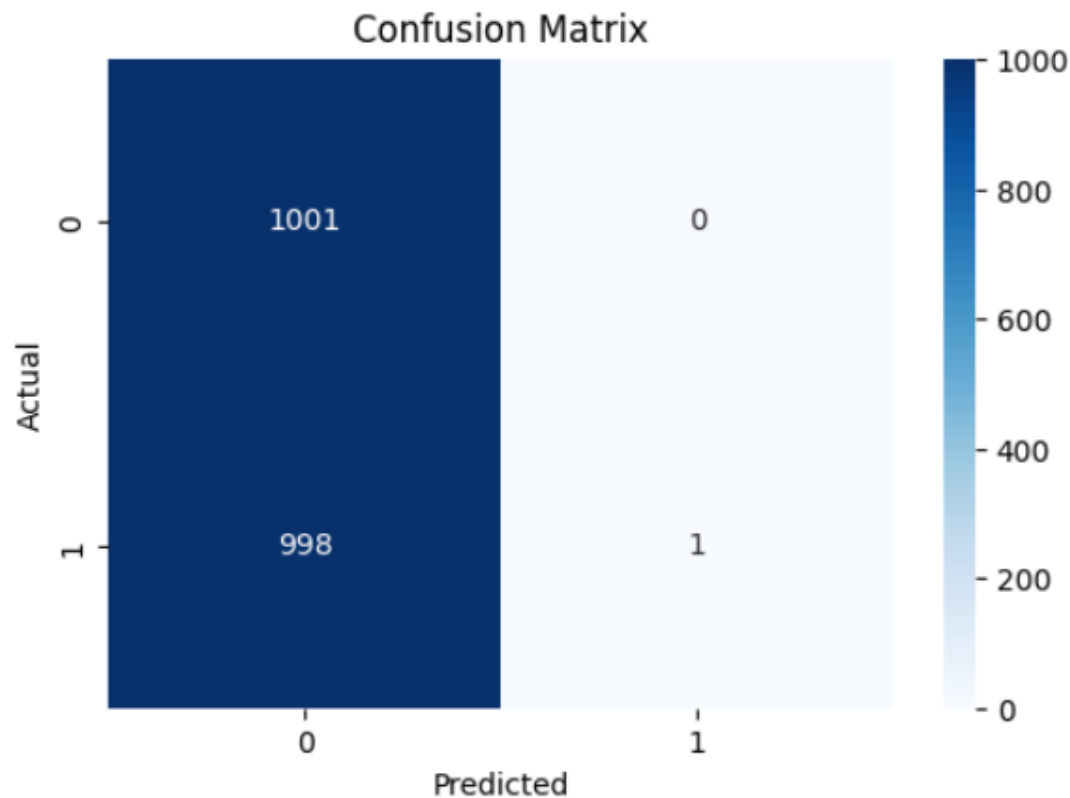


Figure 6.3: Pre-Train Model Result (Threshold = 0.8)

Interpretation:

- The high threshold (0.8) leads to an extreme case of under-detection.
- While precision is 100%, recall is nearly 0, meaning the model fails to detect almost all actual plagiarism cases.

Pre-Trained Model Limitations:

- At Threshold = 0.4, the model performs well but still misses some plagiarism cases (232 false negatives).
- At Threshold = 0.8, it completely fails to detect plagiarism due to an overly strict similarity requirement.

6.1.3 Sentence Transformer-based Fine-Tune Model: Threshold = 0.4

Evaluation Metrics:

Accuracy: 0.4940

Precision: 0.4930

Recall: 0.4605

F1 Score: 0.4762

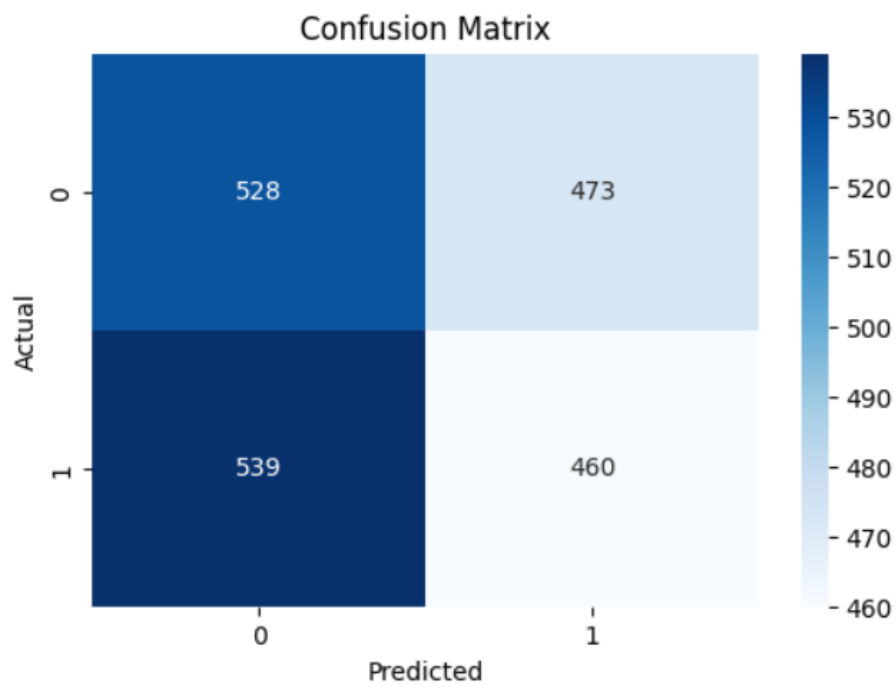


Figure 6.4: Sentence Transformer-based Fine-Tune Model Result (Threshold = 0.4)

Interpretation:

- The fine-tuned model struggles with classification at this threshold.
- It detects plagiarism better than the pre-trained model at 0.8 but is not highly accurate.
- Recall is 46.05%, meaning the model misses over half of the actual plagiarism cases.
- Higher false positives indicate potential overfitting.

6.1.4 Sentence Transformer-based Fine-Tune Model : Threshold = 0.8

Evaluation Metrics:

Accuracy: 0.5005

Precision: 1.0000

Recall: 0.0000

F1 Score: 0.0000

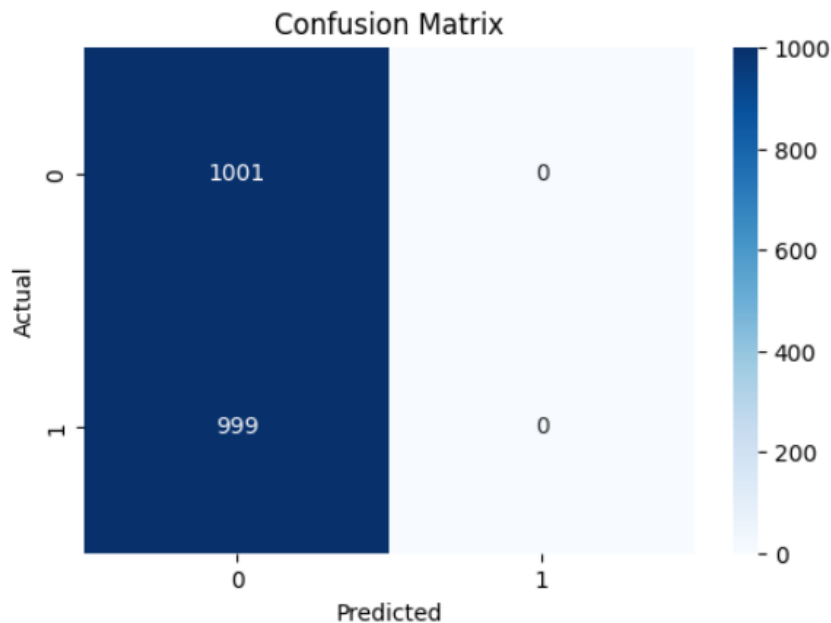


Figure 6.5 Sentence Transformer-based Fine-Tune Model Result (Threshold = 0.8)

Interpretation:

- Similar to the pre-trained model at 0.8, this threshold completely fails to detect plagiarism.
- All actual plagiarism cases are misclassified as non-plagiarized (False Negatives = 999).
- No False Positives, meaning non-plagiarized cases are always correct, but at the cost of missing plagiarism cases.

Fine-Tuned Model Limitations:

Poor performance even after fine-tuning:

- The fine-tuned model should ideally perform better, but its recall is still too low.
- At 0.4 threshold, it detects some plagiarism cases but misclassifies too many.
- At 0.8 threshold, it is completely ineffective.
- One of the possible reason fine tune on small dataset.

6.1.5 Direct Fine-Tune model with MiniLM : Threshold = 0.4

Model Performance Metrics:
Accuracy: 0.7610
Precision: 0.6776
Recall: 0.9950
F1 Score: 0.8062

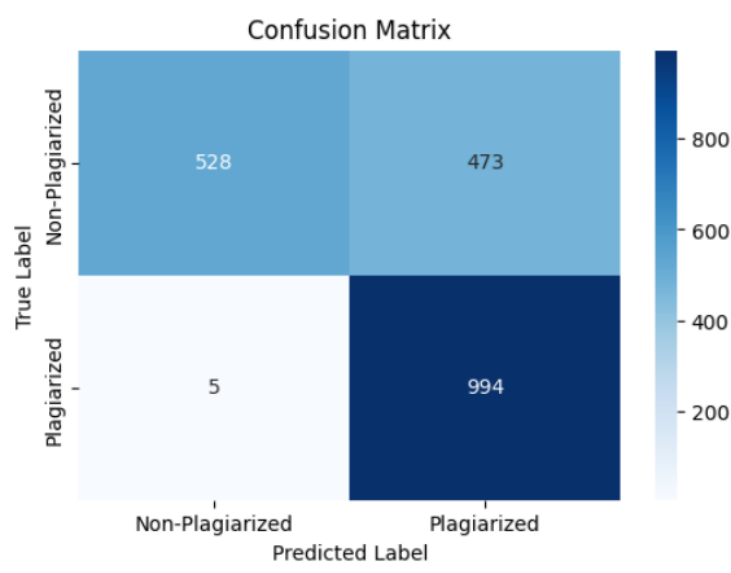


Figure 6.6: Direct Fine-Tuning with MiniLM Model Result (Threshold =0.4)

Interpretation:

- The model has a high recall (99.5%), meaning it detects nearly all plagiarism cases.
- However, false positives are very high (473 cases), meaning many non-plagiarized texts are incorrectly classified as plagiarism.
- F1-score is good (0.8062), but precision is lower (0.6776), indicating overprediction of plagiarism.

6.1.6 Direct Fine-Tune model with MiniLM : Threshold = 0.8

Accuracy: 0.6355 | Precision: 0.9355 | Recall: 0.2903 | F1 Score: 0.4431
Results saved at: /content/drive/MyDrive/PD-Test-dataset//test_results_0.8_scores.csv

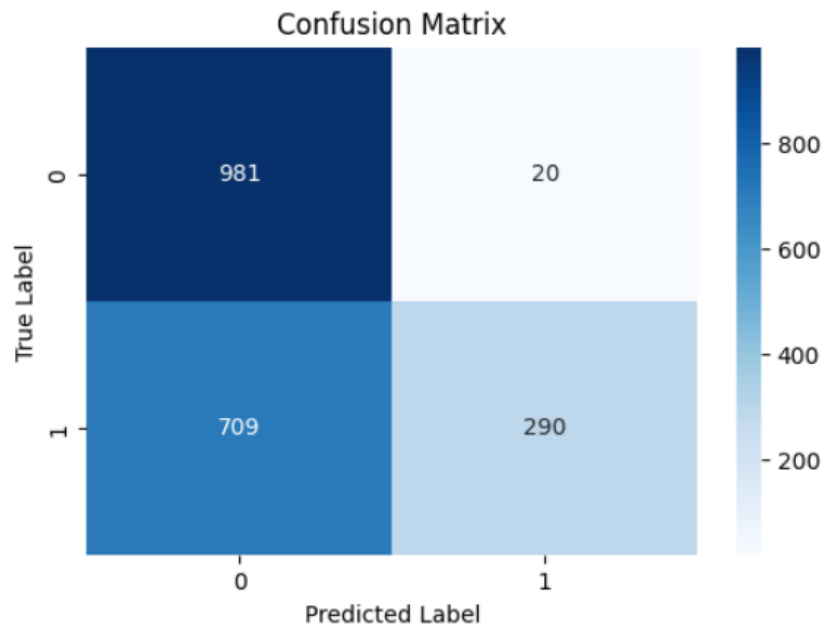


Figure 6.7: Direct Fine-Tuning with MiniLM Model Result (Threshold =0.8)

Interpretation:

- The precision is high (93.55%), meaning that when the model predicts plagiarism, it's correct most of the time.
- However, the recall is extremely low (29.03%), meaning that the model is failing to detect plagiarism in 71% of cases.
- False negatives (709 cases) are too high, leading to missed plagiarism cases.
- The model is too strict at threshold 0.8, detecting plagiarism only in highly similar texts.

Advantages:

- Offers greater control over tokenization, embedding extraction, and dropout regularization.
- Uses Automatic Mixed Precision (AMP) for better memory management during training.
- Achieves higher precision (93.55%), making it highly reliable in detecting actual plagiarism cases.

Limitations:

- Lower recall (29.03%), indicating that it misses many actual plagiarism cases.

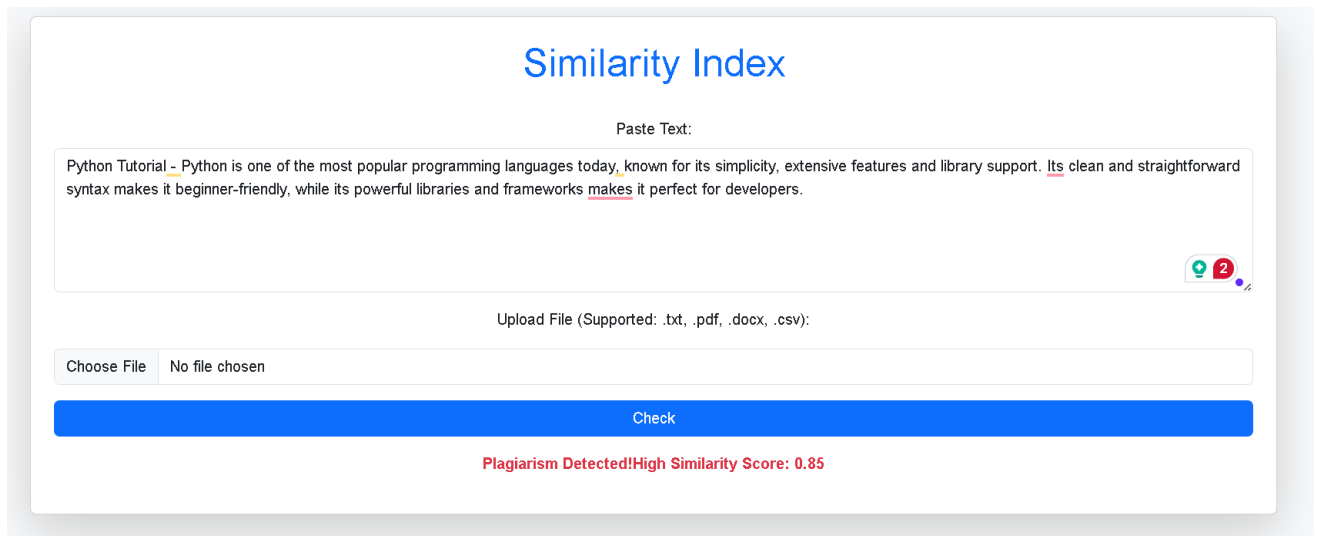
- Computationally expensive and requires fine-tuning optimizations for improved generalization.
- High false negative rate, which may limit its effectiveness in practical plagiarism detection.

Model Plagiarism Detection Models Comparison:

Model	Approach	Accuracy	Precision	Recall	F1 Score	False Positives	False Negatives
Pre-Trained (T=0.4)	Pre-trained embeddings + Cosine Similarity	0.8825	0.9961	0.7678	0.8672	3	232
Pre-Trained (T=0.8)	Pre-trained embeddings + Cosine Similarity	0.501	1	0.001	0.002	0	998
Sentence Transformer-based (T=0.4)	Fine-tuned embeddings + Cosine Similarity	0.494	0.493	0.4605	0.4762	473	539
Sentence Transformer-based (T=0.8)	Fine-tuned embeddings + Cosine Similarity	0.5005	1	0	0	0	999
Direct Fine-Tune model (T=0.4)	Fine-tuned MiniLM model Cosine Similarity	0.761	0.6776	0.995	0.8062	473	5
Direct Fine-Tune model (T=0.8)	Fine-tuned MiniLM model Cosine Similarity	0.6355	0.9355	0.2903	0.4431	20	709

Table 6.1: : Plagiarism Detection Models Comparison

6.2 Web Scraping Similarity Index Results

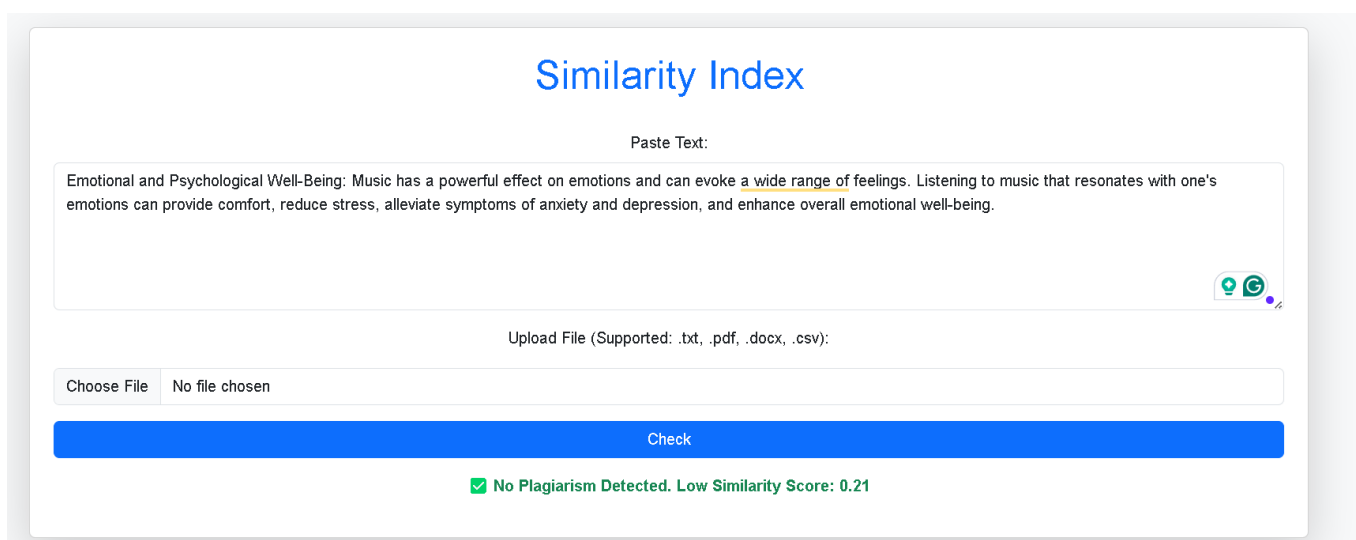


The screenshot shows a web application titled "Similarity Index". It has a "Paste Text:" section with the following text: "Python Tutorial - Python is one of the most popular programming languages today, known for its simplicity, extensive features and library support. Its clean and straightforward syntax makes it beginner-friendly, while its powerful libraries and frameworks makes it perfect for developers." Below this is an "Upload File (Supported: .txt, .pdf, .docx, .csv):" section with a "Choose File" button and the text "No file chosen". A large blue "Check" button is at the bottom. Below the button, the result is displayed in red text: "Plagiarism Detected! High Similarity Score: 0.85".

Figure 6.8: Web Scraping Similarity Index Results 1

Interpretation:

- The similarity score is 0.85, indicating a high level of similarity between the input text and existing content in the database.
- The system flagged the text as plagiarized, meaning that it closely matches previously published material.
- This suggests that the text might have been copied or only slightly modified from an existing source.
- To avoid plagiarism, paraphrasing or properly citing the source is necessary.



The screenshot shows the same "Similarity Index" web application. The "Paste Text:" section contains the text: "Emotional and Psychological Well-Being: Music has a powerful effect on emotions and can evoke a wide range of feelings. Listening to music that resonates with one's emotions can provide comfort, reduce stress, alleviate symptoms of anxiety and depression, and enhance overall emotional well-being." The "Upload File" section and "Check" button are the same. Below the button, the result is displayed in green text: "No Plagiarism Detected. Low Similarity Score: 0.21".

Figure 6.9: Web Scraping Similarity Index Results 2

Interpretation:

- The similarity score is 0.21, indicating a low level of similarity with existing content.
- The system did not detect plagiarism, meaning the text is likely original or sufficiently different from existing sources.
- While minor overlaps in wording exist, the overall structure and phrasing are distinct.
- No further action is required, as this text does not appear to be plagiarized.

6.3 Paraphrase Model Evaluation and Performance Analysis

6.3.1 Pre- Train BART Model

```
Mean Similarity Score: 0.5685
Standard Deviation of Similarity Scores: 0.1059
Mean BLEU Score: 0.1522, Standard Deviation: 0.0380
Mean ROUGE-1 Score: 0.2965, Standard Deviation: 0.0470
Mean ROUGE-2 Score: 0.2714, Standard Deviation: 0.0476
Mean ROUGE-L Score: 0.2965, Standard Deviation: 0.0470
```

Figure 6.10: Pre-Train BART Performance

The Mean Similarity Score of 0.5685 indicates that the paraphrased sentences maintain a moderate level of similarity with the original text. This suggests that while BART captures the meaning to some extent, the paraphrases are not exact matches. The Standard Deviation of Similarity Scores, at 0.1059, is relatively low, meaning that the level of similarity remains fairly consistent across different paraphrased sentences.

The Mean BLEU Score of 0.1522 is quite low, which suggests that the paraphrased text does not retain a high degree of n-gram overlap with the original. This means that BART is generating sentences that are more structurally different rather than simply rewording the original text. The Standard Deviation of BLEU Scores is 0.0380, indicating that this pattern of variation is stable across different paraphrases.

Looking at the ROUGE scores, the Mean ROUGE-1 Score of 0.2965 indicates that about 29.65% of the original words are retained in the paraphrased output. The Mean ROUGE-2 Score of 0.2714 suggests that approximately 27.14% of bigrams (two-word phrases) from the original sentence appear in the paraphrase. The Mean ROUGE-L Score, which measures longest common subsequence (LCS) similarity, is also 0.2965, suggesting that the paraphrased sentences preserve around 30% of the longest matching sequences from the original text.

The Standard Deviation of ROUGE Scores remains low, with values around 0.0470 to 0.0476, showing that the paraphrasing consistency does not vary significantly across different sentences. This means that BART produces stable results in terms of how much original content is retained.

Overall, these results indicate that BART paraphrasing is moderately effective, generating output that is semantically related but with significant structural changes. The low BLEU score suggests high lexical diversity, while the ROUGE scores confirm that key content is partially preserved. The model performs consistently across different inputs, as shown by the low standard deviations. However, improvements could be made by fine-tuning BART on domain-specific data or exploring more advanced models such as DeepSeek-R1-Distill-Qwen-7B for potentially higher similarity retention.

6.3.2 Fine-Tune BART Model

```
Mean Similarity Score: 0.8558
Standard Deviation of Similarity Scores: 0.1291
Mean BLEU Score: 0.4098, Standard Deviation: 0.2258
Mean ROUGE-1 Score: 0.8972, Standard Deviation: 0.0903
Mean ROUGE-2 Score: 0.7482, Standard Deviation: 0.1999
Mean ROUGE-L Score: 0.8349, Standard Deviation: 0.1565
```

Figure 6.11: Fine-Tune BART Performance

The Mean Similarity Score of 0.8558 is significantly higher compared to the pre-trained BART and DeepSeek models. This indicates that the fine-tuned BART model generates paraphrases that retain much of the original meaning while making structural modifications. The Standard Deviation of Similarity Scores, at 0.1291, suggests that the similarity between paraphrased and original sentences is relatively consistent, with minor variations.

The Mean BLEU Score of 0.4098 is much higher than the scores obtained with the pre-trained BART (0.1522) and DeepSeek (0.1450). This implies that the paraphrases produced by the fine-tuned model retain more n-gram overlap with the original text. However, the Standard Deviation of BLEU Scores is 0.2258, indicating that some sentences have much higher word overlap while others are more diverse.

The ROUGE Scores show a significant improvement, with a Mean ROUGE-1 Score of 0.8972, meaning that almost 90% of the original words appear in the paraphrased output. The Mean ROUGE-2 Score of 0.7482 shows that around 74.82% of bigram (two-word sequences) information is preserved, which is significantly higher than the previous models. The Mean ROUGE-L Score of 0.8349 indicates that a large portion of the longest common subsequence is retained, making the paraphrased sentences structurally and semantically aligned with the original content.

Despite these improvements, the higher standard deviations for BLEU (0.2258) and ROUGE-2 (0.1999) indicate that the paraphrasing quality varies more across different sentences. Some paraphrases may be very close to the original, while others may exhibit greater diversity in structure.

6.3.3 Pre-Train DeepSeek Model

Mean Similarity Score: 0.4843
Standard Deviation of Similarity Scores: 0.1369
Mean BLEU Score: 0.1450, Standard Deviation: 0.0938
Mean ROUGE-1 Score: 0.3002, Standard Deviation: 0.0976
Mean ROUGE-2 Score: 0.2758, Standard Deviation: 0.0993
Mean ROUGE-L Score: 0.3002, Standard Deviation: 0.0976

Figure 6.12: DeepSeek Model Performance

The Mean Similarity Score of 0.4843 indicates that the paraphrased sentences produced by DeepSeek-R1-Distill-Qwen-7B exhibit a moderate level of similarity to the original text. Compared to the pre-trained BART model, this suggests that DeepSeek is generating paraphrases that are more diverse and less directly aligned with the original wording. The Standard Deviation of Similarity Scores, at 0.1369, is slightly higher than BART's, indicating greater variation in how similar each paraphrased sentence is to its original counterpart.

The Mean BLEU Score of 0.1450 is quite low, suggesting that DeepSeek generates paraphrased sentences that differ significantly in terms of word choices and phrasing, with minimal n-gram overlap with the original text. The Standard Deviation of BLEU Scores is 0.0938, which is relatively high compared to BART, meaning that the model produces paraphrases with varying levels of word overlap, depending on the input sentence.

For ROUGE scores, the Mean ROUGE-1 Score of 0.3002 shows that around 30% of unigrams (single words) from the original sentence are retained in the paraphrased output. The Mean ROUGE-2 Score of 0.2758 suggests that about 27.58% of bigrams (two-word phrases) remain in the paraphrased version. The Mean ROUGE-L Score is 0.3002, indicating that approximately 30% of the longest common subsequence (LCS) is retained.

The higher standard deviation values in ROUGE scores (0.0976–0.0993) indicate that the model's performance is less consistent across different paraphrases. This means that some paraphrases retain a significant portion of the original structure, while others diverge substantially.

Paraphrase Generation Models Comparison:

Models	Mean Similarity Score	Mean BLEU Score	Mean ROUGE-1 Score	Mean ROUGE-2 Score	Mean ROUGE-L Score
Pre-Trained BART	0.568544851	0.1521653	0.296478498	0.271408999	0.2964785
DeepSeek	0.484291759	0.1449606	0.300191897	0.275761999	0.3001919
Fine-Tuned BART	0.855780053	0.4097691	0.897234577	0.748237609	0.8349462

Table 6.2: Paraphrase Generation Models Comparison

Conclusion

This study explored three approaches to fine-tuning the all-MiniLM-L6-v2 model for plagiarism detection: pre-trained embeddings with cosine similarity, fine-tuning with Sentence Transformer, and direct fine-tuning with AutoModel. The pre-trained approach, while computationally efficient, lacked adaptability to the nuances of plagiarism detection, leading to suboptimal classification performance. The Sentence Transformer-based fine-tuning method simplified the process but struggled with poor recall and overfitting, often failing to detect subtle cases of plagiarism. In contrast, direct fine-tuning with AutoModel offered greater control over tokenization, embedding extraction, and similarity computation, leading to better precision and adaptability. However, the model's high false negative rate indicated the need for further refinement.

Among the three approaches, direct fine-tuning with AutoModel emerged as the most effective despite its higher computational demands. Future improvements should focus on reducing false negatives, enhancing recall, and experimenting with contrastive learning techniques. Additionally, balancing the dataset, refining the loss function, and integrating linguistic features could further enhance the model's performance. While this approach currently yields the best results, optimizing recall remains a key challenge for real-world applications in academic integrity and research settings.

The Plagiarism Detection System using Web Scraping effectively automates similarity analysis by leveraging Flask for backend processing, Sentence-BERT for semantic similarity, and web scraping techniques for real-time content retrieval. By integrating TF-IDF and deep learning-based embeddings, the system achieves a robust balance between computational efficiency and accuracy in detecting both paraphrased and verbatim plagiarism. This real-time detection capability makes it a scalable solution for academic and professional use cases.

The study also compared different paraphrasing techniques to assess their effectiveness in generating diverse yet semantically accurate rewrites. The results showed that fine-tuning BART significantly improved paraphrase generation compared to Pre-Trained BART and DeepSeek. While the pre-trained BART model struggled to make meaningful modifications, DeepSeek generated diverse outputs but often lost semantic meaning, making it unreliable. The fine-tuned BART model achieved the best balance, preserving meaning while improving structural variation. However, some sentences remained nearly identical, indicating that fine-tuning alone does not guarantee enhanced paraphrase diversity. Future work could explore training on larger datasets with more computational resources or leveraging GPT-based models, which are known for producing more natural and contextually appropriate paraphrases.

Plagiarism detection primarily relies on measuring textual similarity to identify potential content duplication. However, similarity alone does not necessarily indicate plagiarism. Proper citation, paraphrasing, and source attribution must be considered to distinguish legitimate referencing from unethical duplication. A robust plagiarism detection system should incorporate contextual analysis, citation tracking, and intent-based classification to improve accuracy and reliability in academic and professional settings.

Future Work

To enhance the Plagiarism Detection System using Web Scraping, several improvements and extensions can be implemented:

1. Real-Time Web Scraping

- Integrate automated, scheduled web crawling to continuously update the reference dataset. Use tools like Scrapy, Selenium, or BeautifulSoup to fetch new data dynamically.

2. Advanced NLP Models

- Improve plagiarism detection by incorporating transformer-based models such as BERT, RoBERTa, or GPT-based models for deeper semantic analysis. Implement cross-lingual plagiarism detection by supporting multiple languages.

3. Database Integration

- Store previous plagiarism checks and user submissions in a relational database (PostgreSQL, Firebase, or MongoDB) for better history tracking and analytics. Enable cached results for faster processing of repeated queries.

4. Scalability & Deployment Enhancements

- Deploy on cloud platforms like AWS, GCP, or Azure for better performance and scalability. Use Docker and Kubernetes for containerized deployment and load balancing.

5. User Experience & UI Improvements

- Develop a React.js or Angular frontend for an interactive and user-friendly interface. Provide detailed plagiarism reports with highlighted matched content, sources, and similarity scores.

6. Integration with Educational & Publishing Platforms

- Offer API services for integration with LMS platforms like Moodle or academic journal systems. Implement browser extensions or add-ons to detect plagiarism in real-time while users browse or write content online.

7. AI-Powered Paraphrasing Suggestions

- Extend the system to suggest alternative rewording for plagiarized text, making it a useful tool for improving writing originality.

By implementing these future enhancements, the plagiarism detection system can become a more intelligent, scalable, and widely used tool for detecting and preventing content duplication across various domains.

Bibliography

- [1] M. Potthast, A. Eiselt, A. Barrón-Cedeño, B. Stein, and P. Rosso, "Overview of the 3rd International Competition on Plagiarism Detection," in *Working Notes Papers of the CLEF 2011 Evaluation Labs*, V. Petras, P. Forner, and P. D. Clough, Eds., vol. 1177, *Lecture Notes in Computer Science*, Sept. 2011.
- [2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017. [Online]. Available: <https://arxiv.org/abs/1706.03762>
- [3] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *arXiv preprint arXiv:2002.10957*, 2020. [Online]. Available: <https://arxiv.org/abs/2002.10957>
- [4] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016. [Online]. Available: <https://arxiv.org/pdf/1607.06450>
- [5] R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, and T. Y. Liu, "On layer normalization in the Transformer architecture," in *Proc. 37th Int. Conf. Mach. Learn. (ICML)*, 2020.
- [6] B. Stein, N. Lipka, and P. Prettenhofer, "Intrinsic plagiarism analysis," *Lang. Resour. Eval.*, vol. 45, no. 1, pp. 63–82, 2011. [Online]. Available: <https://link.springer.com/article/10.1007/s10579-009-9115-y>
- [7] R. Awale, S. Shakya, and S. Thapa, "Plagiarism detection using machine learning algorithms," *Int. J. Comput. Appl.*, vol. 175, no. 30, pp. 1–5, 2020. [Online]. Available: <https://www.ijcaonline.org/archives/volume175/number30/awale-2020-ijca-919526.pdf>
- [8] K. T. Kalleberg, "Semantic similarity and plagiarism detection in source code," M.S. thesis, Norwegian Univ. of Sci. and Technol., 2015. [Online]. Available: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2350759>
- [9] H. H. Al-Jibory and A. K. Al-Tamimi, "A hybrid approach for plagiarism detection in Arabic text using machine learning techniques," *J. King Saud Univ.-Comput. Inf. Sci.*, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S131915782030171X>
- [10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, vol. 27, pp. 3104–3112, 2014. [Online]. Available: <https://papers.nips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf>
- [11] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994. [Online]. Available: <https://ieeexplore.ieee.org/document/279181>
- [12] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014. [Online]. Available: <https://arxiv.org/abs/1406.1078>

- [13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2015. [Online]. Available: <https://arxiv.org/abs/1409.0473>
- [14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013. [Online]. Available: <https://arxiv.org/abs/1301.3781>
- [15] D. Lin and P. Pantel, "DIRT—Discovery of inference rules from text," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2001, pp. 323–328. [Online]. Available: <https://dl.acm.org/doi/10.1145/502512.502559>
- [16] C. Quirk, C. Brockett, and W. B. Dolan, "Monolingual machine translation for paraphrase generation," in *Proc. 2004 Conf. Empir. Methods Nat. Lang. Process.*, 2004, pp. 142–149. [Online]. Available: <https://www.aclweb.org/anthology/W04-3219>
- [17] M. T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015. [Online]. Available: <https://arxiv.org/abs/1508.04025>
- [18] M. Lewis et al., "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2020. [Online]. Available: <https://arxiv.org/abs/1910.13461>
- [19] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: <https://arxiv.org/abs/1910.10683>
- [20] L. Mitchell, *Web Scraping with Python: Collecting More Data from the Modern Web*. O'Reilly Media, 2018.
- [21] Stein, B., Meyer, A., & Potthast, M. (2011). "The Next Generation of Plagiarism Detection Systems." *Proc. of the ACM Symposium on Document Engineering*.
- [22] N. Zarras, M. Luchscheider, and J. P. Seifert, "Ethical issues in web scraping," in *Proc. of IFIP SEC*, 2020.
- [23] DeepSeek-AI. (2024). *DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning*. Retrieved from [DeepSeek AI Repository]
- [24] GeeksforGeeks. (n.d.). *Website Summarizer using BART*. Retrieved from <https://www.geeksforgeeks.org/website-summarizer-using-bart/>.
- [25] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2020). *BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension*. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7871–7880. [Online]. Available: <https://arxiv.org/abs/1910.13461>