Détection des éléments générés automatiquement Contenu académique

Vijini Pilana Liyanage

UNIVERSITÉ PARIS XIII - SORBONNE PARIS NORD

École Doctorale Galilée

Laboratoire d'Informatique de Paris Nord

Detection of Automatically Generated Academic Content

Doctoral thesis

Defended by

Vijini LIYANAGE

For obtaining the degree of

Doctor in Computer Science

Defend on 16/05/2024 before the jury composed of:

Cyril LABBÉ ..... Rapporteur
Didier SCHWAB ..... Rapporteur
Karen FÖRT ..... Examiner
Adeline NAZARENKO ..... Thesis Director
Davide BUSCALDI ..... Thesis co-supervisor

I would like to dedicate this thesis to my loving husband ...

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Acknowledgement

First of all, I would like to thank my esteemed supervisors Prof. Davide Buscaldi and Prof. Adeline Nazerenko for their invaluable supervision, support and tutelage during the course of my PhD degree. Especially prof. Davide should be further appreciated for the immense assistance he has provided us from day one in France, not only for my PhD, but also in building our lives in Paris. I also could not have undertaken this journey without my defense committee, who generously provided knowledge and expertise. Additionally, this endeavor would not have been possible without the generous support from the Institute of Technology at University Sorbonne Paris Nord and its staff. Moreover I would like to express my gratitude to all the colleagues of Laboratoire d'Informatique de Paris Nord for a cherished time spent together in the lab, and in social settings.

I would be remiss to not mention my family, especially I am so grateful to my husband, Mr. Ruchira Jayathunga, for all the sacrifices he made in keeping my spirits and motivation high during this process. Without his support this would have only been a dream. I would also like to thank my parents for believing in me and helping me which were crucial in making me what I am today. Moreover, I would like to extend my gratitude to my parents in law, our brothers and our friends for the immense support they have provided throughout this journey. I would also like to thank my three dogs for all the entertainment and emotional support. Furthermore, I cannot forget my grandmother who took care of me in my childhood. I am grateful to her as well.

Moreover, I would like to thank all the teachers and lecturers who have guided me up to this day, in shaping me for what I am today. I would like to thankfully mention Sacred Heart Convent pre-school, Southlands College, University of Moratuwa and University Sorbonne Paris Nord for being the pillars in providing me all the opportunities to become what I am today. Last but not least I would like to mention my sincere gratitude to all the beautiful people in Sri Lanka and France with whom tax money my education was financed since primary school.

Abstract

The emergence of State of the art (SOTA) deep learning models have paved the way for massive improvements in the field of Natural Language Processing (NLP). Especially, when it comes to Natural Language Generation, latest models such as GPT-4 are capable enough to produce text that looks as if they are written by a human. This is beneficial for humans in composing text. But on the other hand, such technologies can be leveraged for malicious tasks. Principally when focused on the academic domain, these tools can be involved in article generation, plagiarism etc. Thus it is vital to distinguish automatically generated text from human written content. Detection of machine generated text has won the attention of many researchers. But regarding the detection of automatically generated academic content, only a few researches are concentrated on the task. Thus there is a demanding requirement in building models/ technologies that supports in detecting automatically generated academic text.

In this thesis, we have focused our interest on identifying technologies /methodologies in detecting artificially generated academic content. The principal contributions of this thesis are threefold. First, we built several corpora that are composed of machine generated academic text. In this task we utilized several latest NLG models for the generation task. These corpora contain contents that are fully generated as well as contents that are composed in a hybrid manner (with human intervention). Then, we employed several statistical as well as deep learning models for the detection of generated contents from original (human written) content. In this scenario, we considered detection as a binary classification task. Thus several SOTA classification models were employed. The models were improved or modified using ensembling techniques to gain higher accuracies in detection. Moreover, we made use of several latest detection tools to identify their capability in distinguishing machine generated text. Finally, the generated corpora were tested against knowledge bases to find any mismatches that could help to improve the detection task. The results of this thesis underline the importance of mimicking human behavior in leveraging the generation models as well of using realistic and challenging corpora in future research aimed at detecting artificially generated text. Finally, we would like to

highlight the fact that no matter how advanced the technology is, it is always crucial to concentrate on the ethical aspect of making use of such technology.

Keywords : Natural Language Generation, Automatically Generated text, Detection, Classification, Knowledge bases

Résumé

L'émergence de modèles d'apprentissage profond (State of the art, SOTA) a ouvert la voie à des améliorations massives dans le domaine du traitement du langage naturel (NLP). En particulier, en ce qui concerne la génération de langage naturel, les modèles les plus récents, tels que GPT-4, sont capables de produire des textes qui ont l'air d'avoir été écrits par un être humain. C'est un avantage pour les humains lorsqu'ils composent des textes. Mais d'un autre côté, ces technologies peuvent être utilisées à des fins malveillantes. Principalement dans le domaine universitaire, ces outils peuvent être utilisés pour la création d'articles, le plagiat, etc. Il est donc essentiel de distinguer le texte généré automatiquement du contenu écrit par l'homme. La détection des textes générés par des machines a attiré l'attention de nombreux chercheurs. Cependant, seules quelques recherches se sont concentrées sur la détection du contenu académique généré automatiquement. Il est donc nécessaire d'élaborer des modèles/technologies qui permettent de détecter les textes académiques générés automatiquement.

Dans cette thèse, nous nous sommes intéressés à l'identification des technologies/méthodologies permettant de détecter les contenus académiques générés artificiellement. Les principales contributions de cette thèse sont triples. Premièrement, nous avons construit plusieurs corpus composés de textes académiques générés par des machines. Dans cette tâche, nous avons utilisé plusieurs modèles NLG récents pour la tâche de génération. Ces corpus contiennent des contenus entièrement générés ainsi que des contenus composés de manière hybride (avec intervention humaine). Ensuite, nous avons utilisé plusieurs modèles statistiques ainsi que des modèles d'apprentissage profond pour la détection des contenus générés à partir du contenu original (écrit par l'homme). Dans ce scénario, nous avons considéré la détection comme une tâche de classification binaire. Plusieurs modèles de classification SOTA ont donc été utilisés. Les modèles ont été améliorés ou modifiés à l'aide de techniques d'assemblage afin d'obtenir une plus grande précision dans la détection. En outre, nous avons utilisé plusieurs outils de détection récents afin d'identifier leur capacité à distinguer les textes générés par des machines. Enfin, les corpus générés ont été testés par rapport à des bases de connaissances afin de

détecter toute inadéquation susceptible d'améliorer la tâche de détection. Les résultats de cette thèse soulignent l'importance d'imiter le comportement humain pour tirer parti des modèles de génération ainsi que d'utiliser des ensembles de données réalistes et difficiles dans les futures recherches visant à détecter les textes générés artificiellement. Enfin, nous aimerions souligner le fait que, quel que soit le degré d'avancement de la technologie, il est toujours crucial de se concentrer sur l'aspect éthique de l'utilisation d'une telle technologie.

Mots clés : Génération de langage naturel, textes générés automatiquement, détection, classification, bases de connaissances

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI . . . . *Artificial Intelligence*

NLP . . *Natural Language Processing*

NLG . . *Natural Language Generation*

ML . . . *Machine Learning*

SOTA . *State-of-the-art*

GPT . . *Generative Pre-training model*

RNN . . *Recurrent Neural Networks*

ANN . . *Artificial Neural Networks*

CNN . . *Convolutional Neural Networks*

LSTM . *Long Short Term Memory Networks*

GRU . . *Gated Recurrent Unit*

NB . . . *Naive Bayes*

PA . . . *Passive Aggressive*

SVM . . *Support Vector Machine*

BERT . *Bidirectional Encoder Representations from Transformers*

RoBERTa *Robustly Optimized BERT-Pretraining Approach*

DeBERTa *Decoding-enhanced BERT with Disentangled Attention*

ELECTRA *Efficiently Learning an Encoder that Classifies Token Replacements Accurately*

RoFT . *Real or Fake Text*

GLTR . *Giant Language model Test Room*

CTRL . *A Conditional Transformer Language Model for Controllable Generation*

ULMFit *Universal Language Model Fine-tuning*

Tf-idf . *Term frequency-inverse document frequency*

Word2Vec *Word to Vector*

BoW . . *Bag of Words*

CPCO . *Consistency of anti preceding sentence using Cosine words Overlapping*

CICO . *Consistency of opposing Input sentences using Cosine words Overlapping*

PCFG . *Probabilistic Context Free Grammar*

LED . . *Longformer Encoder-Decoder*

KPA . . *Key Point Analysis*

BLEU . *Bilingual Evaluation Understudy*

ROUGE *Recall-Oriented Understudy for Gisting Evaluation*

Introduction

Natural Language Generation (NLG) is the process of producing natural language with the help of artificial intelligence. Currently this is widely used for many tasks such as report generation, image captioning, creative writing (eg: poems, stories) and in chatbots. From rule based programs such as ELIZA [Wei83] which was developed in 1960s, NLG has been a popular domain in Natural Language Processing (NLP). The integration of neural networks around 2014 [GB+14] marked a significant advancement in Natural Language Generation(NLG).

Currently with the emergence of sophisticated deep learning techniques such as transformer model [Vas+17], NLG has reached its limitless advancements. Since then, various forms of state-of-the-art (SOTA) transformer models such as the Generative Pre-training model (GPT) [Rad+18], BERT [KT19] and Transformer-XL [Dai+19] have been introduced and utilized for a diverse amount of NLP tasks. To cite some, Natural Language Generation (NLG) [Rad+19], text classification [Yan+19], machine translation [CL19] and text summarization [Lew+20]. The aforementioned research show that the transformer models are capable of producing outstanding results.

One of the most notable breakthroughs in recent years is the evolution of OpenAI's GPT models which are capable of generating text that looks as if they are written by a human. Especially, latest models such as ChatGPT (GPT4)[Ach+23] have won global attention such that they are the first thing that comes to mind of a person who wants to find a solution to even a very general problem such as cooking a meal, solving an equation, writing a novel and the list continues.

Moreover, within academic circles, there has been a discernible trend wherein an increasing number of researchers are embracing these SOTA models. This trend is particularly pronounced in tasks requiring sophisticated language generation, such as article writing. The adoption of GPT models in academia attests to their efficacy in addressing complex linguistic tasks and underscores their role in advancing research capabilities across various disciplines. As these models continue to push the boundaries of what is achievable in natural language understanding and generation, their impact resonates not

only in practical applications but also in shaping the trajectory of scholarly pursuits.

Motivation

Regardless of the valuable contribution provided by these state-of-the-art models for the betterment of NLP in general, some concerns have been raised about the potential risks associated with such models. These models can be misused for malicious tasks such as fake news generation [Zel+19], [VRA18], fake review generation [Ade+20] and viral story generation [Far+17], [WD17].

Recently, some of these models have been applied for the computer-assisted writing of research papers [Wan+19], reviews [Wan+20] or theses [AGM+19]. Despite the advantages in alleviating researchers' workload, a risk for misuse of these technologies exists. Recent works ([CL21], [CLM21] shows that old textual generation models, are being actively used in academic publishing, although their detection is relatively easy since they tend to contains "fingerprints" – words sequences specific to PCFG generators – or nonsense like "tortured phrases" – weirdly paraphrased versions of scientific terms.

Furthermore, ethical considerations arise when individuals fail to disclose their use of artificial models or tools for generating academic text. This lack of transparency introduces a potential ethical dilemma, particularly in the evaluation and review of articles. Unfairness may ensue, as writers who have painstakingly created authentic content may find themselves at a disadvantage when compared to those who leverage automated tools. Disclosure of the use of artificial assistance becomes crucial not only for transparency but also to ensure fairness and equity in the assessment of scholarly contributions.

Therefore, immediate research on detection of academic texts that are artificially generated is imperative. In this way, researchers will also have a tool to determine whether these powerful models were used in a responsible way or not. This serves as the primary motivation behind the focus of this thesis, which centers on the detection of automatically generated academic content. By focusing on detection methodologies, the research aims to contribute not only to the academic community's awareness but also to the establishment of ethical guidelines surrounding the utilization of cutting-edge language models. Through this endeavor, the thesis endeavors to fortify academic practices, ensuring that technological innovations align harmoniously with principles of responsibility, transparency, and scholarly authenticity.

Research Problem

At the commencement of our research journey, a significant hurdle confronted us—the quest for an appropriate dataset housing generated academic content. Given the nascent nature of the domain at that time, a dedicated and readily available corpus proved elusive. Faced with this challenge, our initial focus pivoted towards the creation of a benchmark dataset tailored to our research objectives. Recognizing the absence of a pre-existing corpus suitable for our purposes, we embarked on the foundational step of generating a comprehensive and representative benchmark corpus. This pivotal phase laid the groundwork for our subsequent investigations and underscored the necessity of building foundational resources to propel research in the evolving landscape of generated academic content.

Among the straightforward and widely employed methods, leveraging a pre-trained model to generate content emerged as a common approach. However, upon scrutiny of the samples generated by these pre-trained models, a notable discrepancy became evident—namely, the generated content frequently deviated from the context provided (indicating a misalignment with the seed texts used). In response to this challenge, a strategic decision was made to fine-tune the models despite the inherent time and effort demands associated with this process. This deliberate choice aimed at refining the models to better align with the desired context, ensuring that the generated content would exhibit a closer coherence with the seed texts. The investment of additional time and effort in the fine-tuning process was deemed essential to enhance the contextual fidelity of the generated academic content and meet the specific requirements of our research objectives.

Subsequent to the generation of corpora, a crucial phase in our research involved the evaluation of the content quality and detectability of the generated material. To gauge the intrinsic quality of the generated text, we devised a plan to utilize n-gram-based scoring metrics. These metrics facilitated the measurement of n-gram similarity between the original and the generated text, providing a quantitative assessment of the textual coherence. Simultaneously, the assessment of detectability emerged as a pivotal aspect, aiming to accurately classify a given text as either generated or original. This task presented a notable challenge, particularly when dealing with cases where the generated text closely resembled the original. In such instances, high n-gram similarity indicated good quality, but it posed a challenge for detectability. Striking a balance between these two criteria became imperative, as it allowed us to assess not only the quality of the generated content but also its susceptibility to detection, ensuring a comprehensive evaluation of the effectiveness of our generated corpora.

Outline

Under this section, we present the general context of our work. This thesis mainly aims at improving the detectability of automatically generated academic content, which is important in preserving the authenticity of texts and determining whether these tools can be considered lawful writing support or instead computer-assisted plagiarism. Thus the rest of the manuscript is organized as follows:

— **Chapter 1:** This section provides existing work regarding detection of automatically generated text.

— **Chapter 2:** Methodologies adopted in composing corpora and related statistics are explained under this section.

— **Chapter 3:** This chapter elaborates the methods, models and technologies that were employed in detecting machine generated text and the respective results obtained.

— **Chapter 4:** Under this section the results and evaluations are described.

— **Chapter 5:** This section is dedicated to explain further experiments that were carried out regarding the detection task.

— **Chapter 6:** Finally this chapter concludes this manuscript by providing conclusion remarks and future perspectives of this research.

Chapter 1

Related Work

1.1 Natural Language Generation

1.1.1 Overview

Text generation by models refers to the process in which computer algorithms, often powered by advanced machine learning techniques, generate human-like text based on input data or prompts. These models are trained on vast amounts of textual data and learn patterns, styles, and structures present in the training data. The generation process involves predicting the next word or sequence of words based on the context provided, creating coherent and contextually relevant text.

There are various approaches to text generation, with some models utilizing recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and more recently, transformer models. Notable examples include OpenAI's GPT (Generative Pre-trained Transformer) models and recurrent models like LSTM and GRU (Gated Recurrent Unit).

Text generation finds applications in diverse areas, including natural language interfaces, chatbots, content creation, code generation, language translation, and more. These models have demonstrated the ability to produce human-like text, leading to advancements in various fields and opening up new possibilities for creative and practical applications. However, ethical considerations and challenges, such as ensuring the authenticity and responsible use of generated content, are crucial aspects that researchers and developers continue to address in the evolving landscape of text generation models.

1.1.2 History

The history of natural language generation (NLG) can be traced back to the early days of artificial intelligence (AI) research. The roots of NLG can be found in the 1960s when researchers began exploring the idea of using computers to generate human-like language. Initial efforts focused on rule-based systems, where explicit linguistic rules were programmed to produce text based on input data.

In the 1980s and 1990s, NLG saw advancements with the development of expert systems and knowledge-based approaches. These systems aimed to generate coherent and contextually relevant text by leveraging domain-specific knowledge and rules. Rule-based NLG systems were applied in fields such as weather reporting and financial journalism.

The late 20th century witnessed the emergence of statistical and machine learning approaches to NLG. With the advent of large datasets and improved algorithms, researchers explored the use of statistical models and probabilistic methods for language generation. These approaches marked a shift from rule-based systems to more data-driven and adaptive NLG techniques.

The 21st century brought about significant breakthroughs in NLG with the rise of neural network architectures. Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformer models revolutionized natural language processing tasks, including language generation. OpenAI's GPT (Generative Pre-trained Transformer) models, which was introduced in 2018, demonstrated the capability of pre-trained models to generate coherent and contextually rich text.

Today, NLG continues to evolve with the development of advanced language models, including transformer-based architectures. These models leverage massive amounts of pre-existing text data to generate human-like content, enabling applications such as chatbots, content creation, summarization, and more. The history of NLG reflects a journey from rule-based systems to sophisticated, data-driven models, showcasing the ongoing quest for more accurate, creative, and contextually aware language generation.

1.1.3 Transformer Models on Natural Language Generation

Text generation with Transformer models involves utilizing the architecture's self-attention mechanism to generate coherent and contextually relevant sequences of text. The Transformer model, introduced by [Vas+17], has demonstrated remarkable success in various natural language processing tasks, including text generation.

Figure 1.1 shows the overall encoder-decoder architecture of the Transformer model and the following is a step-by-step explanation of how text generation occurs with it:

1. **Input Representation:** The input to the Transformer model is typically a sequence of tokens representing the context or prompt for text generation. These tokens can be words, subwords, or characters, depending on the tokenization scheme used.
2. **Embedding Layer:** The input tokens are passed through an embedding layer to convert them into continuous vector representations. This layer learns distributed representations of the input tokens, capturing their semantic meanings.
3. **Positional Encoding:** Since Transformer models lack inherent sequential information, positional encoding is added to the embedded vectors to provide information about the position of each token in the input sequence. This helps the model understand the order of the tokens.
4. **Encoder Stack:** The encoded input is then processed through a stack of encoder layers. Each encoder layer in the stack consists of sub-layers: a multi-head self-attention mechanism and a feedforward neural network. The self-attention mechanism allows the model to weigh the importance of different tokens in the input sequence for each position.
5. **Decoder Stack:** For text generation, a decoder stack is employed after the encoder stack. The decoder generates tokens autoregressively, attending to the encoder's output and its own previously generated tokens.
6. **Attention Masking:** During text generation, attention masking is applied to prevent the model from attending to future tokens, ensuring a left-to-right generation process.
7. **Softmax Activation:** The final layer in the decoder outputs a probability distribution over the vocabulary. The softmax activation function is applied to obtain probabilities for each token in the vocabulary.

8. **Sampling or Beam Search:** During generation, a decoding strategy is employed to select the next token. This can involve sampling from the probability distribution or using beam search to find the most probable sequence of tokens.
9. **Repeat:** The process is repeated iteratively to generate the desired length of the text or until a specific stopping criterion is met.

The attention mechanism in the Transformer model is a key innovation that significantly enhances the model's ability to capture long-range dependencies and relationships within sequences of data. Following is an in-detailed description on how attention mechanism plays a pivotal role throughout the transformer architecture,

1. **Self-Attention Mechanism:** The attention mechanism allows the model to weigh the importance of different elements in the input sequence when processing a particular element. In the context of the Transformer, this is referred to as the self-attention mechanism. Given an input sequence, each element (or token) can attend to all other elements, and the attention weights are computed based on their relevance.
2. **Scaled Dot-Product Attention:** The self-attention mechanism in the Transformer uses the scaled dot-product attention mechanism. For each element in the sequence, attention scores are computed by taking the dot product of the query, key, and value vectors. These scores are then scaled to prevent saturation issues.
3. **Multi-Head Attention:** To capture different aspects of relationships, the self-attention mechanism is used in multiple parallel operations, known as attention heads. Each attention head learns different aspects of the relationships within the sequence. The outputs of these heads are concatenated and linearly transformed.
4. **Cross-Attention in Decoder:** In the decoder part of the Transformer, a similar attention mechanism is used, but with an additional cross-attention component. This allows the decoder to attend to the encoder's output, aligning the generated output with the input context.

In summary, the attention mechanism in the Transformer model plays a pivotal role in capturing contextual information, handling long-range dependencies, and improving the model's overall performance in various natural language processing tasks. Its innovative design has contributed to the success and widespread adoption of the Transformer architecture in the field of deep learning.

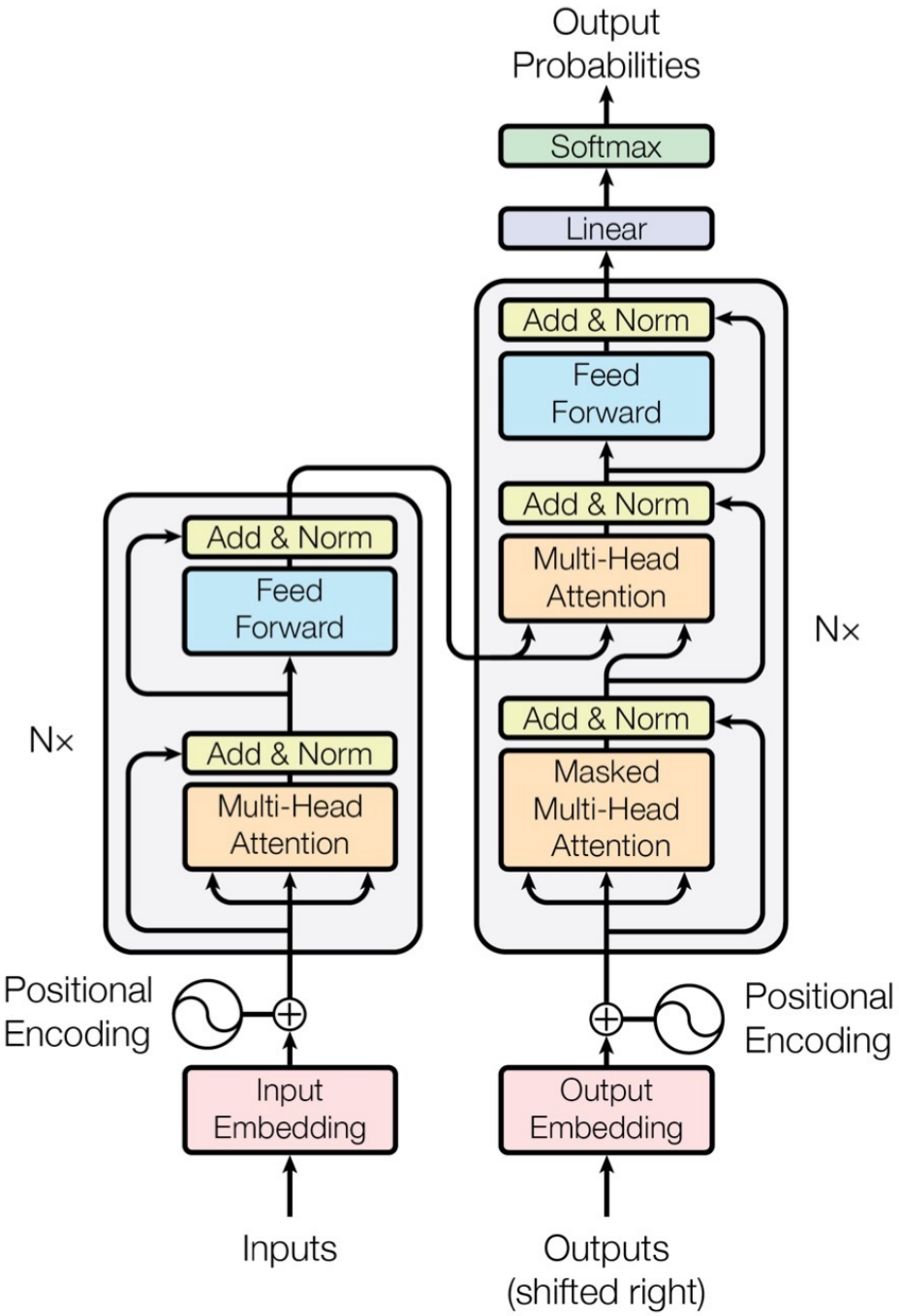

Figure 1.1 – Transformer Architecture (Source: [Vas+17])

1.1.4 Text Generation with GPT models

GPT (Generative Pre-trained Transformer) models stand out for their adeptness in comprehending and generating natural language, a feat that has garnered widespread recognition. Developed by OpenAI, these models harness the potent Transformer architecture, empowering them to analyze and produce text by considering pertinent contextual cues embedded within extensive training datasets. Preceding their deployment, these models undergo rigorous pre-training on expansive collections of textual data sourced from the internet, a process that equips them with a nuanced understanding of linguistic structures and semantics. With the capability for fine-tuning across various tasks or domains, GPT models demonstrate versatility in adapting their knowledge to generate contextually fitting text, thereby serving diverse applications spanning language translation, text completion, and dialogue generation.

In the text generation process, the GPT models break down the text into bite-sized pieces, which are termed as tokens. Each of these tokens are allocated a token ID to identify them. Following Figure 1.2 represents the tokens generated by the GPT tokenizer regarding the example sentence "Text Generation with GPT models". How the model break down language into tokens can vary, but the key thing to grasp is that we end up with special codes (token IDs) for each of these language bits.

Afterwards, these tokens are encoded before them being fed to the language model. The models employ one-hot encoding, a method which converts categorical data, such as words in text or categories in a dataset, into binary vectors. Each category is given a distinct index, and subsequently, it is transformed into a binary vector. In this vector, all elements are set to zero except for the one that corresponds to the index of the category, which is set to one. This approach facilitates the algorithm in comprehending and manipulating categorical data with greater efficiency, as it transforms the data into a format that aligns well with mathematical algorithms.

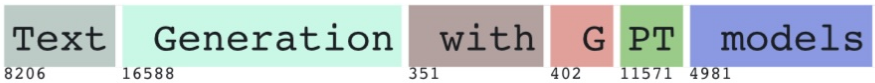

Figure 1.2 – Tokens Produced by GPT Tokenizer for an Example Sentence

Then, each vector possess a length which is equal to the total number of tokens in

the considered language. For example, if it contains 50k tokens, each token has a special vector of length 50k, where only one spot is marked with a 1, and the rest are zeroes. Since every vector has just one non-zero element, this makes it a sparse representation and an inefficient mechanism which wastes a lot of memory. Moreover, it fails to convey the meanings of the words. For instance, if we consider the word "party", it is uncertain whether it refers to a a celebration or a political group.

To eliminate the aforementioned issues, the GPT models utilize embeddings, which provides a compact way to represent tokens with lots of information packed in. In the embedding layer of the models, each token is transformed into a fixed-size continuous vector. For example, in GPT-2, every token gets its own vector made up of 768 numbers. Initially, these numbers are random picked, but during the training phase, the model figures out the best values for these numbers. Figure 1.3 shows an excerpt from the embedding representation produced by the GPT-2 Pre-trained model for the word "Dog".

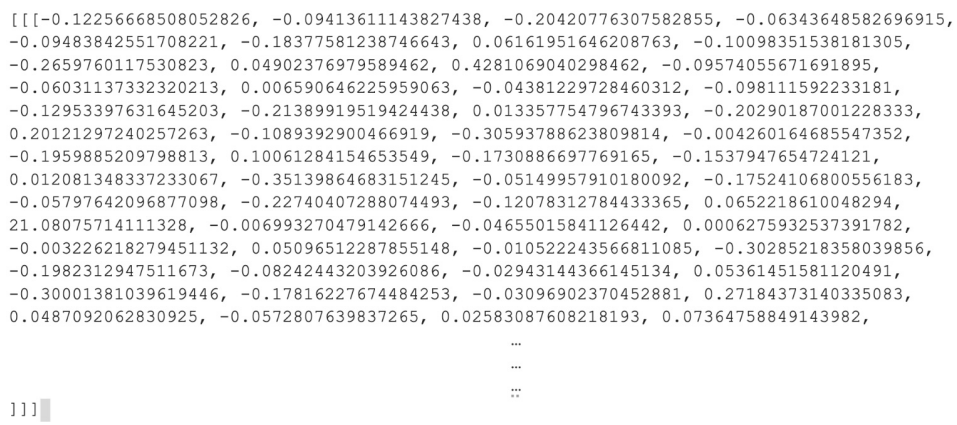

Figure 1.3 – Embeddings Produced by GPT-2 Pre-trained Model for the Word "Dog"

The new vectors are significantly more memory-efficient, with dimensions of 50,000 * 786, a considerable reduction compared to the one-hot encoding's dimensions of 50,000 * 50,000. Consequently, these vectors, known as embedding vectors, serve as inputs for the model. Additionally, if two tokens share similar meanings, their embedding vectors act akin to neighbors, as they tend to be closer to each other in the vector space.

In addition to the token embeddings, the GPT models employ self attention mechanism to intricate complexities of the meanings hidden in human language. Here, each token "pays attention" to every other token in the input sentence, even itself, and works out a set of attention weights which are known as "contextual embeddings". What it essentially

does is assign a fresh set of numbers (attention weights) by calculating the importance of words in the input sentence based on the token embeddings. Ultimately, each word is given a score based on how crucial it is in the sentence.

Following Figure 1.4 is a visualisation which demonstrates the "attention" of the token "cats" to the rest of the tokens in the sentence. The strength of the connection indicates how important or relevant the tokens are.

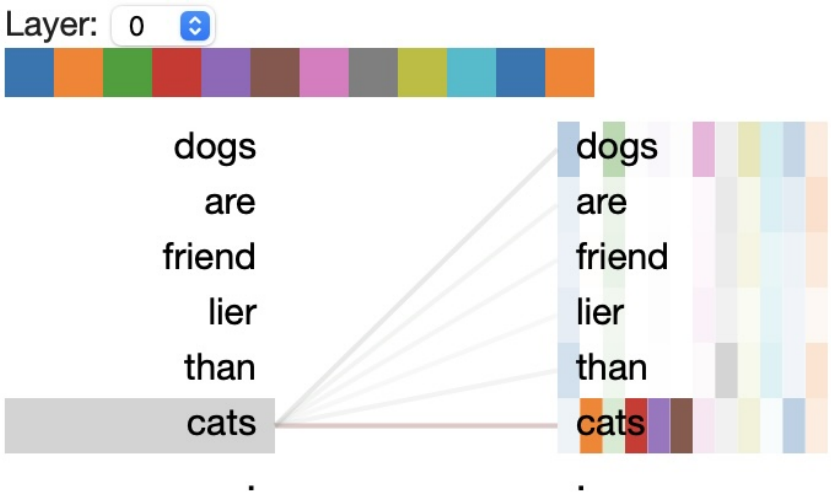

Figure 1.4 – Attention weights of the token (BertViz attention-head view)

In the attention mechanism, we craft a fresh set of weight matrices – specifically, the Query, Key, and Value matrices (simply q, k, v). These matrices, arranged in cascades of the same size (usually smaller than the embedding vectors), join the architecture to capture the intricacies within language units. The attention parameters are learned to unravel the connections between words, pairs of words, and even more complex combinations like pairs of words.

A crucial innovation introduced by transformers involves incorporating positional encodings into vector embeddings. This addition is driven by the need to capture the positional information of words, aiming to improve the accuracy of predicting the next token within the genuine sentence context. This inclusion proves essential because, in many instances, altering the order of words can completely alter the contextual meaning.

All these mathematical techniques we've delved into at a foundational level serve a common purpose: predicting the next token based on the sequence of input tokens. GPT,

fundamentally, undergoes training on a straightforward task—text generation, or more precisely, predicting the next token. Essentially, at its core, we assess the likelihood of a token appearing, considering the sequence of tokens that precede it.

1.2 Ethical Implications and Challenges of Natural Language Generation in AI Systems

Natural Language Generation (NLG) is a swiftly evolving branch of artificial intelligence that allows machines to create text resembling human writing with impressive precision and fluency. This technology is transforming numerous sectors, from producing news articles and automating customer support to crafting personalized content. However, these advancements bring about notable ethical concerns and challenges. This section examines the ethical issues associated with NLG, including bias, misinformation, privacy risks, and intellectual property dilemmas, while also addressing the broader challenges posed by these technologies. By tackling these concerns, we can more effectively manage the integration of NLG into society in a responsible and ethical manner.

Large Language Models (LLMs) have the potential to amplify biases inherent in their training data, resulting in unfair or harmful outputs. As these models are trained on extensive datasets that mirror human language and societal norms, they may unintentionally perpetuate and even exacerbate stereotypes and prejudices. This can lead to biased decision-making, discriminatory content, and the spread of misinformation, presenting considerable ethical and social challenges.

NLG system designers need to exercise great care regarding the data shaping their generated text. Ensuring diversity and representativeness during the initial gathering and preparation of training data is vital. This means capturing a broad spectrum of human experiences and viewpoints. While efforts are underway to explore technical fixes like de-biasing algorithms and better representative data, they're not silver bullets. What's becoming increasingly clear is the necessity for interdisciplinary collaboration, blending technological progress with perspectives from the social sciences to craft algorithms that are ethically robust.

Additionally, transparency in the training and auditing of models is crucial, as is involving affected communities in the development process. Furthermore, transparent communication about the capabilities and limitations of LLMs, including their vulnerability to bias, is essential. Establishing mechanisms for accountability that enable users to evaluate and challenge AI-generated content can also help mitigate bias in LLMs.

LLMs have the capability to generate content that closely resembles human writing. This ability facilitates the swift production of false or misleading information, eroding public trust and fueling social discord. Misinformation refers to false information shared without harmful intent, while disinformation involves false information shared deliber-

ately to deceive. With the aid of LLMs, both misinformation and disinformation can spread widely and rapidly. In today's age, where misinformation proliferates swiftly via social media and other digital avenues, the seamless generation of such content by LLMs amplifies the difficulty in distinguishing fact from fiction. This issue is especially pressing in critical scenarios such as elections, public health emergencies, and financial markets, where the dissemination of false information can yield profound societal repercussions, spanning from the erosion of public confidence to tangible harm.

For example, in academia LLM generated papers could pollute the body of scholarly work, making it difficult for researchers to rely on published materials. Additionally, the generation of fake reviews for products or services can distort consumer choices and market dynamics. The stakes are equally high for democratic processes. Democracy depends on an informed electorate making decisions grounded in accurate information. The dissemination of misinformation and disinformation through LLMs undermines this foundation, potentially swaying election results, referendums, and public policies based on falsehoods.

To mitigate these potential risks, various tools and models are currently employed to distinguish AI-generated content from human-written content. However, given the rapidly advancing capabilities of LLMs, these tools alone are not sufficient. Regulatory frameworks could also help by mandating that machine-generated content be clearly labeled to inform readers of its origin. Additionally, ethical guidelines for the responsible use of LLMs, especially in high-risk areas prone to misinformation, could provide an extra layer of protection.

The use of LLMs brings significant privacy concerns to the forefront. These models are trained on extensive datasets that often contain personal information, sometimes without the knowledge or consent of those involved. Consequently, LLMs can unintentionally generate text that discloses sensitive or private details. The risk of data breaches and misuse of generated content further intensifies these privacy issues. Despite efforts to clean and anonymize data, the complexity and scale of these models make it challenging to ensure that no sensitive information is inadvertently included. This concern is especially critical in applications demanding high confidentiality, such as healthcare and legal services.

When users interact with LLMs, privacy concerns become more pronounced. These interactions often involve the collection and processing of user inputs, which can include personal or sensitive information. The main risks revolve around how this data is stored, used, and potentially incorporated into future training cycles.

To address these concerns effectively, it's crucial to implement robust data protection measures and adhere to privacy regulations. Specifically, organizations developing these

models should follow industry best practices for safeguarding data. Additionally, it's important to provide users with education about the potential privacy risks linked to engaging with these models.

The black box nature of LLMs tends to make these models less transparent and more complex in terms of understanding the logic behind their operations. In contexts like healthcare, where accountability is paramount, the inability to justify decisions made by an LLM can carry significant consequences. The opacity of these models further complicates efforts to pinpoint and rectify biases or errors, potentially perpetuating unfair or harmful behaviors. Developing techniques such as model interpretability and explainability can help illuminate the decision-making processes of these intricate models.

Moreover, it is of paramount importance to understand how the usage of LLMs can affect human creativity and logical thinking power. Currently, many individuals choose to depend on LLMs to generate content, be it an article, an essay, a quote or even an aesthetic product such as a song or or a verse. Although this seems to be efficient in terms of time, the creativity and thinkability of individuals can be hugely affected. Furthermore, the adoption of LLMs carries profound economic ramifications, especially within labor markets. Industries like customer service and journalism that heavily reliant on language-based tasks are particularly susceptible to this transition. These sectors might shift to significant automation through the implementation of chatbots powered by LLMs, potentially displacing human labour. These shifts not only have economic consequences but also bear social implications. Job displacement and economic uncertainty can fuel societal challenges, including mental health issues and heightened economic disparity.

Determining ownership of LLM-generated content has become a complex endeavor. It's not clear-cut to ascertain the authors of such content—whether it's the language model itself, the users interacting with the model, or the developers who created it. Consequently, attributing copyright to the author is ambiguous, raising significant challenges in identifying accountability for the generated content. For instance, in crucial fields like healthcare and law, if decisions rely on machine-generated outputs, it remains unclear who bears responsibility for the outcomes stemming from those decisions.

The creation and implementation of LLMs require significant computational resources and expertise, often making them expensive and inaccessible to individuals or smaller organizations. Access to advanced LLMs may be limited to well-resourced entities, widening inequalities across sectors like healthcare, education, and commerce. Geographic and demographic disparities also affect access, with technologically advanced countries and

dominant languages benefiting disproportionately. Additionally, LLMs may offer limited utility for minority languages or cultures, exacerbating marginalization.

To address inequality in accessing LLMs, one approach is to open-source and offer pre-trained models and tools that demand fewer resources for implementation. Additionally, educational initiatives targeting underrepresented regions or disadvantaged groups to enhance expertise in machine learning and LLMs could help bridge the gap. Furthermore, deliberate efforts to incorporate a wider range of languages and cultural contexts in the training data can enhance the universality of these models.

In conclusion, the vast potential of NLG comes with significant ethical complexities that demand a comprehensive approach. Collaboration among stakeholders, robust regulatory frameworks, and continual research and development efforts are essential to address these challenges effectively. By placing a premium on ethical considerations, we can leverage the benefits of NLG while proactively managing its risks, thus facilitating a more inclusive and ethically sound integration into society.

1.3 Detection of Automatically Generated Text

Detection of Automatically Generated Text is a critical field of research with increasing relevance in our technologically advanced era. As the capabilities of language models, particularly generative models like GPT, continue to advance, the need for robust mechanisms to distinguish between machine-generated and human-written text becomes imperative.

The timely importance of this research stems from the widespread use of language models in various domains, from content creation and journalism to customer service and academic writing. As these models become more sophisticated, there is a growing concern about the potential misuse of AI-generated content, ranging from misinformation and propaganda to unethical practices such as ghostwriting or automated essay generation.

Detecting automatically generated text holds substantial benefits for society. One of the primary advantages is in combating the spread of misinformation. By accurately identifying content created by AI models, we can better assess the reliability of information circulating online and in various media. This is crucial in an age where fake news and manipulated content pose significant challenges to the integrity of information dissemination.

Moreover, detection of automatically generated text is essential for upholding academic and journalistic standards. Ensuring that scholarly articles, research papers, news articles, and other written content are genuinely human-authored is vital for maintaining the credibility and authenticity of these sources. Academic institutions, publishers, and media outlets can benefit from automated tools that help verify the authenticity of the content they publish.

In the realm of content creation, detecting automatically generated text is pivotal for preserving the uniqueness and originality of human-authored works. Plagiarism, intentional or unintentional, can be mitigated by employing robust detection mechanisms, fostering an environment that values creativity and individual expression.

The societal implications of accurate text detection extend to legal and ethical considerations. As AI-generated content becomes more prevalent, establishing norms and regulations around its use becomes imperative. Detection tools can assist in enforcing ethical guidelines, ensuring transparency, and holding individuals or organizations accountable for their use of AI-generated content.

In conclusion, the Detection of Automatically Generated Text is not merely a technological challenge but a societal necessity. It plays a crucial role in safeguarding the integrity of information, upholding academic and journalistic standards, and navigating the ethical landscape of AI-generated content. By advancing research in this domain, we contribute to the responsible and ethical deployment of language models, promoting a future where technology serves humanity with transparency and accountability.

1.3.1 Existing Research on Detection of Automatically Generated Text

The prevailing research efforts dedicated to the detection of machine-generated text can be systematically classified into two distinct categories, each addressing the challenge from a unique perspective. These categories are elucidated in the subsequent sub-sections, providing a comprehensive delineation of the methodologies employed in human-based detection and automatic detection.

— **Human-Based Detection:** In the realm of human-based detection, the focus of research lies in leveraging human evaluators or annotators to discern and classify text as either machine-generated or human-written. This approach involves subjective assessments and qualitative judgments from individuals trained to identify linguistic nuances and patterns indicative of automated text generation. Human-based detection brings a qualitative dimension to the evaluation process, incorporating human intuition and contextual understanding. However, it also introduces potential biases and scalability challenges associated with manual assessments.

— **Automatic Detection:** In contrast, automatic detection methodologies aim to develop computational models and algorithms capable of autonomously distinguishing between machine-generated and human-written text. This approach involves the application of various machine learning and natural language processing techniques to extract features, patterns, and characteristics indicative of the text's origin. Automatic detection systems often rely on statistical analyses, linguistic features, or neural network architectures to make predictions with a higher degree of scalability. While automatic detection methods offer efficiency and scalability advantages, they also present challenges related to the evolving nature of language models and the need for diverse and representative training datasets.

This dichotomy in research categorization underscores the multifaceted nature of the machine-generated text detection challenge. The nuanced exploration of both human-based and automatic detection approaches provides a holistic view of the ongoing efforts to address this critical aspect of natural language processing.

Human Detection of Machine Generated Text

Numerous research endeavors have delved into the domain of human-based detection of machine-generated text, exploring diverse perspectives and methodologies. Bakhtin et al. [Bak+19] approach human detection as a ranking task, scrutinizing the impact of factors like sampling method and text excerpt length. Ippolito et al. [Ipp+19] delve into the efficacy of human detectors, particularly in identifying semantic errors in machine-generated text. Gehrmann et al. [GSR19] assert that human detection accuracy, without any auxiliary tools, stands at a modest 54%. [ÇB20] propose an innovative methodology automating the human evaluation of machine-generated news.

In the exploration of human-centric evaluation, Ippolito et al. [Ipp+20] highlight the human inclination toward semantic errors, contrasting this with discriminative models like fine-tuned BERT, which emphasize statistical artifacts. The introduction of RoFT (Real or Fake Text) by [Dug+20] unveils the capability of text generation models to deceive humans, demonstrating instances where a few sentences can mislead evaluators. Recent findings by [Cla+21] indicate that training humans on GPT-3 generated text evaluation tasks only marginally improves overall accuracy, reaching up to 50%.

Despite the extensive exploration of human detection capabilities, the landscape is marked by a noticeable gap in research focusing on the development of automated tools designed to discern machine-generated text from its human-written counterpart. This underscores the need for more concerted efforts in automating the detection process and advancing the capabilities of tools in this critical domain.

Automatic Detection of Machine Generated Text

The introduction of the statistical model GLTR (Giant Language model Test Room) [GSR19] can be considered as a major milestone for the detection of automatically generated text. The authors consider the stylometric details of texts by incorporating three types of tests: the probability of the word, the absolute rank of a word and the entropy of the predicted distribution. Afterwards, they compute per-token likelihoods and visualize histograms over them to support humans in detection of automatically generated content. A recent research [AL20] has extended the work done in GLTR by feeding the output of GLTR to a Convolutional Neural Network, which automatically classifies whether the input reviews are human written or machine generated.

Another turning point is the establishment of the GROVER [Zel+19], in which its architecture is a combination of a generation model and detection model. News are generated using a transformer based model which has an architecture similar to GPT-2 [Rad+19]. But [Rad+19] has used conditional generation (on article metadata) and nucleus sampling. Afterwards, a zero shot detection is performed using a simple linear classifier on top of the pre-trained GROVER model. The authors have experimented detection with existing other models as well (fastText [Boj+17] and BERT[KT19]) reporting the highest accuracy for their own GROVER model and claiming that the best models in forming fake content are also the best models in detection. Their results show that the highest accuracy is reported by their GROVER model itself. Therefore they have claimed that best models in forming fake content are the best models in detection as well. Contrary to that, Uchendu [Uch+20] shows, however, that GROVER cannot correctly detect texts generated by language models other than GROVER itself.

Lots of research has leveraged the RoBERTa system [Liu+19], a masked and non-generative language model to detect automatically generated text. [Sol+19] has proved that the discriminative model of RoBERTa outperforms generative models such as GPT-2 in detection tasks. Such findings contradict with GROVER authors' claim that the generative model is better in detecting text generated by itself. [Fag+21] reveals that the RoBERTa can defeat traditional machine learning models, such as bag of words, and neural network models, such as RNNs and CNNs, regarding the detection of automatically generated tweets. Moreover, [Uch+20] shows that RoBERTa outperforms existing detectors in detecting automatically generated news articles and product reviews which are generated by state of the art models like GPT-2. Despite the success of RoBERTa, recent research [JML20] shows that its dependence on large amounts of data limits its

use for detection. [Wol20] challenges the RoBERTa model by exposing it with homoglyph and misspelling attacks and their results show a drastic drop in recall. This shows that the future detection models should be robust against such attacks. To test and evaluate such models, it is of key importance to have a dataset that incorporates such attacks or traps, if possible independent of any specific detection method.

[Sch+19] has identified the weaknesses in provenance based detection methods (*e.g.* [Zel+19]) for fake news detection highlighting the importance of veracity in addition to style and source as well as the importance of fact-verification models. In order to mitigate the limitations in provenance based models, [Zho+20] presents a neural graph based reasoning approach (FAST) which considers the factual structures of documents. They believe with their approach the limitations in coarse grained models are eliminated. The approach is experimented on two datasets, a news-style dataset and a webtext-style dataset, with better results than the RoBERTa. The fake data for their research is generated by GPT-2 and GROVER. Moreover, [Sch+20] highlights the limitations of stylometric approaches in detection of machine generated fake news since they highly depend on distributional features. As a remedy for this, [JML20] advocates the use in detection models of external resources such as knowledge bases and diffusion networks in addition to the source itself.

Many of the research on detection assumes that the generator is known ([GSR19], [Zel+19]). Therefore their approaches are incapable in generalizing the settings so that it works well when the generator architectures and corpora are different in training and testing stages. However, in real world settings, a detection model faces indeterminate and evolving data. This issue has been addressed to a certain level by [Bak+19], which provides a thorough experimental setup and quantitative results. [Ipp+20] fine-tunes the BERT model for detection and analyzes how factors like sampling strategies and text excerpt length impact the detection task. Another research [VKS20] produces a formal hypothesis testing framework and sets error exponents limits (in terms of perplexity and cross-entropy) for large scale models, such as GPT-2 and CTRL, in order to find limits in detecting text generated by them. One of the latest research [MSS21] leverages transformers to detect headlines generated by GPT-2 model. In this approach, the models learn from the datasets and classify text as machine generated text or human-written text. It makes use of 4 types of classifiers: Baselines (Logistic Regression, Elastic Net), Deep learning (CNN, Bi-LSTM, Bi-LSTM with Attention), Transfer learning via ULMFit and Transformers (BERT, DistilBERT) for the detection task. The results show that BERT scores an overall accuracy of 85.7%.

In addition to transformer based models many research works conducted for the detection make use of other types of deep learning models as well as statistical models. [Vij+20] experiments with numeric representation such as Tf-idf and word2vec, as well as neural networks such as ANNs and LSTMs on detection of fake news. A latest research [HBC21] suggests two approaches – CPCO (Consistency of anti preceding sentence using Cosine words Overlapping), and CICO (Consistency of opposing Input sentences using Cosine words Overlapping) –, which utilize sentence coherence for the detection task. Also [Jaw] leverages different discourse models for detection. By exposing style-based classifiers to syntactic and semantic permutations, [BP20] shows the limitations of style-based classifiers which highly rely on provenance to detect fake text. Furthermore, [Pér+18] highlights the importance of linguistic features such as semantic, syntactic and lexical features in distinguishing the machine generated news from human written news.

1.3.2 Existing Research on Detection of Automatically Generated Academic Content

Although there is lots of research conducted for the detection of automatically generated content such as newspaper articles, reviews, tweets, headlines and so on, only a limited number of detection research is dedicated to academia. These works [XH09; LK10] have mainly focused on the authenticity of the references. [Ama15] has examined topological properties in natural and generated papers as a source for detection. [WG15] has proposed various measures in detecting generated academic content.

SciDetect [NL16] is another research which considers inter-textual distance using all the words and nearest neighbour classification for detection. The authors have analyzed the existing methods for detecting automatically generated papers and relied on Probabilistic Context Free Grammar (PCFG) to detect fake academia. However, their approach relies on the fact that, for PCFG based generators, word distributions are dissimilar to that of human written content, an assumption that is less obvious for more recent language models

A recent research [CL21] proposes a rule based detection mechanism which leverages third party search engines to distinguish automatically generated papers based on improbable word sequences found in them, but their approach can only detect grammar based computer generated papers. However, among all the works dealing with the detection of automatically generated text, there is too few research dedicated to the academic or scholarly domain, despite the availability of such data and the potential danger in the misuse of generative models in this domain.

In order to leverage the deep learning models to detect automatically generated research content from human written content, a corpus of artificially generated academic content is required. Many latest research [CL21], [XH09] have leveraged SCIgen to generate a dataset for their research. But SCIgen generates nonsense (because it focuses on amusement rather than coherence) using context free grammar.

DAGPap22 [Kas+22] is a shared task targeted at detecting automatically generated academic text. The competition was aimed at two tasks, 1. To distinguish machine generated content from human written content, and 2. To identify from which models the fake contents were generated. Their dataset consists of original abstracts extracted from "Microprocessors and microsystems (MICPRO)" journal and abstracts copied from papers related to UN's Sustainable Development Goals and fake abstracts generated using summarization and generative models. By utilizing an ensemble of the three models

SciBERT, RoBERTa and DeBERTa, Glazkova and Glazkov., 2022 [GG22] have gained an F1 score of 99.24% on DAGPap22 data. SynSciPass [Ros22] is another latest approach that facilitates the detection of automatically generated scientific content by providing labels for the type of technology adapted for generation. They also used the SciBERT model for detection and obtained an F1 score of 98.3% for DAGPap22 data. Rodriguez et al. [Rod+22] experimented cross-domain applicability of detectors. In their work, they have studied the detectability of tampered (created by a mix of original and generated paragraphs) research papers using BERT-based models and reported accuracies ranging from 86 to 95% across the domains, depending on the configurations.

Although the aforementioned researches have obtained higher results in terms of the detectability of academic text, their considered datasets aren't composed in a manner that a human would possibly employ NLG models in composing a research article.

Chapter 2

Construction of Corpora Composed of Machine Generated Academic Text

To assess the detectability of artificially generated academic text, a prerequisite is a challenging dataset comprising such content. At the initiation of our research, we encountered a significant gap – there was no existing corpus specifically curated for artificially generated academic content. This lack of a suitable dataset prompted us to embark on the creation of a challenging benchmark corpus tailored to address this specific need. Therefore we have composed several corpora (D1, D2, D3, D4 and D5) by adopting various mechanisms to make sure there are substantial amounts of variations among the datasets. Furthermore, throughout the course of my PhD journey, numerous researchers have contributed by publishing several corpora, including D6 and D7, which proved invaluable for our experimental endeavors. In this chapter, we describe the following corpora which we used in our experiments,

1. **D1:** A dataset composed of fully generated articles (using fine-tuned GPT-2 model).
2. **D2:** Second dataset of fully generated articles with an improved level of creativity in the generated content.
3. **D3:** A hybrid dataset composed by a mix of machine generated and human written content.
4. **D4:** A dataset composed of generated abstracts (using pre-trained GPT-2 model)
5. **D5:** A dataset of hotel reviews generated by GPT-2 and GPT-3 models.
6. **D6:** Dataset published by DAGPap22 shared task [Kas+22].
7. **D7:** WikiGPT dataset composed of wikipedia introductions. [Bha23]

The statistics of the considered corpora are provided in Table 2.1. Moreover, some excerpts extracted from academic corpora that we built, are included in Table 2.2. Each of the aforementioned datasets are further explained in the subsections. All the produced data are available at the following URL: `https://github.com/vijini/Detection-of-Automatically-Generated-Academic-Text.git`.

Dataset	Generation model	No.of articles	AWPA(Approx.)
D1: FG full paper (t=0.7)	GPT-2 (fine-tuned)	200	1250
D2: FG full paper (t=0.9)	GPT-2 (fine-tuned)	200	1250
D3: Hybrid Abstract (t=1.0)	GPT-2 (fine-tuned on ArXiv-NLP)	200	150
D4: FG Abstarct (t=0.7)	GPT-2 (pre-trained)	200	125
D5: FG Hotel reviews (t=0.7)	GPT-2 (fine-tuned), GPT-3 (pre-trained)	200	150
D6: DAGPap22	SCIgen	5350	160
D7: WikiGPT	GPT(Curie)	1000	125

Table 2.1 – Statistics of the dataset sections (AWPA: Average Words Per Article, FG: Fully Generated, t=temperature)

Origin of text	Excerpt
Original paper	Moreover, when temporal word embeddings provide words' representations that are more inclined towards their literal core meaning (and not the metaphorical one), models exploiting end up correctly identifying metaphors more easily. **Furthermore, embeddings of words belonging to some language domains in specific time periods can be biased towards their metaphorical meaning, leading to words being used in metaphorical contexts way more than in literal ones. This would prevent neural models from correctly identifying the words as metaphors.**

D1 (fine-tuned GPT-2, temp = 0.7)	Moreover, when temporal word embeddings provide words' representations that are more inclined towards their literal core meaning (and not the metaphorical one), models exploiting end up correctly identifying metaphors more easily. **One way to investigate these hypotheses further is to explore the nearest neighbors of a word in the word embeddings used in a figurative way inside a sentence, both in a static (atemporal) word embedding space, e.g., obtained with GloVe [Jeffrey Pennington, 2014] and in a decade-specific temporal space, e.g., obtained from the CoHa1 corpus with Procrustes [Edouard Grave, 2018].**
D2 (fine-tuned GPT-2, temp = 0.9)	Moreover, when temporal word embeddings provide words' representations that are more inclined towards their literal core meaning (and not the metaphorical one), models exploiting end up correctly identifying metaphors more easily. **One way to investigate these hypotheses further is to explore the nearest neighbors of a word in the word embeddings used in a figurative way inside a sentence, both in a static (atemporal) word embedding, i.e., a decade, time period, and/or a temporal (osfringes) word embedding, i.e., a decade, time period.**
D3 (ArXiV-NLP fine-tuned GPT-2, temp = 0.7)	Moreover, when temporal word embeddings provide words' representations that are more inclined towards their literal core meaning (and not the metaphorical one), models exploiting end up correctly identifying metaphors more easily. **In contrast to sentence embeddings, the method of segmentation also exploits the concept of end-to-end (or equivalence) for metaphors and the corresponding temporal word embeddings. In this work, we conduct a multidisciplinary investigation of a well-known technique for identifying and segmenting metaphors.**

D4 (GPT-2 without fine-tuning, temp = 0.7)	Moreover, when temporal word embeddings provide words' representations that are more inclined towards their literal core meaning (and not the metaphorical one), models exploiting end up correctly identifying metaphors more easily. **For example, if a model is able to identify a metaphor that is more likely to be associated with the literal meaning of a word, then it will be more accurate in identifying the metaphor in the context of the word itself. In this paper, we present a novel approach to the identification of metaphors in language.**

Table 2.2 – Excerpts of a paper and its generated versions from the different dataset sections.

2.1 D1: Dataset of Fully Generated Articles (generated with a temperature parameter of 0.7).

To test methods and encourage research in detection of automatically generated academic content, by the time we started our research, there was a greatest requirement of a dataset composed of artificial academic content. Thus we proposed a benchmark corpus of academic papers, which is explained under this section. Architecture of the overall approach is presented in Figure 2.1.

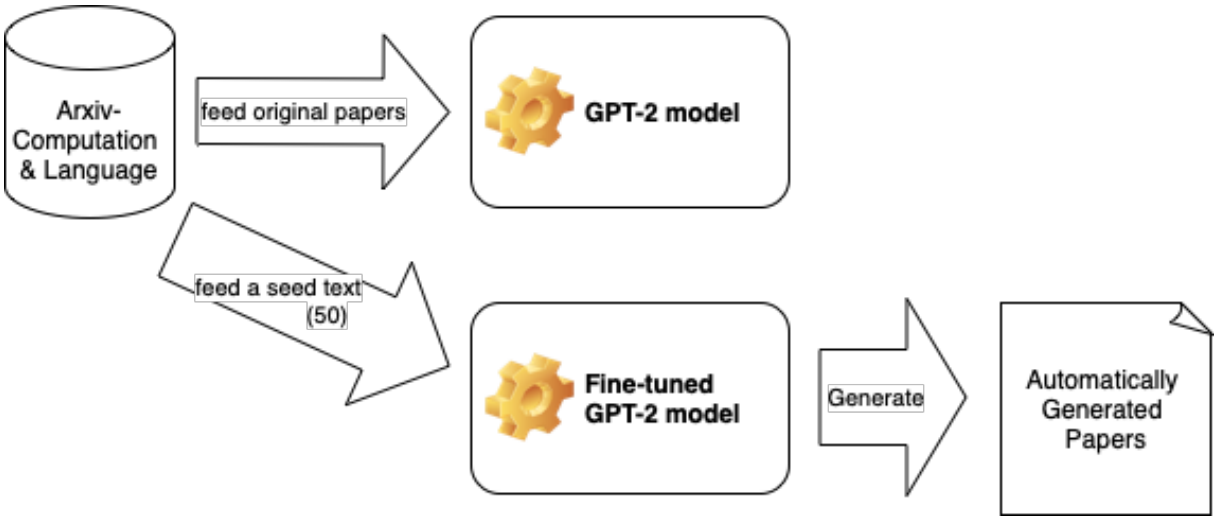

Figure 2.1 – Architecture Followed in Generating Research Articles using Fine-tuned GPT-2

We chose the GPT-2 124M parameter model [Rad+19] for generation since it was by then the most advanced Generative Pre-training Transformer (GPT) model that was available for generation.

Initially, the model was fine-tuned by feeding it with papers extracted from ArXiv [1]. We specifically selected the Computation and Language domain and the 100 latest papers (as of April 2021) were chosen to make sure that the papers we considered were not used to train the original GPT-2 model. Moreover, papers with IEEE citation style were not selected to train the model, because this style represents citations only as numbers, which does not allow a reader to verify whether the citations are appropriate (we do not consider the “References" section for model training). For clarity purposes, we ignored

1. `https://arxiv.org`

the sections such as methodology, results and evaluation and discussion which contain diagrams, tables, equations.

Each paper was separately used to fine-tune the GPT-2 model. Afterwards, by choosing the first 50 words of each original paper as the seed text, a new paper which is of the same length as the original paper was generated using the fine-tuned model. Likewise, 100 novel papers were generated. The temperature parameter was set to 0.7 to make sure that the generated content is neither too random nor too much like the original paper. This process corresponds to the situation in which a malicious author would create a full text to submit, in a similar way to Sci-Gen. Their length is variable as they can be composed from the abstract alone to more sections such as introduction, related work and conclusion.

Few examples showing content from the original paper and the respective generated paper are available in Table 2.3. Here the original papers correspond to the papers which were used as prompts for the models to perform generation. To obtain a proper comparison between the original and generated articles, two n-gram based scores, BLEU and ROUGE were calculated and the respective results are provided in Section 4.

Original Paper	Generated Paper
A possible alternative would be to require each model to set its own minimum confidence threshold to determine if an input argument should not be matched to any KP. Finally, in this task, we used comparative evaluations to **determine** the quality of the sets of generated **KPs. Obviously, this results with rankingbased evaluation. Establishing an absolute evaluation metric in this context merits further investigation. We thank all participating teams for taking the time to participate in this challenging new shared task. We believe that these contributions, along with the data collected and shared in this report, will be valuable to further advance the research around KPA and related tasks.**	A possible alternative would be to require each model to set its own minimum confidence threshold to determine if an input argument should not be matched to any KP. Finally, in this task, we used comparative evaluations to **estimate** the quality of the sets of generated **texts. We expect the task and the findings reported in this paper to be relevant for researchers working on text summarization and argument mining. Introduction Key Point Analysis (KPA) was introduced in BarHaim et al. (2020a,b) as a challenging NLP task with tight relations to Computational Argumentation, Opinion Analysis, and Summarization, and with many practical applications (Bar-Haim et al., 2021).**
We show that the mixed-domain transfer learning approach **does not always work well. We found** the **combination** of the **non-graph DSM (for capturing local relationships) and CNN (for extracting global relationships) to be a promising approach towards creating a gener- alizable solution for detecting COVID-19 online misinformation. In the future, we plan to investigate the gener- alizability of the DSM models created using deep learning architectures such as BERT.**	We show that the mixed-domain transfer learning approach **is more effective across most of** the **dimensions** of the **COVID-19 data. Specifically, we show that the shallow mixed-domain transfer learning approach is more effective across the following dimensions of the COVID-19 data: granular (grained), shallow (skewed), and deep (deeper). We show that the shallow mixed-domain transfer learning approach is more effective across the following dimensions of the COVID-19 data: temporal dimension (the context in the dataset evolves), length dimension.**

Table 2.3 – Some examples of original *vs.* generated papers in the “fully generated" corpus.

2.2 D2: Dataset of Fully Generated Articles (generated with a temperature parameter of 0.9).

Since the seed text which was used in generating the D1 dataset represents an important lead for the generation process, the generated articles contain a lot of content from the respective original paper. Thus to address this issue, we tried to introduce more randomness in the models' output. In language models, one way to introduce more randomness is to increase the temperature τ of the softmax that is used to determine the probabilities of the following token w:

$$\frac{\exp\left(\frac{h^T e_w}{\tau}\right)}{\sum_{w_i \in V} \exp\left(\frac{h^T e_{w_i}}{\tau}\right)} \tag{2.1}$$

where h^T are the outputs of a hidden layer in the LM, and e_w are embeddings of the w token in the vocabulary V. Since increasing the temperature increases the probability of generating more variations in the text with respect to the original, we built a new dataset of fake articles by increasing the temperature of the model to 0.9 instead of the original temperature of 0.7). For this generation process, we followed the same steps applied in composing the D1 dataset. We leveraged BLEU and ROUGE scores to measure the novelty of the generated articles when compared with the original ones and the respective results are provided under Section 4.

2.3 D3: A hybrid dataset composed by a mix of machine generated and human written content.

This corpus is a collection of abstracts which are composed of a combination of original content and machine generated sentences. It corresponds to a situation in which an author would recur to language generation to fill in certain parts of hers paper. To compose this corpus, we ignored papers containing the name of the proposed model, product or project in the proposal statement of the abstract, as the Arxiv-NLP model might suggest a name that would not be consistent with the rest of the abstract (the sentences extracted from the original abstract), thus making the generated abstract too easily distinguishable from human written abstracts. Table 2.4 shows some examples of the original abstracts and their corresponding generated abstracts. For this task we utilized some of the latest abstracts from Artificial Intelligence domain in ArXiv.

Original Abstract	Generated Abstract
Our experiments suggest **that models possess belief-like qualities to only a limited extent, but update methods can both fix incorrect model beliefs and greatly improve their consistency.** Although off-the- shelf optimizers are surprisingly strong belief- updating baselines, our learned optimizers can outperform them in more difficult settings than have been considered in past work.	Our experiments suggest **the importance of model beliefs in learning models, and we show that the approach outperforms automatic model updating systems using word representations.** Although off-the- shelf optimizers are surprisingly strong belief- updating baselines, our learned optimizers can outperform them in more difficult settings than have been considered in past work.

Simultaneously evolving morphologies (bodies) and controllers (brains) of robots can cause a mismatch between the inherited body and brain in the offspring. To mitigate this problem, the addition of an infant learning period by the so-called Triangle of Life framework has been proposed relatively long ago. However, an empirical assessment is still lacking to-date. In this paper we **investigate the effects of such a learning mechanism from different perspectives.**	Simultaneously evolving morphologies (bodies) and controllers (brains) of robots can cause a mismatch between the inherited body and brain in the offspring. To mitigate this problem, the addition of an infant learning period by the so-called Triangle of Life framework has been proposed relatively long ago. However, an empirical assessment is still lacking to-date. In this paper , **we present a method to evaluate the effect of an algorithm based on the development of a hybrid human/bot learning framework, which combines the development of both a hybrid robot and a human model on the same domain.**

Table 2.4 – Some examples of original *vs.* generated abstracts from the “hybrid" corpus.

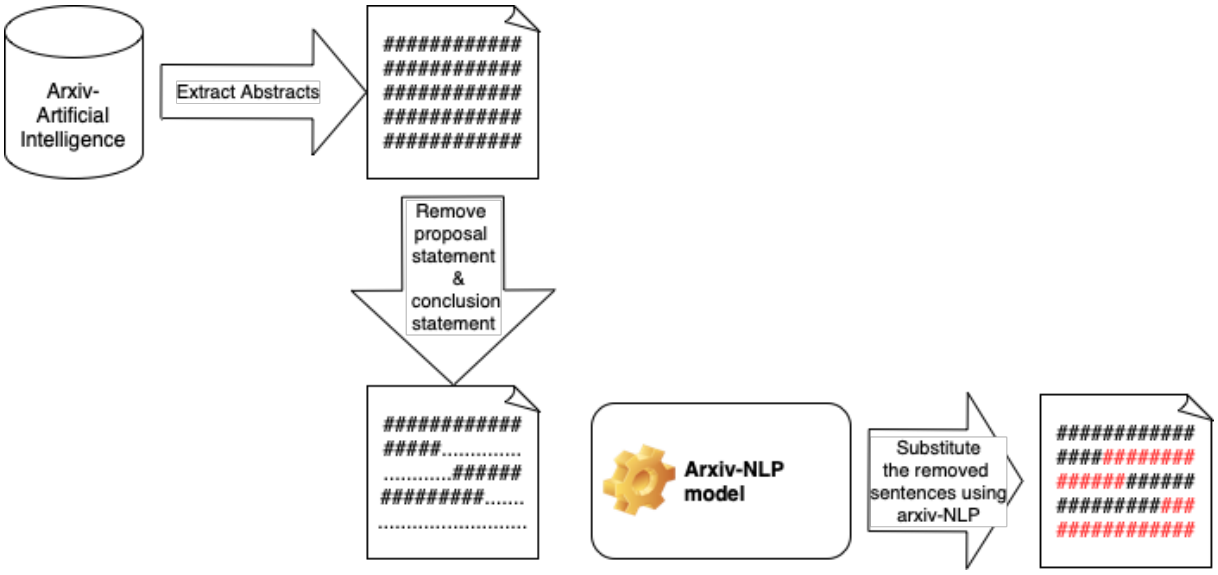

Figure 2.2 – Architecture Followed in Composing Hybrid Dataset

As represented in Figure 2.2, each hybrid abstract is made of 4 parts. The initial content is extracted from an original abstract up to the point where it reveals about the proposal (*e.g.* "In this paper,", "We propose", "Here we"). The next sentence is generated until the first full-stop, using the Arxiv-NLP provided by the Huggingface team [Wol+20a]. Then, the rest of the original abstract is copied until the point that corresponds to the conclusion (*e.g.* "We conclude", "Our results show that"). Again, using the Arxiv-NLP, the rest of the abstract is generated. Likewise 100 new abstracts were composed. The temperature parameter was set to 1 and top-p was 0.9. This generation is done with human intervention, so that it is biased towards the objective strategy of making the generated content difficult to detect.

2.4 D4: A dataset composed of generated abstracts (using pre-trained GPT-2 model)

This dataset of abstracts is built using OpenAI's pre-trained GPT-2 English model without fine-tuning. This dataset corresponds to the way most of the previous works have introduced the generated data in their detection experiments. The GPT-2 model that we used has 24 layers, 1024-hidden states, 16-heads and 345M parameters. The temperature of the model at this generation was set to 0.7. The model has been pre-trained on abstracts from the Arxiv-NLP domain. To prevent the model from repeating the output tokens starting from the beginning, we had to set the early stopping feature of the model to true, which makes the generated abstracts shorter in length than the original abstract lengths. Furthermore, when examining the generated abstracts, we could see that they contain sentences that can be factually incorrect. This is expected since the model is not fine-tuned.

Few examples of this corpus are provided in Table 2.5. As per the first example, we can see that when the model is not fine-tuned, it tends to generate text that can be out of the context (more general in nature). For an example the statement "developing a model that can be used to train a large number of models on a single dataset." is just a general sentence which does not add any information specific to the context, that is focused in the abstract. Thus such generations make it easy to detect them from their respective original contents.

Original Abstract	Generated Abstract
Unsupervised pre-training is now the predominant approach for both text and speech understanding. Self-attention models pre-trained on large amounts of unannotated data have been hugely successful when fine-tuned on downstream tasks from a variety of domains and languages. This paper takes the universality of unsupervised language pre-training one step further, **by unifying speech and text pre-training within a single model. We build a single encoder with the BERT objective on unlabeled text together with the w2v-BERT objective on unlabeled speech.**	Unsupervised pre-training is now the predominant approach for both text and speech understanding. Self-attention models pre-trained on large amounts of unannotated data have been hugely successful when fine-tuned on downstream tasks from a variety of domains and languages. This paper takes the universality of unsupervised language pre-training one step further, **developing a model that can be used to train a large number of models on a single dataset. The model can then be applied to a wide range of real-world tasks, including speech recognition, natural language processing, and machine translation.**
This paper presents an approach to measuring business sentiment based on textual data. Business sentiment has been measured by traditional surveys, which are costly and time-consuming to conduct. To address the issues, we take advantage of daily newspaper articles and adopt a self-attention-based model to define a business sentiment index, **named S-APIR, where outlier detection models are investigated to properly handle various genres of news articles. Moreover, we propose a simple approach to temporally analyzing how much any given event contributed to the predicted business sentiment index.**	This paper presents an approach to measuring business sentiment based on textual data. Business sentiment has been measured by traditional surveys, which are costly and time-consuming to conduct. To address the issues, we take advantage of daily newspaper articles and adopt a self-attention-based model to define a business sentiment index, **i.e. a measure of how people perceive the state of the economy. We find that the index is highly correlated with the stock market, with a correlation coefficient (r) of 0.78.**

Table 2.5 – Some examples of original *vs.* generated abstracts from the “hybrid" corpus.

2.5 D5: A dataset of hotel reviews generated by GPT-2 and GPT-3 models.

While our primary focus was on detecting artificially generated academic content, we decided to diversify our approach by creating a corpus of hotel reviews. The objective was to explore and evaluate the challenges posed by detection of generated academic content in contrast to detection of other types of generated content.

This corpus consists of AI-generated hotel reviews of 20 Chicago hotels. The original reviews were extracted from the famous Myle Ott's opinion spam dataset [Ott+11] [OCH13]. Despite the fact that our research is aimed at detecting academic content, we created this dataset since it is easier to build a model of the hotel as a knowledge graph (e.g; it has a swimming pool, gym) from the true ones and compare with the fake ones, than to do that with academic content. Thus it was our step one in the process of making use of knowledge bases for detection.

For this task, we leveraged the 124M-parameter GPT-2 and GPT-3 models. GPT-2 was finetuned with the original reviews. Then, following the same methodology that we used in generating D1 corpus, 400 new reviews (20 for each hotel) were generated. GPT-3 was not fine-tuned, instead the pre-trained model was used for generation. In this task, the first 10 words of the original reviews from the Myle Ott Opinion Spam corpus were used as the seed text. And the rest is generated using the GPT-3 model, to complete for a length comparable to the length of the original review. Likewise 20 reviews were generated for each hotel making a total of 400 reviews. The original and generated hotel reviews are available at [2].

Furthermore, adopting the counter-detection strategy as outlined by [Sad+23], we constructed a final dataset using paraphrasing techniques. Specifically, a T5 model fine-tuned on paraphrases generated by chatGPT, accessible on Hugging Face [3], served as the chosen paraphrasing model. For each original review, we generated five distinct paraphrases and randomly selected one from the set. Employing this approach, we applied paraphrasing to the reviews generated by the GPT-3 model, resulting in an additional set comprising 800 fake reviews. This method allowed for a diverse and varied collection of paraphrased content to enhance the robustness of the dataset for experimentation and analysis.

2. https://github.com/dbuscaldi/op_spam_GPT3
3. https://huggingface.co/humarin/chatgpt_paraphraser_on_T5_base

Below are examples of negative reviews:

— **Original review:** This hotel must have originally been an ordinary downtown hotel in a not very nice neighborhood. The owners have painted it over and fixed up the lobby to make it an "upscale" overpriced boutique hotel. But it's still just an ordinary hotel with a paint job. Stay here if you want to experience staying in a flophouse, but with all the sleaze removed. Definitely stay at the chain's other location, the Burnham, for a much, much nicer hotel.

— **Fine-tuned GPT-2 generated review:** We stayed in hotel Allegro and it was bad. Although the rooms are decent, they are not called "The Best in a Hilliard" and are not suited with insole. The bathroom was very small and has anearth and mineral salts. The pipes and sinks were old and the shower and tub are in need of a major overhaul. The hotel charges for internet and/or rental for 5 internet sessions a day. If you sign up for wireless internet, they do not fund wireless internet so slow and un-favorable wireless speeds.

— **Pre-trained GPT-3 generated review:** This hotel must have originally been an ordinary downtown hotel but now it is hardly more than a shabby motel. The walls are dirty and the furniture is worn, with some broken pieces. My room smelled musty and the carpets had stains. The bed was uncomfortable and the linen was old and threadbare. The staff were not particularly helpful and I felt like I was an inconvenience to them.

— **Paraphrased GPT-3 review:** I'm sure the hotel was once a regular downtown hangout, but it's now resembles ruins. The walls are unclean, the furniture is old and scratchy, there are stains in every room, an uncomfortable bed, and lifeless linens.

2.6 D6: Dataset published by DAGPap22 shared task.

DAGPap22 [Kas+22]'s original data is comprised of excerpts extracted from "Microprocessors and microsystems (MICPRO)" journal and abstracts copied from papers related to UN's Sustainable Development Goals [4]. Its fake (generated) content are composed of abstracts generated by GPT, GPT-neo and GPT3 models (The initial sentence of each original abstract is chosen as the prompt), summarized abstracts produced by Longformer Encoder-Decoder (LED) text summarization model (aforementioned original papers were used as the inputs), abstracts paraphrased with Spinbot [5] and excerpts that are taken from retracted papers of the MICPRO journal. Altogether their training set contains 5327 records in which around 69% is fake. Few examples of the dataset are provided in Table 2.6.

Origin of excerpt	**Excerpt**
Original excerpt	In this paper we propose a low-error approximation of the sigmoid function and hyperbolic tangent, which are mainly used to activate the artificial neuron, based on the piecewise linear method. Here, the hyperbolic tangent is alternatively approximated by exploiting its mathematical relationship with the sigmoid function, showing better results. Special attention has been paid to study the minimum number of precision bits to achieve the convergence of a multi-layer perceptron network in finite arithmetic machine. All the approximation results show lower mean relative and absolute error than those reported in the state-of-the-art. Finally, the sigmoid digital implementation is discussed and assessed in terms of work frequency, complexity and error in comparison with the state-of-the-art.

4. https://sdgs.un.org/2030agenda
5. https://spinbot.com

Fake excerpt (Summarized)	In this paper, the aim of the study is to determine the effect of transplanting islets from the stomach into the gastroesophageal space via a gastric gastroscopical gastroscopy and then performing an endoscopic surgical procedure to extract an islet-derived tissue from the gastric mucous membrane. The primary goal of this paper is to explore the effects of this surgical procedure on the sensitivity of the islet tissue to inflammatory drugs and to determine if the procedure can be safely performed without incurring an adverse effect on the patient's blood-clotting immune system by inducing an inflammatory response.

Table 2.6 – Excerpts Extracted from DAGPap22 Dataset

2.7 D7: WikiGPT dataset composed of wikipedia introductions.

“GPT wiki intro" is a dataset introduced by [Bha23], which is composed of human written Wikipedia introductions and GPT(Curie) generated introductions. For the generation, the first seven words of the original introduction are fed to the model as the seed text. In this task, they have considered 150k different topics from various domains (including academia). For our experiments, we extracted 500 original introductions and their respective 500 generated introductions from the original dataset. Table 2.7 provides several examples of this corpus.

Origin of introduction	**Introduction**
Original wikipedia introduction	In mathematics, specifically differential calculus, the inverse function theorem gives a sufficient condition for a function to be invertible in a neighborhood of a point in its domain: namely, that its derivative is continuous and non-zero at the point. The theorem also gives a formula for the derivative of the inverse function. In multivariable calculus, this theorem can be generalized to any continuously differentiable, vector-valued function whose Jacobian determinant is nonzero at a point in its domain, giving a formula for the Jacobian matrix of the inverse. There are also versions of the inverse function theorem for complex holomorphic functions, for differentiable maps between manifolds, for differentiable functions between Banach spaces, and so forth. Statement For functions of a single variable, the theorem states that if is a continuously differentiable function with nonzero derivative at the point ; then is invertible in a neighborhood of , the inverse is continuously differentiable, and the derivative of the inverse function at is the reciprocal of the derivative of at :

Generated wikipedia introduction	In mathematics, specifically differential calculus, the inverse function theorem states that for every real-valued function there exists an inverse function that satisfies the following two conditions: 1. The inverse function is continuous at every point where the original function is continuous. 2. The inverse function is unique up to a constant multiple of the given function's derivative at any given point.

Table 2.7 – Excerpts Extracted from WikiGPT Dataset

Chapter 3

Methodology Adopted in Detecting Artificial Text

This chapter is dedicated to the main contribution of our research, the detection of automatically generated content. We considered detection as a binary classification task. In our work, several SOTA classification models as well as novel detection tools were leveraged for the detection task. Moreover, the attention of classification models was examined to understand their performance levels. Thus the rest of this chapter is organized as follows,

1. Evaluation of the quality of the artificially generated text.

2. Detection as a binary classification task.
   (a) Statistical models and their ensembles employed in classification.
   (b) Recurrent network models and their ensembles employed in classification.
   (c) Transformer architectures and their ensembles employed in classification.

3. Leveraging detection tools to distinguish machine generated content from human written content.

3.1 Evaluation of the Quality of the Artificially Generated Text.

We present a double approach to evaluate the utility of the produced corpora for the task of classifying artificially generated and human written academic texts in a context where neural-based generation models have become common. We first evaluate the intrinsic quality of the generated texts, assuming that the more natural they seem the more difficult and the more misleading they are for the detection methods. We also perform an application-based evaluation using a panel of state-of-art detection models to assess the difficulty of the classification task and to check that our benchmark is not biased towards a specific detection approach.

To assess the intrinsic quality of our benchmark, we leveraged two measures: BLEU (Bilingual Evaluation Understudy) [Pap+02] and ROUGE (Recall-Oriented Understudy for Gisting Evaluation) [Lin04] to compare the generated contents with their respective original contents. BLEU and ROUGE are two traditional measures used to compare the candidate (generated) text with the reference (original) text. They are traditionally considered as fluency measures that indicate how natural and artificial text is compared to a natural original one. BLEU is a precision-based score while ROUGE is based on recall. We calculated the BLEU score at uni-gram and sentence levels and ROUGE at uni-gram (1), bi-gram (2) and Longest Common Sub-sequence (L) levels.

BLEU is a measure commonly used to evaluate machine-translated text, which can be leveraged to quantify the similarity between original and generated sentences. BLEU score is calculated by considering the similarity of the candidate sentence (eg: regarding our task, the machine generated text) with the reference sentence (original text) and it can take values ranging from 0 to 1. A value of 0 means that the candidate output has no overlap with the reference sentence while a value of 1 means there is perfect overlap with the reference.

As shown in the following equations (3.1, 3.2), the brevity penalty penalizes generated contents that are too short compared to the closest reference length with an exponential decay. The n-gram overlap counts how many unigrams, bigrams, trigrams, and four-grams (i=1,...,4) match their n-gram counterpart in the reference contents. This term acts as a precision measure.

$$\text{BLEU} = \underbrace{\min\left(1, \exp\left(1 - \frac{\text{reference-length}}{\text{output-length}}\right)\right)}_{\text{brevity penalty}} \underbrace{\left(\prod_{i=1}^{4} precision_i\right)^{1/4}}_{\text{n-gram overlap}} \tag{3.1}$$

with

$$precision_i = \frac{\sum_{\text{snt} \in \text{Cand-Corpus}} \sum_{i \in \text{snt}} \min(m^i_{cand}, m^i_{ref})}{w^i_t = \sum_{\text{snt'} \in \text{Cand-Corpus}} \sum_{i' \in \text{snt'}} m^{i'}_{cand}} \tag{3.2}$$

where

m^i_{cand} is the count of i-gram in candidate matching the reference content m^i_{ref} is the count of i-gram in the reference content w^i_t is the total number of i-grams in candidate content.

As depicted in equation 3.3, ROUGE-N is an n-gram recall between a candidate content and a set of reference contents.

$$ROUGE - N = \frac{\sum_{\text{snt} \in \text{references}} \sum_{gram_n \in \text{snt}} count_{match}(gram_n)}{\sum_{\text{snt} \in \text{references}} \sum_{gram_n \in \text{snt}} count(gram_n)} \tag{3.3}$$

where n stands for the length of the n-gram, $gram_n$, and $count_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries.

3.2 Detection as a Binary Classification Task

We considered detection as a binary classification task for which we employed several statistical as well as deep learning models to experiment detectability of the considered corpora. Leveraging a variety of classification models helps us in verifying that the datasets composed in our work are not biased towards a specific model type. Methodology adapted with each model and the respective results are elaborated under the following subsections.

3.2.1 Statistical Models and Their Ensembles Employed in Classification

Standalone Statistical Models for Classification

In our research we leveraged several statistical classification models based on Bag of Words [Har54] and tf-idf such as Multinomial Naive Bayes, Passive Aggressive Classifier Multinomial Classifier with Hyper-parameter (alpha) algorithms and Support Vector Machine (SVM) [CV95]. For the vocabulary, we considered not only single words but n-grams of size 1-3 words.

The "Bag of Words" (BoW) is a representation technique used to transform text data into numerical vectors that can be processed by machine learning algorithms. In a "bag of words" representation, the text is treated as an unordered collection (or "bag") of words, and the frequency of each word in the text is counted. The order of the words and the grammar of the text are disregarded; only the occurrence of words matters. This representation is particularly useful for tasks like text classification.

Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic used in natural language processing and information retrieval to evaluate the importance of a word within a document relative to a collection of documents, known as a corpus. TF-IDF is calculated by multiplying two values: Term Frequency (TF), which measures how often a term appears in a document, and Inverse Document Frequency (IDF), which measures how unique or rare a term is across the entire corpus. The idea is that a term is considered significant if it appears frequently in a specific document but is not overly common across the entire collection.

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem and it is widely used for tasks such as spam detection. It assumes that the features are conditionally independent given the class label. Passive Aggressive is a type of algorithm that aims to make aggressive updates when it encounters a misclassified point and passive updates when the point is correctly classified. SVM is a powerful supervised machine learning algorithm used for classification and regression tasks. It is a popular algorithm in text classification tasks.

Ensemble Architectures Formed with Statistical Classification Models

We harnessed the capabilities of ensembles comprising the aforementioned statistical models, applying various ensemble methodologies such as voting, stacking, bagging, and boosting. By amalgamating the predictions of multiple models, ensemble techniques aim to enhance the overall predictive power of our system.

1. Voting: combines the outputs through a majority or weighted decision.
2. Stacking: involves training a meta-model on the predictions of base models.
3. Bagging: leverages bootstrapped subsets of data for training individual models.
4. Boosting: iteratively adjusts model weights to prioritize difficult-to-classify instances.

Through these ensemble strategies, we sought to extract richer insights from our data and attain improved classification performance.

3.2.2 Recurrent Network Models and Their Ensembles Employed in Classification

Recurrent models, a subset of neural network architectures, are models designed to capture temporal dependencies and patterns within sequences.

Standalone Recurrent Network Models for Classification

We conducted experiments with Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory (Bi-LSTM) models, which are specialized recurrent neural network (RNN) architectures designed to address the challenges of capturing long-range dependencies in sequential data.

LSTM is a type of RNN that mitigates the vanishing gradient problem, allowing it to capture and remember information over long sequences more effectively. Traditional RNNs struggle with maintaining information over time due to the nature of gradient updates during training. LSTM introduces a memory cell and gating mechanisms that control the flow of information, facilitating the retention of relevant information over extended periods. This makes LSTMs particularly well-suited for tasks involving sequences with long-term dependencies, such as natural language processing and time series analysis.

Bi-LSTM enhances the capabilities of the LSTM by processing input sequences in both forward and backward directions. This bidirectional approach allows the network to consider context from both preceding and succeeding elements in the sequence. By capturing information from both directions, Bi-LSTM is better equipped to understand the context and dependencies within the entire sequence, leading to improved performance in tasks requiring a comprehensive understanding of the input data. This architecture is commonly used in applications like speech recognition, sentiment analysis, and named entity recognition.

In summary, while LSTM addresses the challenges of capturing long-term dependencies in sequential data, Bi-LSTM further enhances this capability by processing information bidirectionally, providing a more comprehensive understanding of the context within the input sequence.

Ensemble Architectures Formed with Recurrent Neural Network Models

In order to enhance the classification accuracies of our models, we employed an ensemble strategy by integrating them with a Convolutional Neural Networks (CNNs) ar-

chitecture. This novel hybrid approach, combining Recurrent Neural Networks (RNNs) with CNNs, aims to leverage the inherent strengths of each model in capturing temporal dependencies and spatial features, respectively.

The integration of the proposed RNN-CNN ensemble contributes to an overall improvement in predictive capabilities. RNNs are well-suited for understanding temporal relationships within sequential data, while CNNs excel in extracting spatial features from input data. By combining these strengths, our approach seeks to create a more robust and comprehensive model for classification tasks.

Importantly, we conducted end-to-end training of the entire ensemble. This holistic training approach allows the network to autonomously learn how to optimally combine the distinctive features extracted by both the LSTM (Long Short-Term Memory) and CNN components. This adaptability ensures that the ensemble model optimally exploits the synergies between temporal and spatial information, leading to enhanced classification performance. The end-to-end training methodology facilitates seamless integration, promoting synergy between the constituent models and enabling the ensemble to effectively learn and leverage the complementary aspects of both RNN and CNN architectures.

3.2.3 Transformer Architectures and Their Ensembles Employed in Classification

Standalone Transformer Architectures for Classification

For our classification experiments, we leveraged cutting-edge transformer models, namely BERT, DistilBERT, SciBERT, DeBERTa, XLNet and ELECTRA. These state-of-the-art architectures have demonstrated exceptional proficiency in a wide spectrum of natural language processing tasks, including classification.

BERT (Bidirectional Encoder Representations from Transformers) [KT19] is based on the Transformer architecture, which uses self-attention mechanisms to weigh the importance of different words in a sentence adaptively. The model is pre-trained on vast amounts of diverse text data, learning to predict missing words in a sentence. After pre-training, BERT can be fine-tuned for specific NLP tasks such as text classification, named entity recognition, question answering, and more. One of BERT's notable features is its ability to generate contextualized embeddings, providing a dynamic representation of words based on their context within a sentence. This contextual awareness has made BERT a cornerstone in various NLP applications, and its pre-trained representations often serve as a starting point for fine-tuning on domain-specific tasks. BERT has significantly advanced the state of the art in natural language understanding and has become a widely adopted model in the NLP community.

DistilBERT [San+19] is a distilled version of the BERT model, developed by Hugging Face. The primary goal of DistilBERT is to provide a more computationally efficient and lightweight alternative to the original BERT while maintaining competitive performance in natural language processing (NLP) tasks. The distillation process involves training DistilBERT to mimic the behavior of the larger BERT model by leveraging its knowledge and representations. This allows DistilBERT to capture the essential contextual information and semantic understanding present in BERT but in a more compact form.

SciBERT [BLC19], a model tailored for scientific text and domain-specific language understanding. The key distinction of SciBERT lies in its pre-training on large-scale scientific corpora, including research papers from disciplines such as computer science, biology, chemistry, and physics. This domain-specific pre-training allows SciBERT to grasp the unique vocabulary, syntax, and context prevalent in scientific texts. As a result, it excels in tasks related to scientific document analysis, information retrieval, and understanding specialized terminology.

DeBERTa (Decoding-enhanced BERT with Disentangled Attention) [He+20] introduces enhancements for better decoding and disentangled attention. One key feature of DeBERTa is its ability to dynamically adjust attention during both pre-training and fine-tuning phases, allowing the model to capture dependencies more effectively. By disentangling attention mechanisms, DeBERTa can better focus on relevant parts of the input sequence, enhancing its performance in tasks such as natural language understanding and generation. DeBERTa has demonstrated improvements in various downstream tasks, including text classification, question answering, and language modeling. The model's focus on disentangled attention aims to address challenges related to capturing long-range dependencies and understanding complex contextual relationships within text.

XLNet [Yan+19] introduces the concept of permutation language modeling, combining the strengths of both autoregressive and autoencoding models. Unlike BERT, which masks random words in a sentence for prediction during training, XLNet considers all words but still maintains the bidirectional context. It achieves this by leveraging the idea of permuting the order of words in a sequence and training the model to predict the original order. This approach allows XLNet to capture bidirectional context information while avoiding the limitations of purely autoregressive or autoencoding methods.

ELECTRA, short for "Efficiently Learning an Encoder that Classifies Token Replacements Accurately," [Cla+16] is a pre-training language model designed to improve the efficiency of large-scale language representation learning. ELECTRA follows the masked language model paradigm, similar to BERT (Bidirectional Encoder Representations from Transformers), but with a novel approach for training efficiency. In ELECTRA, a small portion of input tokens is replaced with incorrect ones, and the model is trained to distinguish between the original and replaced tokens. This approach is in contrast to BERT's masked language model, where some tokens are randomly masked, and the model learns to predict those masked tokens. ELECTRA's method is more computationally efficient as it avoids the need to predict every masked token, focusing on discriminating between genuine and replaced tokens instead.

Ensembles Formed with Transformer Architectures for Classification

We mainly built two forms of ensembles by utilizing transformer architectures,

1. **Ensembles created by combining the capabilities of SciBERT and DeBERTa models with the foundational BERT model.**
   As represented in architectural diagram 3.1, ensembling involves combining the individual capabilities of these models to achieve a more robust and accurate overall prediction.

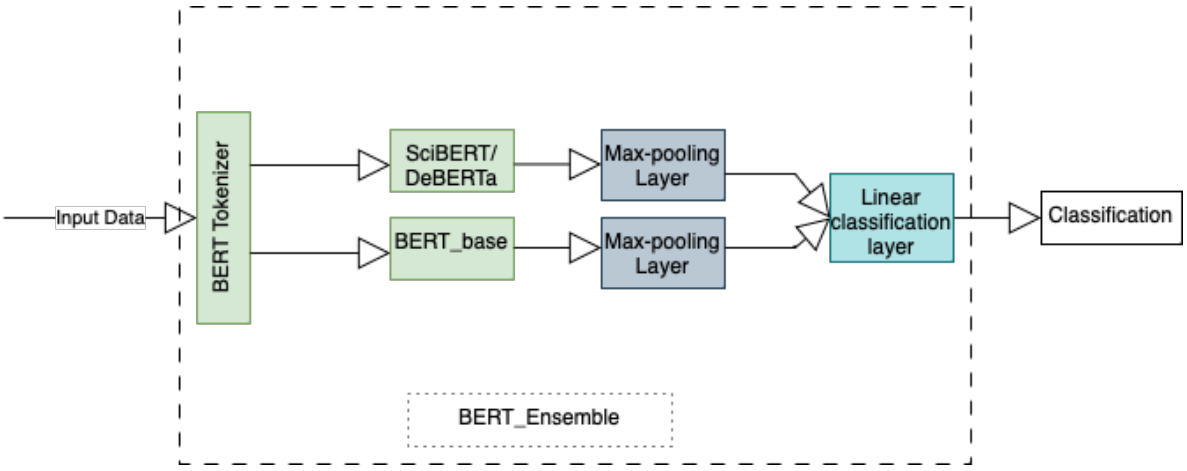

Figure 3.1 – Architecture of DeBERTa/ SciBERT + BERT Ensemble

The process begins by channeling the input data through each base model separately. Each base model consists of a transformer block, a crucial component in transformer architectures for capturing contextual information in the data, followed by a max pooling layer. The transformer block is responsible for processing and contextualizing the input sequence, while the max pooling layer helps extract the most salient features from the processed data.

After passing through the transformer block and max pooling layer of each base model, the outcomes from these individual models are collected and concatenated. This concatenation generates a unified representation that encapsulates the unique insights and features extracted by each model. This combined representation forms a richer and more comprehensive understanding of the input data.

Subsequently, the unified representation is channeled into a linear classification layer. This layer is responsible for making refined predictions based on the aggregated information from the ensemble of models. The linear classification layer uses

the concatenated features to generate a final output, which is interpreted as the model's prediction for the given input.

By ensembling SciBERT, DeBERTa, and BERT in this manner, we aim to capitalize on the diverse strengths of each model, capturing a broader range of contextual and semantic information from the input data. This ensemble approach enhances the overall predictive performance, making the model more robust and effective in handling a variety of natural language processing tasks.

2. **Ensembles created by incorporating the transformer model with Convolutional Neural Networks (CNNs) layers.**

The architectural diagram referenced as 3.2 illustrates the configuration of this combined model.

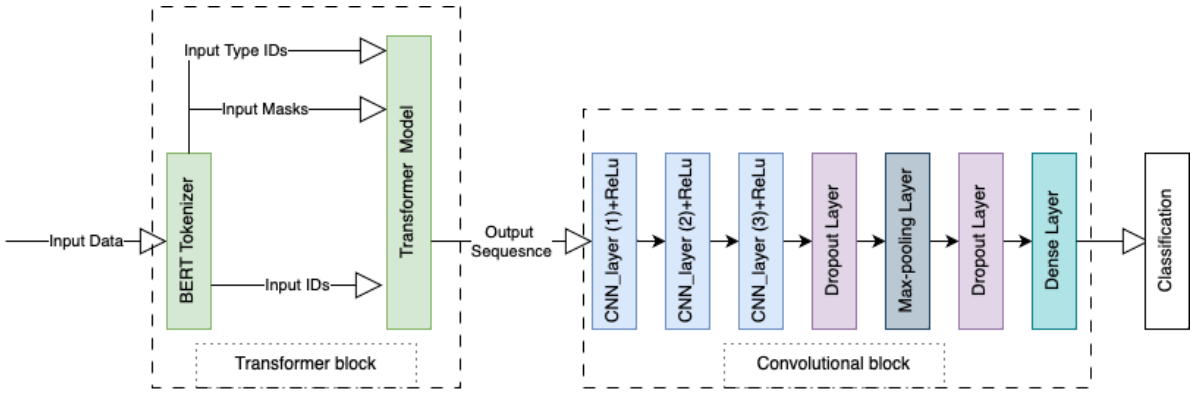

Figure 3.2 – Architecture of Transformer-CNN Ensemble (Here, the "input type ids," "input masks," and "input ids" are the components used to prepare and encode the input data for the transformer model.)

In developing the Transformer-CNN ensemble architecture, we followed a similar approach to what Andrew Fogarty have proposed in their article [1]. The process begins with the transformer model, which is employed to generate embeddings for the input data. These embeddings capture the contextual information and semantic features of the input sequence, providing a rich representation of the data. Instead of utilizing nn.Embedding layers and a lookup table, as is often done in traditional approaches [Kim14], we directly use the transformer-generated embeddings as the input for the subsequent CNN layers.

The CNN component consists of three stacked convolutional layers, designed to cover a substantial portion of the input data and extract relevant features through convolutional

1. $http://seekinginference.com/applied_nlp/bert-cnn.html$

operations. This hierarchical stacking allows the model to capture different levels of abstraction in the input sequence. Following the convolutional layers, the output undergoes a dropout operation, introducing a form of regularization to prevent overfitting.

Next, a max pooling layer is applied to the output of the convolutional layers. Max pooling helps retain the most salient features by selecting the maximum value within specific regions of the data. Another dropout layer is introduced after max pooling to further enhance the model's robustness.

The processed data is then fed into a dense layer for the final classification. The dense layer is responsible for transforming the features extracted by the transformer and CNN components into a format suitable for the desired classification task. The absence of nn.Embedding layers in our approach reflects the seamless integration of the transformer and CNN components, with the embeddings directly serving as the input for the subsequent CNN layers.

Overall, this ensemble architecture, combining the strengths of transformers and CNNs, is tailored to effectively capture both contextual information and local features in the input data. The resulting model exhibits enhanced performance, making it well-suited for a variety of natural language processing tasks.

3.3 Leveraging Detection Tools to Distinguish Machine Generated Content from Human Written Content

Currently, there is a numerous number of detection tools which can be employed to distinguish machine generated text from human written text. These tools leverage advanced algorithms, linguistic analyses, and anomaly detection techniques to scrutinize various aspects of the text, such as writing style, language patterns, and contextual coherence. Ranging from stylometric analysis to rule-based systems, these tools play a crucial role in addressing the challenges posed by the proliferation of machine-generated content, providing valuable resources to verify the authenticity of textual information in diverse contexts. Researchers and practitioners can choose from a diverse array of these tools, each offering unique capabilities and methodologies, to enhance the accuracy and reliability of discerning between human and machine-generated text. In our work, we employed the following detection tools,

1. **GLTR (Giant Language Model Test Room) [GSR19]:**

   In our evaluation process, we utilized GLTR , a visualization tool designed to assist humans in distinguishing between text generated by large language models and human-written content. It analyzes the probability distribution of tokens generated by a language model, highlighting those that are most likely to be machine-generated. The tool provides visualizations where different colors represent the likelihood of a token being among the top-ranked predictions by the model. For instance, tokens in green are among the top 10 most probable, facilitating a quick and intuitive assessment of the generated text's authenticity. GLTR is particularly useful for evaluating the naturalness and distinguishability of text produced by language models, aiding researchers and practitioners in understanding and mitigating the challenges associated with automated content generation.

2. **DetectGPT[Mit+23]:**

   DetectGPT is a tool that assesses whether a given text has been generated by a GPT (Generative Pre-trained Transformer) model or if it originates from human-written content. The evaluation is based on a Z-score calculation, which is a statistical measure indicating how many standard deviations a particular data point is from the mean of a group of data. Here's a breakdown of the process:

— Original Log Probability: DetectGPT starts by computing the log probability of the tokens in the original text. The log probability provides a measure of how likely each token is according to the language model.
— Perturbed Log Probability: Perturbation involves introducing slight modifications or noise to the original text. DetectGPT computes the log probability of the perturbed text tokens.
— Z-Score Calculation: The Z-score is then calculated by finding the difference between the original log probability and the average perturbed log probability. This difference is divided by the standard deviation of the perturbed log probability.
— Thresholds for Claiming Generation: If the resulting Z-score is greater than 1, DetectGPT asserts that the text is likely generated by a GPT model. A Z-score greater than 1 indicates that the original log probability significantly deviates from the perturbed log probability. Conversely, if the Z-score is lower than 0.25, DetectGPT claims that the text is not generated by a GPT model. A Z-score below 0.25 suggests that the original log probability is well within the range of perturbed log probabilities expected for human-generated text.

In summary, DetectGPT uses statistical analysis through Z-scores to quantify the likelihood that a given text is machine-generated by a GPT model. This approach considers the deviation of the original log probability from the average perturbed log probability, providing a threshold-based classification for the origin of the text.

3. **GPT-2 Output Detector[Sol+19]:**

GPT-2 Output Detector is a model introduced by OpenAI concurrently with the release of the weights for the largest GPT-2 model. It is designed by fine-tuning a RoBERTa model with the outputs of the 1.5B-parameter GPT-2 model. This detector indicates the fake token percentage of a given text and the tool claims to have an accuracy around 95% for detecting GPT-2 generated text. One can run this detector model either by launching their web UI or even by training the model on a new dataset. The tool examines your text and shows the chance it was written by a human or an AI. It uses green for real content and red for AI-generated. Nevertheless not much details about this detector architecture are publicly available.

4. **GPTZero [2]:**

   GPTZero, a language model designed to distinguish between human-written content and material generated by AI models. Trained on a diverse corpus [TC23] encompassing both human and AI-generated text, with a focus on English prose, GPTZero introduces three key features:

   — Multi-Level Classification: GPTZero offers users the ability to assess the AI-generated content within a document at multiple levels—sentence, paragraph, and document. On a sentence level, it adeptly identifies portions crafted by AI or human authors, enabling a precise evaluation of the proportion of AI-generated content in the document. For documents yielding mixed outcomes, GPTZero furnishes a score indicating the likelihood that the entire document was AI-generated.

   — Perplexity Score: The Perplexity Score serves as a measure reflecting how effectively a language model can predict a sequence of words.
   Perplexity is defined as the exponentiated average negative log-likelihood of a sequence. If we have a tokenized sequence $X = (x_0, x_1, ..., x_t)$,
   then the perplexity of X is,

$$PPL(X) = \exp\{\frac{-1}{t}\sum_{i}^{t} logp_\theta(x_i|x_{<}i)\} \tag{3.4}$$

   where $logp_\theta(x_i|x_{<}i)$ is the log-likelihood of the ith token conditioned on the preceding tokens $x_{<}i$ according to the considered model [3].
   GPTZero computes this score for each document, providing insights into the model's uncertainty when distinguishing between human and AI-generated content. A low perplexity score indicates high confidence in the model's predictions, while a higher score implies a degree of uncertainty, offering valuable information on the model's text classification accuracy.

   — Burstiness Score: GPTZero introduces the Burstiness Score, a metric gauging the recurrence of similar phrases or topics within a document. This score helps determine the level of similarity between different sections.
   Burstiness can be calculated using the following formula:

$$B = (\lambda - k)/(\lambda + k) \tag{3.5}$$

2. `https://gptzero.me/`
3. https://huggingface.co/docs/transformers/en/perplexity

where, B = Burstiness, λ = Mean inter arrival time between bursts and k = Mean burst length [4].
A higher burstiness Score suggests increased chances of AI involvement in text generation, as machine-generated content often exhibits repetitive patterns.

However, [WF23] claims that by introducing spelling mistakes in generated content and processing them with QuillBot [5] can make the results of GPT-2 Output Detector and GPTZero less accurate.

4. https://ramblersm.medium.com/exploring-burstiness-evaluating-language-dynamics-in-llm-generated-texts
5. https://quillbot.com

Chapter 4

Results and Evaluation

This chapter provides scores generated using various evaluation measures employed by different models and tools.

4.1 BLEU and ROUGE Scores

As previously mentioned in Chapter 3, BLEU and ROUGE scores were used to evaluate the quality of the generated corpora, in comparison to the original (human written) content.

We calculated BLEU and ROUGE scores for full article-datasets, which were generated using the fine-tuned GPT-2 models, the abstract dataset generated using pre-trained GPT-2 and the hybrid dataset. The respective average BLEU and ROUGE scores are provided in Table 4.1.

Corpus	**U-BLEU**	**S-BLEU**	**Rouge-1**	**Rouge-2**	**Rouge-L**
D1: Fine-tuned GPT-2(temp = 0.7)	0.867	0.809	0.853	0.810	0.853
D2: Fine-tuned GPT-2(temp = 0.9)	0.858	0.766	0.834	0.810	0.834
D3: Hybrid dataset	0.824	0.792	0.882	0.840	0.881
D4: GPT-2 without fine-tuning(temp = 0.7)	0.467	0.356	0.533	0.509	0.533

Table 4.1 – Average BLEU and ROUGE Scores

Rouge-1 and Rouge-L scores are almost similar when they are rounded off to three decimal places. But when we consider more decimal places, it can be seen that the Rouge-1 scores are slightly higher than the Rouge-L scores. If we consider overall Unigram *vs.* sentence BLEU, as the unigram BLEU score is more tolerant than the sentence one, it is normal that the sentence BLEU scores, although quite high, are lower than the unigram BLEU scores.

It can be seen that the average BLEU scores are always higher in the fully generated dataset compared to hybrid dataset, which means the fully generated dataset has a better precision (that is, a good portion of the generated n-grams are also in the original text). On the other hand, it can be seen that the average ROUGE scores of the fully generated dataset is lower than the hybrid dataset. This means that the recall of the hybrid dataset is at a better level compared to the other dataset (this is expected, as parts of the original text are included in the generated one). The minimum scores are relatively high for both BLEU and Rouge, indicating that there is a good degree of similarity between the texts even when they are most dissimilar.

Despite the trade-off between precision and recall, the overall results show that the academic contents that are generated with fine-tuned models have gained quite high (more than 0.8), which indicates that those contents are quite similar to their respective natural ones. Of course, some of the generated texts could be identified as “copies" of the original texts but they have nevertheless been artificially generated and it is interesting to evaluate the detection models against these texts that “look" natural.

It can be seen that the n-gram scores are decreased when the model temperature is increased. This is justifiable since the more the randomness is the more the differences in the generated text in comparison to the original. Moreover, there is a significant decrease of scores in data produced by the pre-trained model, manifesting the fact that when a generation model is not fine-tuned, it tends to output more random (out of the context) content.

4.2 Results Produced by Classification Models

4.2.1 Experimental Setup

The text underwent initial processing, which included the removal of stopwords and stemming, before being fed into either statistical or neural network architectures. After this preprocessing, the data was converted into numerical vectors using Bag of Words (BoW) or tf-idf encoding techniques. These numerical representations were then used as inputs for the statistical models. All the statistical models, along with their corresponding ensemble methods, were implemented using the Scikit-learn library.

To construct Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) models, the relevant layers were imported from the TensorFlow's Keras module. The training process for these recurrent models, including those integrated with CNN ensembles, involved running 10 epochs. For the LSTM and Bidirectional LSTM (Bi-LSTM) architectures, training was performed with batch sizes of 64 and 128, respectively. This comprehensive approach encompassed both traditional statistical methodologies and advanced neural network architectures for a thorough exploration of text data.

Concerning transformer architectures and their associated ensembles, pre-trained models from Hugging Face [Wol+20b] were imported and subsequently fine-tuned using Simple Transformers [1]. An overview of the transformer models utilized in our work is provided in Table 4.2. Consistency was maintained across all models by using the BERT tokenizer. The fine-tuning process comprised 3 epochs, a batch size of 16, and a maximum sequence length of 128. Utilizing the T4 GPU Hardware accelerator, the average training time for models was approximately 30 minutes. For standalone models, the input consisted of unprocessed text, while ensembles underwent pre-processing, including punctuation removal and conversion to lowercase. This meticulous approach ensured the effective integration of transformer models and their ensembles into the overall methodology.

The datasets were split into 60:20:20 for training, testing and validation. To assess the classification performance of the models under consideration, the F1 score was employed. This score, being a balanced combination of precision and recall, offers a comprehensive evaluation. Each model underwent a total of five experimental iterations, and the resultant average F1 scores are presented in the following tables.

1. `https://simpletransformers.ai`

Model	Vocab (K)	Hidden Size	Layers	Batch Size	Parameters(M)
BERT_{base}	30	762	12	64	110
DistilBERT	30	768	6	16	66
SciBERT_{base}	30	768	12	16	110
RoBERTa_{large}	50	1024	16	16	355
DeBERTa_{large}	50	1024	24	16	350
Electra_{base}	30	768	12	16	110
XLNet_{base}	32	768	12	16	110

Table 4.2 – Hyper Parameter of the Classification Models

Classification Model	Fully Generated	Hybrid
BoW, Multinomial Naive Bayes Algorithm	19.7	24.2
BoW, Passive Aggressive Classifier Algorithm	31.8	30.3
BoW, SVM	37.9	37.9
LSTM model	59.1	50.0
Bi-LSTM (Latest Paper)	40.9	47.0
BERT	52.5	50.0
DistilBERT	**62.5**	**70.2**

Table 4.3 – Classification Results for Fully Generated Dataset and Hybrid Dataset (BoW: Bag of Words)

4.2.2 Results Produced by Classification Models on the Benchmark Dataset

As we explained in Chapter 2, our benchmark dataset is comprised of two corpora: one containing automatically generated papers and another hybrid dataset which contains original (human written) abstracts in which some sentences are substituted with machine generated sentences.

The classification results are presented in Table 4.3. As per the results, the highest classification score was obtained by the DistilBERT model regarding both datasets: the scores are 62.5% and 70.2% for the fully generated dataset and the hybrid dataset, respectively. These results show that the generated corpora are competent enough to be used as a baseline datasets to experiment detection.

As expected, the scores differs from one model to the other, the deep learning based models having higher accuracy scores than the statistical models. The higher scores are obtained by DistilBERT model on both datasets: 62.5% and 70.2% for the fully generated

Model	Fully Generated Dataset	Hybrid Dataset	Maronikolakis et al., 2020
Bi-LSTM	40.9	47.0	82.8
BERT	52.5	50.0	**85.7**
DistilBERT	**62.5**	**70.2**	85.5

Table 4.4 – Classification Results Comparison for Bi-LSTM, BERT and DistilBERT models

dataset and the hybrid dataset, respectively. Interestingly, if most models perform slightly better on the Hybrid Dataset, it is not the case of LSTM model which achieves a much better score on the Fully Generated Dataset and of BERT and passive aggressive classifier at lesser degrees. This is an argument for including both datasets and different types of generated data in our benchmark. Despite these differences, we observe globally that the accuracy scores are not very high, even for DistilBERT. This is the most important point to assess the quality of our benchmark in terms of classification difficulty.

We did a comparison of our results to the latest research works [MSS21] and the results are depicted in Table 4.4. The overall accuracies regarding our datasets are lower when compared with the aforementioned research. This may be due to the fact that Maronikolakis *et al.* focus on the generation of short content (headlines) but it shows that our datasets are more difficult to classify than theirs, which makes it a better benchmark proposal.

4.2.3 Results Produced by Classification Models on Various Corpora

We experimented detectability of the considered corpora by classifying them using transformer models. All the models were trained for 3 epochs and the average F1 scores are presented in Table 4.5.

In the overall comparison, data derived from WikiGPT, DAGPap22, and the original GPT2-generated dataset without fine-tuning exhibit superior F1 scores when compared to the other two datasets produced using a GPT2 model that underwent fine-tuning. This indicates that distinguishing the generated content from the original is more challenging in the latter cases. The difficulty arises because, during fine-tuning with original data,

Model	**WikiGPT**	**DAGPap22**	**Fine-tuned GPT2 (0.7)**	**Fine-tuned GPT2 (0.9)**	**GPT2 without finetuning (0.7)**
SciBERT	92.97	95.02	84.65	79.16	94.99
RoBERTa	97.00	96.87	67.83	33.33	84.85
DeBERTa	98.50	97.17	67.03	48.13	95.00
Electra$_{base}$	81.95	96.64	56.16	48.13	95.00
XLNet$_{base}$	84.12	95.67	56.36	59.60	97.50

Table 4.5 – F1 Scores Produced by Models on Classification Task

the GPT2 model adapts to the original content, enabling it to generate text that closely resembles the input. Notably, SciBERT achieves the highest classification results for the fine-tuned datasets. This can be attributed to SciBERT being pre-trained on extensive scientific domain corpora, allowing it to better capture the nuances and context specific to scientific language. As a result, SciBERT demonstrates a heightened capability to discern and classify content generated by a fine-tuned GPT2 model, particularly in domains with specialized and technical language.

4.2.4 Results Produced by Classification Models & Their Ensembles on ALTA Shared Task Data

Each model underwent a total of five experimental iterations, and the resultant average F1 scores are presented in Table 4.6.

In general, the ensemble architectures have exhibited superior performance compared to their corresponding original models. Our best-performing solution is the combination of DeBERTa$_{large}$ with CNN, achieving an F1 score of **98.36%**.

Considering that baseline models such as Naïve Bayes and tf.idf weighting obtain scores close to 90%, it is clear that the dataset is not well balanced. In fact, looking at the Multinomial Naïve Bayes and the log probabilities differences for all features, we observed a thematic bias. Specifically, the top most probable words in the negative category (human-generated) are law-oriented: “plaintiff", “defendant", and “judgment". On the other hand, LLM-generated text contains words like “round", “league", “players", etc. Therefore, it is not clear whether these results are generalizable to the general task of detecting artificial text.

Model	F1
Statistical Models	
NB + BoW	89.04
PA + BoW	84.07
SVM + BoW	87.51
NB + tf-idf	89.02
NB + tf-idf	91.00
NB + tf-idf	91.42
Ensembles of Statistical Models	
Voting (NB + PA + SVM) + BoW	90.29
Stacking (NB + PA + SVM) + BoW	88.23
Bagging (NB + PA + SVM) + BoW	91.56
Boosting (NB + PA + SVM) + BoW	90.28
Recurrent Models	
LSTM	49.08
Bi-LSTM	90.58
Ensembles of RNNs	
LSTM + CNN	49.08
Bi-LSTM + CNN	90.02
Transformer Models	
BERT_{base}	90.81
SciBERT	94.89
DeBERTa_{large}	96.67
XLNet_{large}	93.62
Ensembles of BERT models	
BERT_{base} + SciBERT	97.80
BERT_{base} + DeBERTa_{large}	97.47
Ensembles of transformers with CNN	
BERT_{base} + CNN	97.42
SciBERT + CNN	97.56
DeBERTa_{large} + CNN	**98.36**
XLNet_{base} + CNN	97.44

Table 4.6 – Classification Scores

4.2.5 Results Produced by Classification Models on Hotel Review Data

The dataset was split into an 80:20 ratio for training and testing. Each model underwent three experimental iterations, and the average F1 scores resulting from these experiments are provided in Table 4.7. It can be seen that the F1-scores for the GPT-3 generated dataset and paraphrased dataset are higher in general, indicating that they are easier to detect than the GPT-2 based one, which was built with fine-tuning to be less predictable, generating reviews that are more similar in style to the human-written ones. Among all LLMs, BERT$_{base}$ seems to be the most effective in detecting the generated content.

Model	GPT-2	GPT-3	paraphrased
Statistical Models			
NB + BoW	89.04	90.45	92.31
PA + BoW	84.07	88.73	88.03
SVM + BoW	87.51	90.24	90.33
NB + tf-idf	83.07	87.23	88.01
PA + tf-idf	91.00	92.45	92.31
SVM + tf-idf	91.42	93.25	94.67
Ensembles of Statistical Models			
Voting (NB + PA + SVM) + BoW	90.29	92.23	90.43
Stacking (NB + PA + SVM) + BoW	88.23	89.01	90.67
Bagging (NB + PA + SVM) + BoW	91.56	90.87	90.45
Boosting (NB + PA + SVM) + BoW	90.28	92.00	92.53
Recurrent Models			
LSTM	49.08	53.98	55.04
Bi-LSTM	90.58	91.24	92.03
Ensembles of RNNs			
LSTM + CNN	49.08	50.03	52.45
Bi-LSTM + CNN	90.02	91.28	92.34
Transformer Models			
BERT$_{base}$	97.83	99.38	98.29
SciBERT	93.66	93.75	97.62
XLNet$_{large}$	87.87	92.70	95.32
ELECTRA	92.49	93.49	95.37
Ensembles			
BERT$_{base}$ + CNN	98.32	99.01	96.41
SciBERT + CNN	94.05	95.27	99.79
XLNet$_{base}$ + CNN	93.21	92.70	98.23
ELECTRA$_{small}$ + CNN	93.40	93.82	90.45

Table 4.7 – F1 Scores obtained by Classification Models on Hotel Review Data

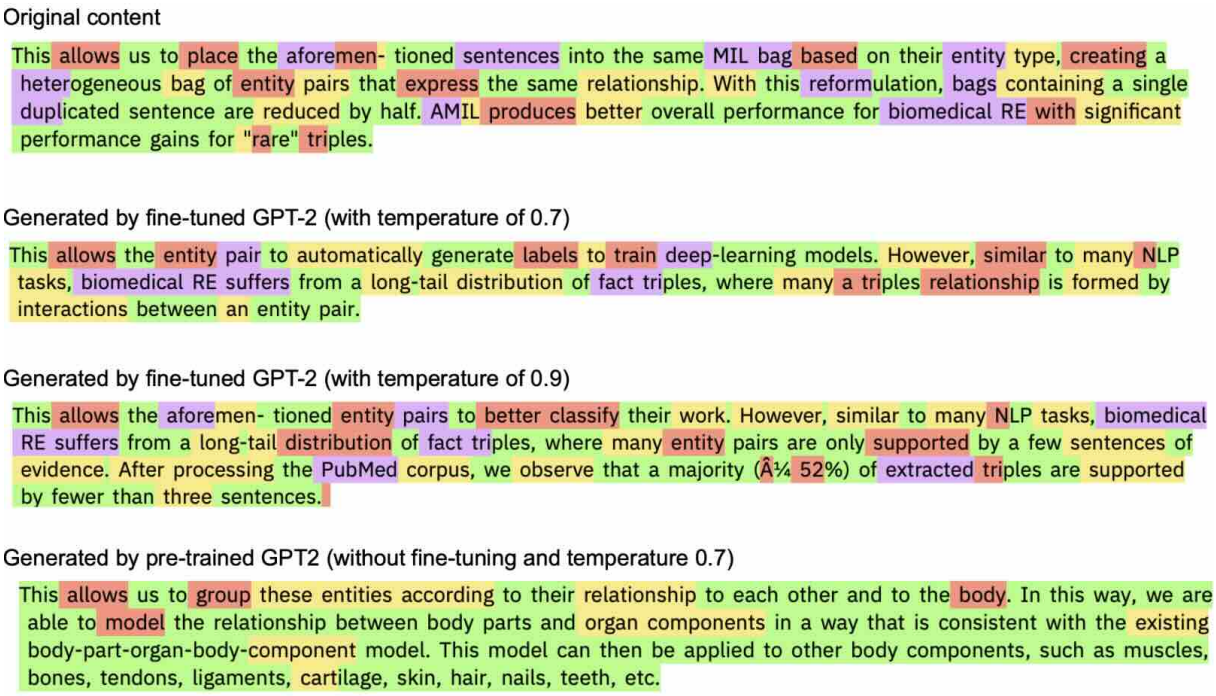

Figure 4.1 – GLTR Outputs for Excerpts of Corpora.

4.3 Results Generated by Detection Tools

4.3.1 Visualizations Produced by GLTR

Since GLTR[GSR19] is a visualization tool to support humans in distinguishing artificially generated text, we leveraged it to examine the detectability of the various corpora. Figure 4.1 shows visualizations provided by GLTR for a few excerpts extracted from several datasets. Here, in green it represents text that belongs to the 10 most probable tokens produced by GPT-2, which means they are very likely to be automatically generated. Likewise, the tool shows tokens belonging to the 100 most probable ones in yellow, top 1,000 in red and in violet those who fall outside the 1,000 most probable tokens. Therefore, according to GLTR, the excerpt produced by the pre-trained model is highly distinguishable from the original one. On the contrary, the other two excerpts which were generated by fine-tuned models have similar visualizations to the original text.

Model	WikiGPT	DAGPap22	Fine-tuned GPT2 (0.7)	Fine-tuned GPT2 (0.9)	GPT2 without finetuning (0.7)
Z-score	1.747	0.240	-0.351	-0.192	0.911

Table 4.8 – Z-scores Produced by DetectGPT

4.3.2 Results Produced by DetectGPT

Since the majority of our considered data are generated using GPT models, we leveraged the latest DetectGPT[Mit+23] to check the detectability of our generated data. DetectGPT calculates a Z-score which is computed by considering the difference of the original log probability of text tokens and the average perturbed log probability as a proportion of the standard deviation of the perturbed log probability. If the score is greater than 1, then the text is claimed to be generated, if it is lower than 0.25, then the text is claimed to be not generated by a GPT model. We ran the experiments with 20% of the generated data from each dataset (due to time constraints) for 3 runs and the average score is represented in Table 4.8.

The highest z-score gained by the WikiGPT dataset proves its high detectability. Also, the abstracts generated by the GPT2 without fine-tuning gained a low score demonstrating the likelihood of being detected. Low z-score for DAGPap22 data, states that they are not generated by a GPT model (although some of its data are generated so). This might be due to the reason that the generated text in DAGPap22 dataset is produced by several other models in addition to GPT models, thus making the average Z-score slightly lower than 0.25. Surprisingly the two other datasets which are generated using fine-tuned GPT2 are recognized as not containing data generated by a GPT model. Therefore, it is important to note that when a GPT model is fine-tuned, the generated data is difficult to be distinguished by DetectGPT itself.

Model	**DAGPap22**	**Fine-tuned GPT2 (0.7)**
GPT2 O/P detector (Fake token %)	99.98%	0.03%
GPTZero (Avg Perplexity Score)	15.20	87.18

Table 4.9 – Results Produced by GPTZero and GPT-2 Output Detector

4.3.3 Results Produced by GPTZero and GPT-2 Output Detector

We conducted an experiment to assess the detectability of two tools, GPTZero and the GPT-2 Output Detector, on two distinct corpora: the DAGPap22 dataset and a dataset entirely generated by the GPT-2 model. The outcomes, as outlined in Table 4.9, unveil intriguing insights. Notably, the DAGPap22 dataset exhibits a high percentage of fake tokens and a low perplexity score, indicating the prevalence of synthetic text. In contrast, the fully generated dataset shows a low fake token percentage and a high perplexity score, challenging the tools' claims of the absence of generated content in this corpus.

This discrepancy in results underscores the complexity of accurately detecting machine-generated text, particularly when such text is created using a fine-tuned model. Despite the purported high accuracies of these detection tools, they encounter difficulties in distinguishing machine-generated content correctly, emphasizing the need for further refinement and comprehensive evaluation in real-world scenarios.

Chapter 5

Further Experiments on Detection Task

5.1 Examining the Influence of Attention Feature of Transformer Based Models on Classification Task

Transformer architectures have revolutionized natural language processing by incorporating attention mechanisms that assign varying degrees of importance to different input tokens during sequence processing. Thus in our research we wanted to examine the relationship between the attention features and their consequences on the outcomes of classification tasks.

The study likely involves a multifaceted approach, encompassing experimental design, data analysis, and interpretation of results. Researchers are expected to scrutinize the attention mechanisms at various levels of the Transformer model, potentially including self-attention patterns and layer-wise attention distributions. By delving into the attention mechanisms, the goal is to discern patterns, understand their influence on classification decisions, and identify factors contributing to both accurate and erroneous classifications.

In the course of this undertaking, we harnessed the capabilities of Bertvitz [Vig19], an interactive visualization tool meticulously designed for the analysis of attention mechanisms within Transformer language models like BERT, GPT-2, or T5. Bertvitz offers an insightful interface that enables a comprehensive examination of attention patterns exhibited by these advanced language models. This tool serves as a valuable asset, facilitating a better exploration of the intricate attention mechanisms present in Transformer architectures during the analysis of textual data.

Bertvitz provides three types of views regarding attention namely head view, model view and neuron view. For our experiments we chose the head view since it is easier in understanding the attention distribution of tokens. The attention head view visualizes attention for one or more attention heads in the same layer. Following diagrams 5.1, 5.2 and 5.3 represent the attention head view of the layer one of the three models BERT, DistilBERT and RoBERTa regarding two sentences (an original sentence and its corresponding generated sentence) extracted from the corpus containing fully generated articles (D1).

Upon analyzing the attention head views of the examined BERT variants, a noteworthy pattern emerged. Specifically, when a specific word, such as "poorer," is selected, all models exhibit heightened attention on identical tokens, such as "CLS" and "performance." However, it's crucial to note that the attention for these tokens occurs in distinct layers, as indicated by the varying color bands. This discrepancy across layers could potentially contribute to the observed differences in classification accuracies among the BERT variants concerning the detection task.

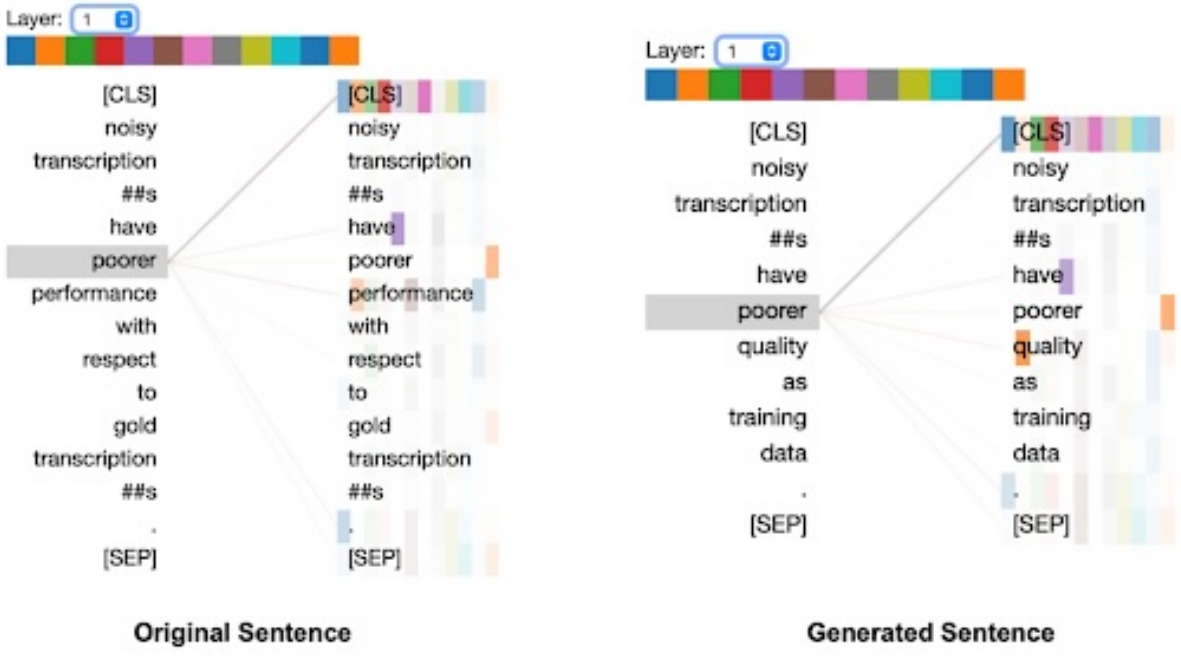

Figure 5.1 – Attention Head View of BERT Model

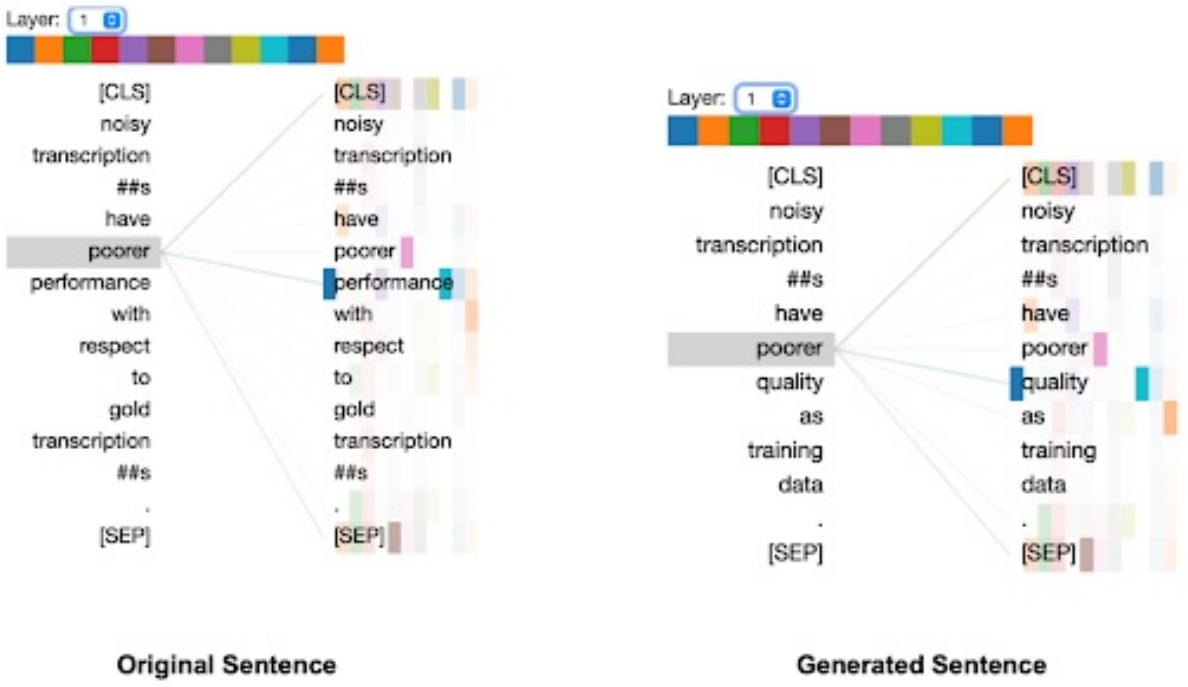

Figure 5.2 – Attention Head View of DistilBERT Model

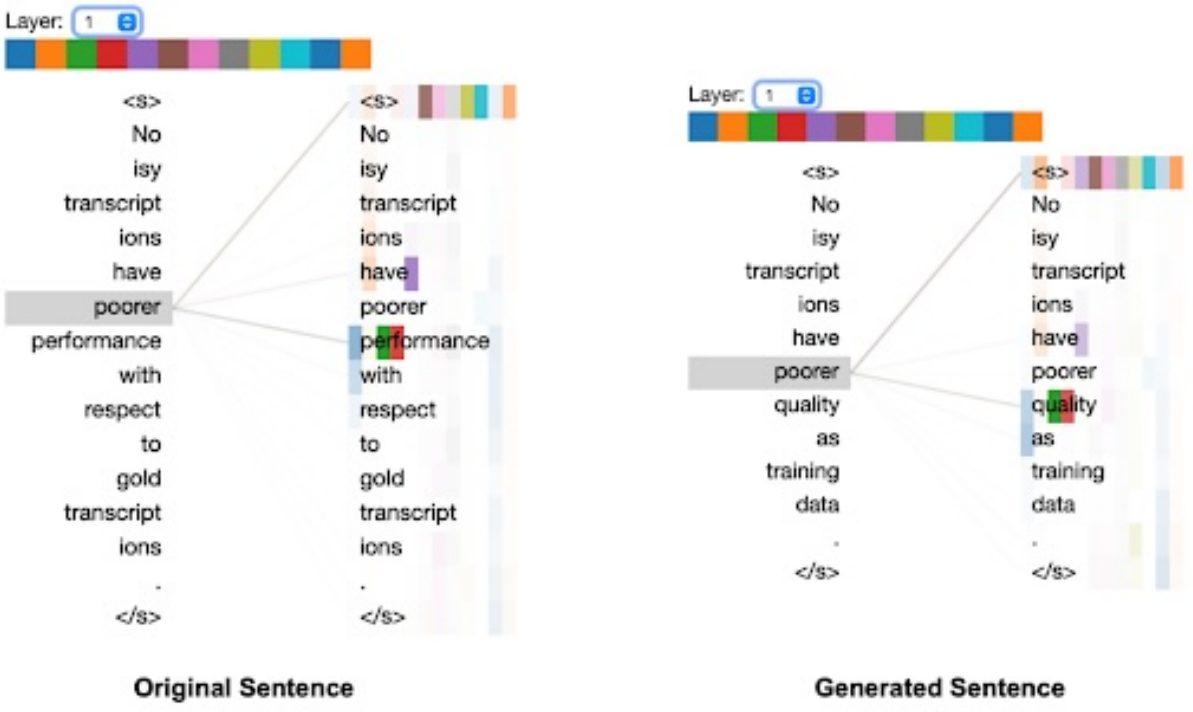

Figure 5.3 – Attention Head View of RoBERTa Model

5.2 Cross Validation Performed on Classification Task

Cross-validation stands as a fundamental and widely embraced methodology in machine learning, particularly within the realm of classification tasks. Its significance lies in its pivotal role in evaluating model performance, ensuring generalizability, and safeguarding against the pitfalls of overfitting or underfitting. The crux of cross-validation lies in its capability to furnish a more robust and dependable estimate of a model's proficiency compared to a singular train-test split.

Overfitting, a common challenge, arises when a model becomes overly attuned to the training data, capturing noise and outliers that lack relevance to new data. Cross-validation serves as a potent tool in identifying and mitigating overfitting by scrutinizing the model's performance across diverse subsets, thereby reducing the risk of being misled by the nuances of a singular train-test partition.

Beyond this, cross-validation plays a vital role in uncovering data variability. Given that datasets often manifest variability, a solitary random division into training and testing sets may inadequately encapsulate the complete spectrum of patterns within the data. Employing multiple folds in cross-validation reveals the model's ability to generalize across distinct data distributions, ensuring resilience and insensitivity to the peculiarities of a specific subset.

Moreover, cross-validation proves invaluable in the realm of hyperparameter tuning. By executing cross-validation for various hyperparameter configurations, one can pinpoint the set of hyperparameters that elicits optimal average performance across multiple folds. This process contributes to the development of a more robust and broadly applicable model, enhancing its effectiveness in real-world applications.

In our study, we incorporated cross-validation as a pivotal strategy to enhance the precision of our classification task outcomes. Employing a five-fold validation approach on our dataset, the ensuing Table 5.1 encapsulates the cross-validation results for three distinct datasets: the fully generated article corpus with a temperature of 0.7, the fully generated article corpus with a temperature of 0.9, and the hybrid corpus. Here we have considered the Accuracy or the Fraction correct which is used in binary classification,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where, TP = True positive, FP = False positive, TN = True negative and FN = False negative

Classification Model	**Fully Generated (temp = 0.7)**	**Fully Generated (temp = 0.9)**	**Hybrid**
BoW, Multinomial Naive Bayes	12.0	14.4	16.4
BoW, Passive Aggressive	22.4	18.0	23.2
BoW, SVM	33.6	27.2	32.8
LSTM	62.5	57.5	12.5
Bi-LSTM	17.5	42.5	12.5
BERT	49.5	47.0	41.5
DistilBERT	49.0	53.0	52.0
RoBERTa	13.5	12.0	14.5

Table 5.1 – Cross Validation Results (Metric: Accuracy /Fraction Correct)

According to the shown results, all the models have comparatively low scores. Thus we considered inverse classification to evaluate the results.

Inverse classification, also known as reverse classification, involves swapping the roles of the target and predictor variables in a machine learning task. In the context of cross-validation, inverse classification is essential for assessing the robustness and reliability of a model by examining its ability to perform effectively in both forward and backward scenarios. Thus, in terms of the inverse classification, RoBERTa model outperforms the other considered models.

5.3 Examining the Log Probabilities to Further Understand the Classification Results of the Hotel Review Dataset

To assess the detectability of fake reviews, we employed a diverse set of classification models. Our initial approach involved utilizing a basic Multinomial Naïve Bayes model with tf-idf weights, excluding lemmatization and stopwords removal. The model underwent evaluation using an 80:20 random split for training and testing. Remarkably, this model yielded highly favorable results, achieving an impressive F-1 score of 96%, suggesting that discerning fake reviews could be accomplished solely by examining the vocabulary.

To delve deeper, we conducted a comparative analysis of the log-probabilities associated with words in the generated and non-generated classes, calculating the differences. The findings are presented in Table 5.2, showcasing the 20 most discriminating words for both classes.

The analysis reveals a distinct pattern in the most discriminative words for the generated category, predominantly encompassing attributes like "unhelpful," "terrible," "delicious," and "outdated." Conversely, the discriminating words for the non-generated category exhibit associations with objects or places, such as "door," "floor," "coffee," "Michigan," and "Ave," along with personal pronouns like "she," "he," and "your."

In further exploration, we conducted similar experiments utilizing bi-grams and trigrams as features instead of individual words. This shift accentuated the stylistic differences even more prominently. Notably, the most critical trigrams for the generated class displayed a recurring pattern of "X was/were Y," where X typically denotes a service or an aspect of the hotel, and Y represents an adjective. In contrast, the most representative trigrams for the non-generated class comprised phrases like "in the room," "in the bathroom," and "the first night." This contrast in stylistic preferences aligns with expectations, as prior research on generated text detection, such as [Ant+23], has highlighted the propensity of Language Models to produce recurrent patterns in their output.

As shown in table 5.3 ,upon conducting a comparative analysis of log-probabilities within corpora comprised of academic text, we noted that a similar level of difference in vocabulary that we observed for the hotel review is not detectable in the academic texts. This observation suggests that while regularities in style and vocabulary are generally

Generated		**Authentic**	
Word	**delta**	**Word**	**delta**
unhelpful	2.889	door	-1.694
incredibly	2.620	floor	-1.680
delicious	2.609	coffee	-1.656
outdated	2.402	next	-1.636
terrible	2.306	your	-1.557
accommodating	2.257	concierge	-1.513
anyone	2.229	she	-1.478
uncomfortable	2.129	ave	-1.468
amenities	1.907	mile	-1.420
musty	1.873	etc	-1.402
enjoyable	1.725	call	-1.387
unprofessional	1.546	michigan	-1.384
notch	1.539	he	-1.358
experience	1.535	corner	-1.339
variety	1.533	use	-1.334
looking	1.515	river	-1.333
food	1.504	if	-1.321
amazing	1.492	park	-1.318
maintained	1.485	quiet	-1.317
atmosphere	1.484	parking	-1.290

Table 5.2 – The 20 most discriminating words for each category (GPT-3 dataset) with their log-probability difference (delta).

Generated		**Authentic**	
Word	**delta**	**Word**	**delta**
future	2.134	new languages	-1.282
conclusion	2.121	<BIAS>	-1.174
model	2.021	introduction	-1.007
results	1.936	natural language	-0.897
work	1.533	natural	-0.876
future work	1.410	languages data	-0.855
new language	1.397	term	-0.833
topic	1.333	original	-0.778
addition	1.302	autoregressive	-0.767
sentiment	1.253	achieved	-0.764
relationship	1.251	model model	-0.745
framework	1.167	rosales nunez	-0.724
interesting	1.146	nunez et	-0.724
module	1.137	nunez	-0.724
hope	1.118	transformer model	-0.708
shot	1.113	summaries	-0.707
performance	1.110	sentence	-0.707
better	1.105	computational costs	-0.705
approach	1.059	dialog	-0.692
models	1.059	qa	-0.692

Table 5.3 – The 20 most discriminating words for each category (Fine-tuned GPT-2 generated academic dataset) with their log-probability difference (delta).

detectable, the task becomes more challenging for language models when replicating texts with a more intricate structure, such as academic content.

To substantiate the significance of vocabulary overlap in detection, we conducted an experiment involving the manipulation of the training and test data proportions. It's noteworthy that in practical scenarios, training data is typically unbalanced, given that annotated corpora represent only a fraction of the vast number of reviews available on platforms. Our study comprised 10 experiments for each variation in the ratio of test to training data. The outcomes, delineating recall and precision for each class, are illustrated in Figure 5.4.

The observed trend reveals a noteworthy pattern in the precision and recall metrics. Specifically, the precision for non-generated texts consistently appears to be lower compared to that for generated texts, while the recall exhibits the opposite trend. This pattern suggests the prevalence of false negative examples, signifying instances where the model is

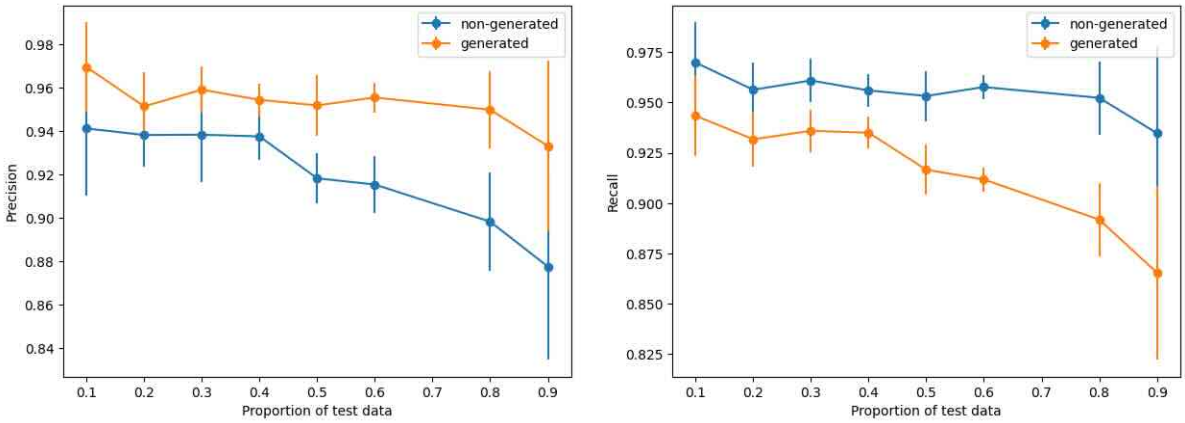

Figure 5.4 – Precision and recall for each class on the GPT3 dataset vs. original reviews, varying the proportion of test and training data. The error bar indicates the standard deviation calculated over 10 experiments.

inclined to misclassify machine-generated text as human-written, particularly when training data is limited. This phenomenon aligns with findings reported by [Wan+24], who similarly noted analogous challenges in experiments related to cross-domain and language classification. The implication is that the model's performance, particularly in identifying non-generated texts, may be impacted by the scarcity of diverse training data, emphasizing the need for robust and extensive datasets for more accurate and reliable classification outcomes.

Considering the inclusion of the paraphrased corpus, it is noteworthy that both precision and recall maintain relatively high levels. However, a discernible sensitivity in the accuracy of detection becomes apparent with respect to the availability of training data, as depicted in Figure 5.5. This suggests that the task of detecting fake reviews becomes subtly more challenging when faced with insufficient training data, particularly in the context of paraphrased content. The reliance on paraphrasing introduces an additional layer of complexity, reinforcing the importance of ample and diverse training datasets for enhancing the robustness and accuracy of fake review detection models.

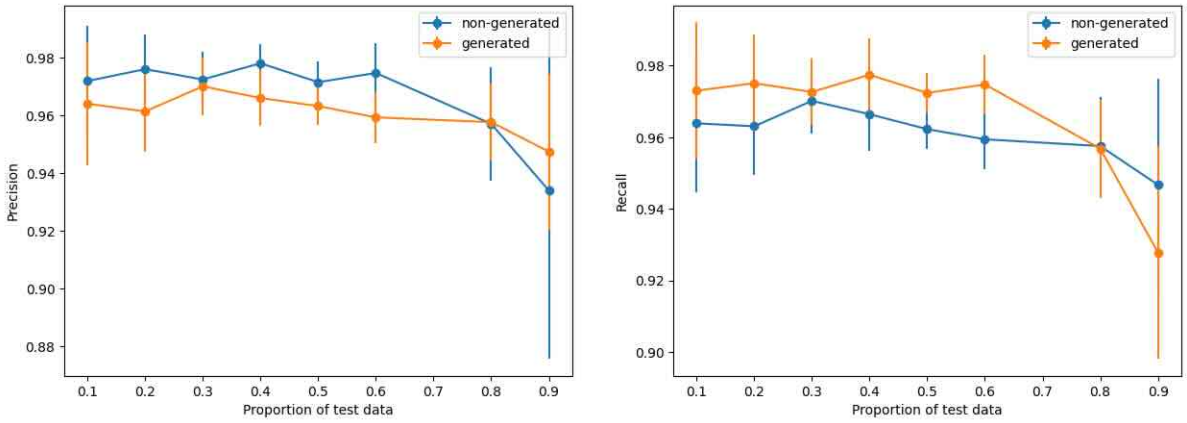

Figure 5.5 – Precision and recall for each class on the GPT3 paraphrased dataset vs. the original reviews, varying the proportion of test and training data.

Chapter 6

Concluding Remarks & Future Perspectives

The central focus of this thesis lies in addressing the research problem associated with the detection of automatically generated academic content. At the inception of our investigation, we encountered a notable gap in available datasets specifically tailored for machine-generated text. To lay the foundation for our work, we took the initiative to construct a benchmark corpus, leveraging cutting-edge generation models. This comprehensive dataset comprised two distinct corpora: one featuring fully generated articles produced through the fine-tuned GPT-2 model, and the other adopting a hybrid approach. In the hybrid corpus, select sentences from original abstracts were replaced with sentences generated by the arxiv-NLP model. To gauge the quality of our dataset, we conducted thorough evaluations utilizing n-gram-based measures such as BLEU and ROUGE.

Subsequently, we approached the detection task as a binary classification problem, treating the identification of machine-generated content as the primary objective. Employing a variety of classification models, we systematically evaluated the detectability of our generated corpus. This involved a comprehensive exploration of precision, recall, and other relevant metrics to ascertain the performance and robustness of our detection methods. This iterative process aimed at not only addressing the initial absence of suitable datasets but also at establishing a rigorous foundation for the subsequent phases of our research.

Upon scrutinizing the fully generated article dataset, it became evident that the generated content exhibited a tendency to replicate material from the original articles used during the fine-tuning process. Motivated by a desire to elevate the level of creativity in our dataset, we embarked on the creation of a novel set of fully generated articles. Employing the same approach as the one utilized for constructing the aforementioned

dataset, we introduced a crucial modification by incorporating a higher model temperature. This adjustment aimed to inject a greater degree of variability and originality into the generated content. Consequently, the resulting dataset reflected a heightened level of creativity in comparison to its predecessor. However, this increase in creativity came at the expense of detectability, as the generated articles displayed lower similarity to the original human-written articles. This delicate balance between creativity and mimicry underscored the nuanced challenges in crafting machine-generated content that not only demonstrates innovation but also aligns with the characteristics of authentic human writing.

Over the course of time, it became apparent that several fellow researchers had also undertaken the creation of corpora containing machine-generated academic content. Intriguingly, these researchers adopted diverse approaches in constructing their datasets. In response to this evolving landscape, we embarked on a comparative analysis to evaluate the detectability of these alternative datasets in contrast to our own. Utilizing the same classification models employed in our prior experiments, we scrutinized the results and noted that the datasets produced by other researchers exhibited a higher susceptibility to detection. Keen to comprehend the factors contributing to this disparity, we hypothesized that the approach used in building these corpora might play a pivotal role. To investigate this, we designed and constructed another dataset using a similar methodology, relying solely on a pre-trained generation model without fine-tuning. Subsequently, we subjected this new dataset to experimentation to assess its detectability. Confirming our conjecture, the results illustrated that academic content generated using a pre-trained model (without fine-tuning) was more readily detectable compared to content generated through the fine-tuning approach. This nuanced exploration underscored the importance of the underlying methodology in shaping the detectability characteristics of machine-generated academic content.

In our pursuit of a deeper comprehension regarding the challenges associated with detecting fake academic content in contrast to other content genres, we ventured into constructing a novel dataset focused on hotel reviews. Employing the same fine-tuned and pre-trained models utilized in our prior generation tasks, we generated synthetic hotel reviews. The subsequent evaluation of the detectability of this corpus, employing our classification models, unveiled intriguing insights. It emerged that the hotel review data exhibited a comparatively higher susceptibility to detection when compared to academic content. Notably, the analysis of the generated content's style alone proved effective in distinguishing fake hotel reviews from their authentic counterparts, showcasing a discernible contrast with the complexities inherent in distinguishing between genuine and machine-

generated academic text. This comparative exploration underscored the nuanced nature of detecting fakery across distinct content domains, shedding light on the unique challenges posed by academic content.

Given the introduction of various cutting-edge detection tools purported to excel in distinguishing machine-generated text, we decided to assess their efficacy in gauging the detectability of our corpora. However, to our astonishment, the results revealed that these state-of-the-art models were unable to accurately identify machine-generated content within our datasets. This unexpected outcome prompted a deeper investigation into the inherent characteristics and nuances of our generated content, highlighting potential areas for improvement and suggesting that the performance of these advanced tools might be contingent on specific contextual factors or dataset intricacies.

To this point, our classification efforts had relied solely on existing state-of-the-art models. Recognizing the potential for enhanced accuracy in terms of detectability, we endeavored to elevate our approach. To achieve this objective, we delved into the creation of several ensemble architectures, amalgamating diverse model components. Among these varied architectures, those fusing the transformer architecture with Convolutional Neural Network (CNN) layers emerged as particularly promising, consistently yielding the highest results in terms of classification accuracy. This strategic integration of different neural network elements aimed to harness the complementary strengths of each, fostering a more robust and nuanced approach to the intricate task of detecting machine-generated content.

Our research yields a dual set of key conclusions. Firstly, discerning automatically generated academic content proves to be a formidable challenge, particularly when juxtaposed with the detection of other types of generated content. This underscores the unique intricacies and subtleties embedded in machine-generated academic text, demanding specialized methodologies for effective identification.

Secondly, a notable finding emerges when considering the manner in which academic content is generated. Specifically, when academic content is crafted in a manner aligning with how a typical human might utilize generation models, such as through fine-tuning, the content becomes inherently more challenging to detect. This complexity contrasts with the detection of content prevalent in many existing datasets, where the generation process relies on a more straightforward utilization of pre-existing pre-trained models. The nuanced interplay between the intricacy of the generation approach and the detectability of the content highlights the need for tailored detection strategies, especially in the context of academic text generation.

Thus, through this research we could deliver the following main contributions,

1. Compilation of diverse corpora consisting of automatically generated academic

content.

2. Recognition of detection as a binary classification task, employing state-of-the-art models for this purpose.
3. Evaluation of the classification models' performance in terms of their ability to detect machine-generated academic content.
4. Development of multiple ensemble architectures to enhance detection capabilities.

Last but not least, we aspire to enhance the efficacy of our research by incorporating additional knowledge into the detection of academic content. Recognizing the dynamic nature of academic discourse and the evolving strategies employed by content manipulators, our ongoing efforts include exploring interdisciplinary collaborations. Integrating expertise from fields such as information retrieval, citation analysis, and domain-specific nuances can offer a nuanced perspective on distinguishing authentic academic content from potentially deceptive practices. Embracing advancements in machine learning techniques tailored for academic integrity, alongside staying abreast of emerging scholarly communication trends, is crucial. Through a holistic approach that amalgamates domain expertise and cutting-edge technologies, we aim to fortify the robustness of our detection methods and contribute to the ongoing efforts to maintain the integrity of academic discourse.

Chapter 7

List of Publications

Our work has resulted in five research articles and an extended abstract:

1. Vijni Liyanage, Davide Buscaldi, Adeline Nazerenko, **"A benchmark corpus for the detection of automatically generated text in academic publications"**, In proceedings of LREC 2022

   In this article, we thoroughly outline the steps taken to create our benchmark dataset. Additionally, we delve into the specifics of the experiments conducted to assess both the quality and the detectability level of the dataset.

2. Vijni Liyanage, Davide Buscaldi, **"Detecting Artificially Generated Academic Text: The Importance of Mimicking Human Utilization of Large Language Models"**, In proceedings of NLDB 2023

   In this article, we assess how difficult is to detect our corpora compared to several similar ones. Despite the high classification accuracies reported by other corpora, this article emphasizes the significance of constructing corpora in a way that mirrors how humans would use generation models.

3. Vijni Liyanage, Davide Buscaldi, **"La détection de textes générés par des modèles de langue: une tâche complexe? Une étude sur des textes académiques"**, In proceedings of CORIA TALN RJCRI RECITAL 2023

   This article explains the challenges and complexities faced in detecting academic content.

4. Vijini Liyanage, Davide Buscaldi, **"An Ensemble Method Based on the Combination of Transformers with Convolutional Neural Networks to Detect Artificially Generated Text"**, In proceedings of ALTA 2023

This research paper explains our approaches followed in creating ensemble architectures for the classification task. The results produced by the ensemble architectures prove that they perform better than the corresponding standalone models regarding the detection task.

5. Vijini Liyanage, Davide Buscaldi, Pénelope Forcioli **"Are AI-enhanced Opinion Spambots Worrisome?**
A study on GPT-generated Hotel Reviews", ECNLP at LREC-COLING 2024

   This article compares detectability of generated hotel reviews against the detectability of generated academic contents.

Extended Abstract Vijini Liyanage, **"Is it an Easy Task to Accurately Detect Automatically Generated Academic Content?"**, WiNLP-EMNLP 2023

Bibliography

[AGM+19] E. Abd-Elaal, S. Gamage, J. Mills, et al. "Artificial intelligence is a tool for cheating academic integrity". In: *30th Annual Conference for the Australasian Association for Engineering Education (AAEE 2019): Educators Becoming Agents of Change: Innovate, Integrate, Motivate*. Engineers Australia. 2019, p. 397.

[Ach+23] O. J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, et al. *GPT-4 Technical Report*. 2023.

[Ade+20] D. Adelani, H. Mai, F. Fang, H. Nguyen, J. Yamagishi, and I. Echizen. "Generating sentiment-preserving fake online reviews using neural language models and their human-and machine-based detection". In: *International Conference on Advanced Information Networking and Applications*. Springer. 2020, pp. 1341–1354.

[Ama15] D. R. Amancio. "Comparing the topological properties of real and artificially generated scientific manuscripts". In: *Scientometrics* 105.3 (2015), pp. 1763–1779.

[Ant+23] W. Antoun, V. Mouilleron, B. Sagot, and D. Seddah. *Towards a Robust Detection of Language Model-Generated Text: Is ChatGPT that easy to detect?* 2023.

[Bak+19] A. Bakhtin, S. Gross, M. Ott, Y. Deng, M. Ranzato, and A. Szlam. "Real or Fake? Learning to Discriminate Machine from Human Generated Text". In: *ArXiv* abs/1906.03351 (2019).

[BLC19] I. Beltagy, K. Lo, and A. Cohan. "SciBERT: A Pretrained Language Model for Scientific Text". In: (2019), pp. 3615–3620.

[Bha23] A. Bhat. *GPT-wiki-intro (Revision 0e458f5)*. 2023.

[BP20] M. M. Bhat and S. Parthasarathy. "How Effectively Can Machines Defend Against Machine-Generated Fake News? An Empirical Study". In: *Proceedings of the First Workshop on Insights from Negative Results in NLP*. 2020, pp. 48–53.

[Boj+17] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching word vectors with subword information". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146.

[CL21] G. Cabanac and C. Labbé. "Prevalence of nonsensical algorithmically generated papers in the scientific literature". In: *Journal of the Association for Information Science and Technology* 72.12 (2021), pp. 1461–1476.

[CLM21] G. Cabanac, C. Labbé, and A. Magazinov. "Tortured phrases: A dubious writing style emerging in science. Evidence of critical issues affecting established journals". In: *arXiv preprint arXiv:2107.06751* (2021).

[ÇB20] E. Çano and O. Bojar. "Human or Machine: Automating Human Likeliness Evaluation of NLG Texts". In: *ArXiv* abs/2006.03189 (2020).

[Cla+21] E. Clark, T. August, S. Serrano, N. Haduong, S. Gururangan, and N. A. Smith. "All That's 'Human'Is Not Gold: Evaluating Human Evaluation of Generated Text". In: (2021), pp. 7282–7296.

[Cla+16] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. "ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS". In: *ELECTRA* 85 (2016), p. 90.

[CL19] A. Conneau and G. Lample. "Cross-lingual language model pretraining". In: vol. 32. 2019.

[CV95] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[Dai+19] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov. "Transformer-xl: Attentive language models beyond a fixed-length context." In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 2978–2988.

[Dug+20] L. Dugan, D. Ippolito, A. Kirubarajan, and C. Callison-Burch. "RoFT: A Tool for Evaluating Human Detection of Machine-Generated Text". In: (2020), pp. 189–196.

[Fag+21] T. Fagni, F. Falchi, M. Gambini, A. Martella, and M. Tesconi. "TweepFake: About detecting deepfake tweets". In: *Plos one* 16.5 (2021), e0251415.

[Far+17] R. Faris, H. Roberts, B. Etling, N. Bourassa, E. Zuckerman, and Y. Benkler. "Partisanship, propaganda, and disinformation: Online media and the 2016 US presidential election". In: *Berkman Klein Center Research Publication* 6 (2017).

[GSR19] S. Gehrmann, H. Strobelt, and A. M. Rush. "GLTR: Statistical Detection and Visualization of Generated Text". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2019, pp. 111–116.

[GG22] A. Glazkova and M. Glazkov. "Detecting generated scientific papers using an ensemble of transformer models". In: *Proceedings of the Third Workshop on Scholarly Document Processing*. 2022, pp. 223–228.

[GB+14] E. Grefenstette, P. Blunsom, et al. "A convolutional neural network for modelling sentences". In: *The 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, Maryland*. Vol. 1. 2014, pp. 655–665.

[HBC21] A. Harada, D. Bollegala, and N. P. Chandrasiri. "Discrimination of human-written and human and machine written sentences using text consistency". In: *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*. IEEE. 2021, pp. 41–47.

[Har54] Z. S. Harris. "Distributional structure". In: *Word* 10.2-3 (1954), pp. 146–162.

[He+20] P. He, X. Liu, J. Gao, and W. Chen. "DEBERTA: DECODING-ENHANCED BERT WITH DISENTANGLED ATTENTION". In: (2020).

[Ipp+19] D. Ippolito, D. Duckworth, C. Callison-Burch, and D. Eck. "Human and automatic detection of generated text". In: *arXiv preprint arXiv:1911.00650* (2019).

[Ipp+20] D. Ippolito, D. Duckworth, C. Callison-Burch, and D. Eck. "Automatic Detection of Generated Text is Easiest when Humans are Fooled". In: (2020), pp. 1808–1822.

[Jaw] G. Jawahar. "Detecting human written text from machine generated text by modeling discourse coherence". In: ().

[JML20] G. Jawahar, M. A. Mageed, and V. Laks Lakshmanan. "Automatic Detection of Machine Generated Text: A Critical Survey". In: (2020), pp. 2296–2309.

[AL20] S. Al-Kadhimi and P. Löwenström. *Identification of machine-generated reviews: 1D CNN applied on the GPT-2 neural language model*. 2020.

[Kas+22] Y. Kashnitsky, D. Herrmannova, A. de Waard, G. Tsatsaronis, C. Fennell, and C. Labbé. "Overview of the DAGPap22 shared task on detecting automatically generated scientific papers". In: *Third Workshop on Scholarly Document Processing*. 2022.

[KT19] J. D. M.-W. C. Kenton and L. K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *Proceedings of naacL-HLT*. Vol. 1. 2019, p. 2.

[Kim14] Y. Kim. "Convolutional Neural Networks for Sentence Classification". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by A. Moschitti, B. Pang, and W. Daelemans. Doha, Qatar: Association for Computational Linguistics, Aug. 2014, pp. 1746–1751.

[LK10] A. Lavoie and M. Krishnamoorthy. "Algorithmic Detection of Computer Generated Text". In: *stat* 1050 (2010), p. 4.

[Lew+20] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7871–7880.

[Lin04] C.-Y. Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[Liu+19] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, et al. "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *ArXiv* abs/1907.11692 (2019).

[MSS21] A. Maronikolakis, H. Schütze, and M. Stevenson. "Identifying Automatically Generated Headlines using Transformers". In: (2021), pp. 1–6.

[Mit+23] E. Mitchell, Y. Lee, A. Khazatsky, C. D. Manning, and C. Finn. "Detectgpt: Zero-shot machine-generated text detection using probability curvature". In: (2023), pp. 24950–24962.

[NL16] M. T. Nguyen and C. Labbé. "Engineering a tool to detect automatically generated papers". In: *BIR 2016 Bibliometric-enhanced Information Retrieval*. 2016.

[OCH13] M. Ott, C. Cardie, and J. T. Hancock. "Negative Deceptive Opinion Spam". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 497–501.

[Ott+11] M. Ott, Y. Choi, C. Cardie, and J. T. Hancock. "Finding Deceptive Opinion Spam by Any Stretch of the Imagination". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 309–319.

[Pap+02] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[Pér+18] V. Pérez-Rosas, B. Kleinberg, A. Lefevre, and R. Mihalcea. "Automatic Detection of Fake News". In: (2018), pp. 3391–3401.

[Rad+18] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. "Improving language understanding by generative pre-training". In: OpenAI, 2018.

[Rad+19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. "Language models are unsupervised multitask learners". In: vol. 1. 8. 2019, p. 9.

[Rod+22] J. Rodriguez, T. Hay, D. Gros, Z. Shamsi, and R. Srinivasan. "Cross-Domain Detection of GPT-2-Generated Technical Text". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2022, pp. 1213–1233.

[Ros22] D. Rosati. "SynSciPass: detecting appropriate uses of scientific text generation". In: *Proceedings of the Third Workshop on Scholarly Document Processing*. 2022, pp. 214–222.

[Sad+23] V. S. Sadasivan, A. Kumar, S. Balasubramanian, W. Wang, and S. Feizi. *Can AI-Generated Text be Reliably Detected?* 2023.

[San+19] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *ArXiv* abs/1910.01108 (2019).

[Sch+19] T. Schuster, R. Schuster, D. J. Shah, and R. Barzilay. "Are We Safe Yet? The Limitations of Distributional Features for Fake News Detection". In: *ArXiv* abs/1908.09805 (2019).

[Sch+20] T. Schuster, R. Schuster, D. J. Shah, and R. Barzilay. "The limitations of stylometry for detecting machine-generated fake news". In: *Computational Linguistics* 46.2 (2020), pp. 499–510.

[Sol+19] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, et al. "Release Strategies and the Social Impacts of Language Models". In: *ArXiv* abs/1908.09203 (2019).

[TC23] E. Tian and A. Cui. *GPTZero: Towards detection of AI-generated text using zero-shot and supervised methods*. 2023.

[Uch+20] A. Uchendu, T. Le, K. Shu, and D. Lee. "Authorship attribution for neural text generation". In: *Conf. on Empirical Methods in Natural Language Processing (EMNLP)*. 2020.

[VKS20] L. R. Varshney, N. S. Keskar, and R. Socher. "Limits of detecting text generated by large-scale language models". In: *2020 Information Theory and Applications Workshop (ITA)*. IEEE. 2020, pp. 1–5.

[Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[Vig19] J. Vig. "A Multiscale Visualization of Attention in the Transformer Model". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 37–42.

[Vij+20] S. Vijayaraghavan, Y. Wang, Z. Guo, J. Voong, W. Xu, A. Nasseri, et al. "Fake news detection with different models". In: *arXiv preprint arXiv:2003.04978* (2020).

[VRA18] S. Vosoughi, D. Roy, and S. Aral. "The spread of true and false news online". In: *Science* 359.6380 (2018), pp. 1146–1151.

[Wan+19] Q. Wang, L. Huang, Z. Jiang, K. Knight, H. Ji, M. Bansal, et al. "PaperRobot: Incremental Draft Generation of Scientific Ideas". In: (2019), pp. 1980–1991.

[Wan+20] Q. Wang, Q. Zeng, L. Huang, K. Knight, H. Ji, and N. F. Rajani. "ReviewRobot: Explainable paper review generation based on knowledge synthesis". In: (2020), pp. 384–397.

[Wan+24] Y. Wang, J. Mansurov, P. Ivanov, J. Su, A. Shelmanov, A. Tsvigun, et al. *M4: Multi-generator, Multi-domain, and Multi-lingual Black-Box Machine-Generated Text Detection*. 2024.

[WD17] C. Wardle and H. Derakhshan. "Information disorder: Toward an interdisciplinary framework for research and policy making". In: *Council of Europe* 27 (2017).

[Wei83] J. Weizenbaum. "ELIZA — a Computer Program for the Study of Natural Language Communication between Man and Machine". In: *Commun. ACM* 26.1 (Jan. 1983), pp. 23–28.

[WG15] K. Williams and C. L. Giles. "On the use of similarity search to detect fake scientific papers". In: *International Conference on Similarity Search and Applications*. Springer. 2015, pp. 332–338.

[Wol+20a] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[Wol+20b] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, et al. "Transformers: State-of-the-art natural language processing". In: *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*. 2020, pp. 38–45.

[Wol20] M. Wolff. "Attacking Neural Text Detectors". In: *ArXiv* abs/2002.11768 (2020).

[WF23] H. Wu and T. Flanagan. "The Limits of AI Content Detectors". In: *Journal of Student Research* 12.3 (2023).

[XH09] J. Xiong and T. Huang. "An effective method to identify machine automatically generated paper". In: *2009 Pacific-Asia Conference on Knowledge Engineering and Software Engineering*. IEEE. 2009, pp. 101–102.

[Yan+19] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. "Xlnet: Generalized autoregressive pretraining for language understanding". In: vol. 32. 2019.

[Zel+19] R. Zellers, A. Holtzman, H. Rashkin, Y. Bisk, A. Farhadi, F. Roesner, et al. "Defending against neural fake news". In: *Advances in neural information processing systems* 32 (2019).

[Zho+20] W. Zhong, D. Tang, Z. Xu, R. Wang, N. Duan, M. Zhou, et al. “Neural Deepfake Detection with Factual Structure of Text”. In: (2020), pp. 2461–2470.

Appendices

Appendix A

Code Segments

A.1 Code to Generate Text by Fine-tuning GPT-2 Model

```
!pip install -q gpt-2-simple
import gpt_2_simple as gpt2
from datetime import datetime
from google.colab import files

!nvidia-smi

gpt2.download_gpt2(model_name="124M")

from google.colab import drive
drive.mount('/gdrive', force_remount=True)
from google.colab import drive
drive.mount('/content/drive')
gpt2.mount_gdrive()

file_name = 'N01-1009.txt'
gpt2.copy_file_from_gdrive(file_name)

#get the length of the dataset and the first k =50 words
file = open(file_name, encoding='latin1')
data = file.read()
```

```
words = data.split()

print('Number of words in text file :', len(words))
substring = data.split()[0:50]
#print(substring)
seed = ' '.join(substring)
print(seed)

#fine-tuning
sess = gpt2.start_tf_sess()

gpt2.finetune(sess,
              dataset=file_name,
              model_name='124M',
              steps=1000,
              restore_from='fresh',
              run_name='run1',
              print_every=10,
              sample_every=1000,
              save_every=500
              )

gpt2.copy_checkpoint_to_gdrive(run_name='run1')

String_final=''
output_seed = seed

while True:
  with open('Output_final.txt', 'a') as f:
    f.write(String_final)
    f.close()

  gen_file = 'Output.txt'
  gpt2.generate_to_file(sess,
                        destination_path=gen_file,
                        length=1023,
```

```
                    temperature=0.7,
                    prefix=output_seed,
                    nsamples=1,
                    batch_size=1
                    )

  output_file_name = 'Output.txt'
  output_file = open(output_file_name, encoding='latin1')
  output_data = output_file.read()
  output_words = output_data.split()
  print('Number of words in output file :', len(output_words))
  output_substring = output_words[-51:-1]
  output_seed = ' '.join(output_substring)
  #print(output_seed)
  final_substring = output_words[-len(output_words):-52]
  String_final = ' '.join(final_substring)
  #print(String_final)

  output_final_file = open('Output_final.txt', encoding='latin1')
  output_final_data = output_final_file.read()
  output_final_words = output_final_data.split()
  print('Number of words in final output file :', len(output_final_words))

  if (len(output_final_words) > len(words) ):
    break


#code to make the fake file length equal to original file length
difference = len(output_final_words) - len(words)
#print(difference)
first = len(output_final_words)
last = difference
Final_fake_text = output_final_words[-first:-last]
Final_fake_text_for_the_fake_file = ' '.join(Final_fake_text)
print(Final_fake_text_for_the_fake_file)
with open('Final_fake_file.txt', 'w') as f:
```

```
  f.write(Final_fake_text_for_the_fake_file)
  f.close()


files.download(gen_file)
```

A.2 Code to Generate Text by Pre-trained GPT-2 model

```
!pip install transformers
from transformers import GPT2LMHeadModel , GPT2Tokenizer

tokenizer = GPT2Tokenizer.from_pretrained('gpt2-medium')
model = GPT2LMHeadModel.from_pretrained('gpt2-medium' ,
pad_token_id = tokenizer.eos_token_id)

seed = " This allows the entity pair to automatically generate labels to train deep-lear
input_ids = tokenizer.encode(seed, return_tensors = 'pt')

output = model.generate(input_ids,
max_length = 250,
num_beams = 5,
no_repeat_ngram_size  = 2,
early_stopping = True)

generated= tokenizer.decode(output[0])
print(generated)
```

A.3 Code to Classify Text Using Transformer-based Models

```
import os
import math
import random
import csv
import sys

import numpy as np
import pandas as pd
from sklearn import metrics
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.metrics import classification_report
import statistics as stats

df = pd.read_csv('Hotel_Reviews_Dataset_GPT3_paraPhrased.csv')
df.head()

from sklearn.model_selection import train_test_split
# split the data into train and test set since there is no separate test set yet
train_df, test_df = train_test_split(df, test_size=0.4, random_state=42, shuffle=True)

X = train_df['text']
y = train_df['label']
train_df.shape

!pip install simpletransformers
from simpletransformers.classification import ClassificationModel, ClassificationArgs
import pandas as pd
import logging
from sklearn.metrics import f1_score, recall_score, precision_score

model_args = ClassificationArgs(num_train_epochs=3, overwrite_output_dir=True, \
                                    no_save = True, max_seq_length=128, sliding_w
```

```
model = ClassificationModel(
            'bert', 'bert-base-uncased', args=model_args, use_cuda=True
            #"bert", "allenai/scibert_scivocab_cased", args=model_args, use_cuda=True
            #"roberta", "roberta-large", args=model_args, use_cuda=True#, weight = list(
            #"deberta", "microsoft/deberta-large", args=model_args, use_cuda=True
            #ELECTRA_base
            model = ClassificationModel('electra', 'google/electra-base-discriminator',
            #XLNet
            model = ClassificationModel('xlnet', 'xlnet-base-cased', args=model_args, us
            )

model.train_model(df)
import logging
from sklearn.metrics import f1_score, recall_score, precision_score

test_df.head()
X_test = test_df['text']
y_test = test_df['label']
test_df.shape

predictions, raw_outputs = model.predict(list(X_test))
f1 = f1_score(predictions, y_test, average = 'macro')
print(f1)
```

A.4 Code to Produce Ensemble Architectures

```
!pip install transformers
!pip install optuna
import torch
import torch.nn as nn
import torch.nn.functional as F
from transformers import get_linear_schedule_with_warmup, AdamW
from torch.utils.data import TensorDataset, random_split, DataLoader, RandomSampler, S
import time, datetime, random, optuna, re, string
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
from optuna.pruners import SuccessiveHalvingPruner
from optuna.samplers import TPESampler
from torch.cuda.amp import autocast, GradScaler
from sklearn.model_selection import train_test_split
from collections import Counter
from transformers import BertModel, BertTokenizer

SEED = 15
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.backends.cudnn.deterministic = True
torch.cuda.amp.autocast(enabled=True)

device = torch.device("cuda")
df = pd.read_csv('CV_ArxivDataset_trainSet_FULL_edited.csv')
df.head()

def clean_df(df):
    # strip dash but keep a space
    df['text'] = df['text'].str.replace('-', ' ')
```

```
    # prepare keys for punctuation removal
    translator = str.maketrans(dict.fromkeys(string.punctuation))
    # lower case the data
    df['text'] = df['text'].apply(lambda x: x.lower())
    # remove excess spaces near punctuation
    df['text'] = df['text'].apply(lambda x: re.sub(r'\s([?.!"](?:\s|$))', r'\1', x))
    # remove punctuation  -- f1 improves by .05 by disabling this
    #df['body'] = df['body'].apply(lambda x: x.translate(translator))
    # generate a word count
    df['word_count'] = df['text'].apply(lambda x: len(x.split()))
    # remove excess white spaces
    df['text'] = df['text'].apply(lambda x: " ".join(x.split()))

    return df


df = clean_df(df)
# instantiate BERT tokenizer with upper + lower case
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

word_map = dict(zip(tokenizer.vocab.keys(), range(len(tokenizer))))
word_map.get('the')  # find index value

list(tokenizer.vocab.keys())[2000:2010]
len(tokenizer)

# tokenize corpus using BERT
def tokenize_corpus(df, tokenizer, max_len):
    # token ID storage
    input_ids = []
    # attension mask storage
    attention_masks = []
    # max len -- 512 is max
    max_len = max_len
    # for every document:
    for doc in df:
```

```
        # ‘encode_plus‘ will:
        #   (1) Tokenize the sentence.
        #   (2) Prepend the ‘[CLS]‘ token to the start.
        #   (3) Append the ‘[SEP]‘ token to the end.
        #   (4) Map tokens to their IDs.
        #   (5) Pad or truncate the sentence to ‘max_length‘
        #   (6) Create attention masks for [PAD] tokens.
        encoded_dict = tokenizer.encode_plus(
                            doc,  # document to encode.
                            add_special_tokens=True,  # add '[CLS]' and '[SEP]'
                            max_length=max_len,  # set max length
                            truncation=True,  # truncate longer messages
                            pad_to_max_length=True,  # add padding
                            return_attention_mask=True,  # create attn. masks
                            return_tensors='pt'  # return pytorch tensors
                       )

        # add the tokenized sentence to the list
        input_ids.append(encoded_dict['input_ids'])

        # and its attention mask (differentiates padding from non-padding)
        attention_masks.append(encoded_dict['attention_mask'])

    return torch.cat(input_ids, dim=0), torch.cat(attention_masks, dim=0)


# create tokenized data
input_ids, attention_masks = tokenize_corpus(df['text'].values, tokenizer, 512)

# convert the labels into tensors.
labels = torch.tensor(df['#label'].values.astype(np.float32))

import math
from torch import default_generator, randperm
from torch._utils import _accumulate
from torch.utils.data.dataset import Subset
```

```
def random_split(dataset, lengths,
                 generator=default_generator):
    r"""
    Randomly split a dataset into non-overlapping new datasets of given lengths.

    If a list of fractions that sum up to 1 is given,
    the lengths will be computed automatically as
    floor(frac * len(dataset)) for each fraction provided.

    After computing the lengths, if there are any remainders, 1 count will be
    distributed in round-robin fashion to the lengths
    until there are no remainders left.

    Optionally fix the generator for reproducible results, e.g.:

    >>> random_split(range(10), [3, 7], generator=torch.Generator().manual_seed(42))
    >>> random_split(range(30), [0.3, 0.3, 0.4], generator=torch.Generator(
    ...   ).manual_seed(42))

    Args:
        dataset (Dataset): Dataset to be split
        lengths (sequence): lengths or fractions of splits to be produced
        generator (Generator): Generator used for the random permutation.
    """
    if math.isclose(sum(lengths), 1) and sum(lengths) <= 1:
        subset_lengths: List[int] = []
        for i, frac in enumerate(lengths):
            if frac < 0 or frac > 1:
                raise ValueError(f"Fraction at index {i} is not between 0 and 1")
            n_items_in_split = int(
                math.floor(len(dataset) * frac)  # type: ignore[arg-type]
            )
            subset_lengths.append(n_items_in_split)
        remainder = len(dataset) - sum(subset_lengths)  # type: ignore[arg-type]
        # add 1 to all the lengths in round-robin fashion until the remainder is 0
```

```
        for i in range(remainder):
            idx_to_add_at = i % len(subset_lengths)
            subset_lengths[idx_to_add_at] += 1
        lengths = subset_lengths
        for i, length in enumerate(lengths):
            if length == 0:
                warnings.warn(f"Length of split at index {i} is 0. "
                              f"This might result in an empty dataset.")

    # Cannot verify that dataset is Sized
    if sum(lengths) != len(dataset):    # type: ignore[arg-type]
        raise ValueError("Sum of input lengths does not equal the length of the input

    indices = randperm(sum(lengths), generator=generator).tolist()  # type: ignore[ca
    return [Subset(dataset, indices[offset - length : offset]) for offset, length in z



# prepare tensor data sets
def prepare_dataset(padded_tokens, attention_masks, target):
    # prepare target into np array
    target = np.array(target.values, dtype=np.int64).reshape(-1, 1)
    # create tensor data sets
    tensor_df = TensorDataset(padded_tokens, attention_masks, torch.from_numpy(target)
    # 80% of df
    train_size = int(0.8 * len(df))
    # 20% of df
    val_size = len(df) - train_size
    # 50% of validation
    test_size = int(val_size - 0.5*val_size)
    # divide the dataset by randomly selecting samples
    train_dataset, val_dataset = random_split(tensor_df, [train_size, val_size])
    # divide validation by randomly selecting samples

    new_size = int(0.1 * len(df))
```

```
    val_dataset, test_dataset = random_split(val_dataset, [new_size, new_size])
    #torch.utils.data.random_split(dataset, [int(0.8 * len(dataset)), int(0.2 * len(data

    return train_dataset, val_dataset, test_dataset


# create tensor data sets
train_dataset, val_dataset, test_dataset = prepare_dataset(input_ids,
                                                           attention_masks,
                                                           df['#label'])

# helper function to count target distribution inside tensor data sets
def target_count(tensor_dataset):
    # set empty count containers
    count0 = 0
    count1 = 0
    # set total container to turn into torch tensor
    total = []
    # for every item in the tensor data set
    for i in tensor_dataset:
        # if the target is equal to 0
        if i[2].item() == 0:
            count0 += 1
        # if the target is equal to 1
        elif i[2].item() == 1:
            count1 += 1
    total.append(count0)
    total.append(count1)
    return torch.tensor(total)

# prepare weighted sampling for imbalanced classification
def create_sampler(target_tensor, tensor_dataset):
    # generate class distributions [x, y]
    class_sample_count = target_count(tensor_dataset)
    # weight
    weight = 1. / class_sample_count.float()
```

```
    # produce weights for each observation in the data set
    samples_weight = torch.tensor([weight[t[2]] for t in tensor_dataset])
    # prepare sampler
    sampler = torch.utils.data.WeightedRandomSampler(weights=samples_weight,
                                                     num_samples=len(samples_weight),
                                                     replacement=True)
    return sampler


# create samplers for just the training set
train_sampler = create_sampler(target_count(train_dataset), train_dataset)


# time function
def format_time(elapsed):
    '''
    Takes a time in seconds and returns a string hh:mm:ss
    '''
    # round to the nearest second.
    elapsed_rounded = int(round((elapsed)))
    # format as hh:mm:ss
    return str(datetime.timedelta(seconds=elapsed_rounded))


# create DataLoaders with samplers
train_dataloader = DataLoader(train_dataset,
                              batch_size=8,
                              sampler=train_sampler,
                              shuffle=False)

valid_dataloader = DataLoader(val_dataset,
                              batch_size=8,
                              shuffle=True)

test_dataloader = DataLoader(test_dataset,
                              batch_size=8,
```

```
                                shuffle=True)


def train(model, dataloader, optimizer):

    # capture time
    total_t0 = time.time()

    # Perform one full pass over the training set.
    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch + 1, epochs))
    print('Training...')

    # reset total loss for epoch
    train_total_loss = 0
    total_train_f1 = 0

    # put both models into traning mode
    model.train()
    kim_model.train()

    # for each batch of training data...
    for step, batch in enumerate(dataloader):

        # progress update every 40 batches.
        if step % 40 == 0 and not step == 0:

            # Report progress.
            print('  Batch {:>5,}  of  {:>5,}.'.format(step, len(dataloader)))

        # Unpack this training batch from our dataloader:
        #
        # As we unpack the batch, we'll also copy each tensor to the GPU
        #
        # `batch` contains three pytorch tensors:
        #   [0]: input ids
```

```
#   [1]: attention masks
#   [2]: labels
b_input_ids = batch[0].cuda()
b_input_mask = batch[1].cuda()
b_labels = batch[2].cuda().long()

# clear previously calculated gradients
optimizer.zero_grad()

# runs the forward pass with autocasting.
with autocast():
    # forward propagation (evaluate model on training batch)
    outputs = model(input_ids=b_input_ids, attention_mask=b_input_mask)

    hidden_layers = outputs[2]  # get hidden layers

    hidden_layers = torch.stack(hidden_layers, dim=1)  # stack the layers

    hidden_layers = hidden_layers[:, -4:]  # get the last 4 layers

logits = kim_model(hidden_layers)

loss = criterion(logits.view(-1, 2), b_labels.view(-1))

# sum the training loss over all batches for average loss at end
# loss is a tensor containing a single value
train_total_loss += loss.item()

# Scales loss. Calls backward() on scaled loss to create scaled gradients.
# Backward passes under autocast are not recommended.
# Backward ops run in the same dtype autocast chose for corresponding forward
scaler.scale(loss).backward()

# scaler.step() first unscales the gradients of the optimizer's assigned para
# If these gradients do not contain infs or NaNs, optimizer.step() is then ca
# otherwise, optimizer.step() is skipped.
```

```
        scaler.step(optimizer)

        # Updates the scale for next iteration.
        scaler.update()

        # Update the scheduler
        scheduler.step()

        # calculate preds
        _, predicted = torch.max(logits, 1)

        # move logits and labels to CPU
        predicted = predicted.detach().cpu().numpy()
        y_true = b_labels.detach().cpu().numpy()

        # calculate f1
        total_train_f1 += f1_score(predicted, y_true,
                                   average='weighted',
                                   labels=np.unique(predicted))

    # calculate the average loss over all of the batches
    avg_train_loss = train_total_loss / len(dataloader)

    # calculate the average f1 over all of the batches
    avg_train_f1 = total_train_f1 / len(dataloader)

    # training time end
    training_time = format_time(time.time() - total_t0)

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'Train Loss': avg_train_loss,
            'Train F1': avg_train_f1,
            'Train Time': training_time
        }
```

```
    )

    # print result summaries
    print("")
    print("summary results")
    print("epoch | trn loss | trn f1 | trn time ")
    print(f"{epoch+1:5d} | {avg_train_loss:.5f} | {avg_train_f1:.5f} | {training_time

    #torch.cuda.empty_cache()

    return None


def validating(model, dataloader):

    # capture validation time
    total_t0 = time.time()

    # After the completion of each training epoch, measure our performance on
    # our validation set.
    print("")
    print("Running Validation...")

    # put both models in evaluation mode
    model.eval()
    kim_model.eval()

    # track variables
    total_valid_accuracy = 0
    total_valid_loss = 0
    total_valid_f1 = 0
    total_valid_recall = 0
    total_valid_precision = 0
    total_bert_valid_loss = 0

    # evaluate data for one epoch
```

```
    for batch in dataloader:

        # Unpack this training batch from our dataloader:
        # ‘batch‘ contains three pytorch tensors:
        #   [0]: input ids
        #   [1]: attention masks
        #   [2]: labels
        b_input_ids = batch[0].cuda()
        b_input_mask = batch[1].cuda()
        b_labels = batch[2].cuda().long()

        # tell pytorch not to bother calculating gradients
        with torch.no_grad():
            # forward propagation (evaluate model on training batch)
            outputs = model(input_ids=b_input_ids, attention_mask=b_input_mask)

            hidden_layers = outputs[2]  # get hidden layers

            hidden_layers = torch.stack(hidden_layers, dim=1)  # stack the layers

            hidden_layers = hidden_layers[:, -4:]  # get the last 4 layers

        logits = kim_model(hidden_layers)

        loss = criterion(logits.view(-1, 2), b_labels.view(-1))

        # accumulate validation loss
        total_valid_loss += loss.item()

        # calculate preds
        _, predicted = torch.max(logits, 1)

        # move logits and labels to CPU
        predicted = predicted.detach().cpu().numpy()
        y_true = b_labels.detach().cpu().numpy()
```

```
        # calculate f1
        total_valid_f1 += f1_score(predicted, y_true,
                                   average='weighted',
                                   labels=np.unique(predicted))

        # calculate accuracy
        total_valid_accuracy += accuracy_score(predicted, y_true)

        # calculate precision
        total_valid_precision += precision_score(predicted, y_true,
                                                 average='weighted',
                                                 labels=np.unique(predicted))

        # calculate recall
        total_valid_recall += recall_score(predicted, y_true,
                                           average='weighted',
                                           labels=np.unique(predicted))

    # report final accuracy of validation run
    avg_accuracy = total_valid_accuracy / len(dataloader)

    # report final f1 of validation run
    global avg_val_f1
    avg_val_f1 = total_valid_f1 / len(dataloader)

    # report final f1 of validation run
    avg_precision = total_valid_precision / len(dataloader)

    # report final f1 of validation run
    avg_recall = total_valid_recall / len(dataloader)

    # calculate the average loss over all of the batches.
    global avg_val_loss
    avg_val_loss = total_valid_loss / len(dataloader)

    # capture end validation time
```

```
    training_time = format_time(time.time() - total_t0)

    # Record all statistics from this epoch.
    valid_stats.append(
        {
            'Val Loss': avg_val_loss,
            'Val Accur.': avg_accuracy,
            'Val precision': avg_precision,
            'Val recall': avg_recall,
            'Val F1': avg_val_f1,
            'Val Time': training_time
        }
    )

    # print result summaries
    print("")
    print("summary results")
    print("epoch | val loss | val f1 | val time")
    print(f"{epoch+1:5d} | {avg_val_loss:.5f} | {avg_val_f1:.5f} | {training_time:}")

    return None


def testing(model, dataloader):

    print("")
    print("Running Testing...")

    # capture test time
    total_t0 = time.time()

    # put both models in evaluation mode
    model.eval()
    kim_model.eval()

    # track variables
```

```
total_test_accuracy = 0
total_test_loss = 0
total_test_f1 = 0
total_test_recall = 0
total_test_precision = 0

# evaluate data for one epoch
for batch in dataloader:

    # Unpack this training batch from our dataloader:
    # ‘batch‘ contains three pytorch tensors:
    #   [0]: input ids
    #   [1]: attention masks
    #   [2]: labels
    b_input_ids = batch[0].cuda()
    b_input_mask = batch[1].cuda()
    b_labels = batch[2].cuda().long()

    # tell pytorch not to bother calculating gradients
    with torch.no_grad():
        # forward propagation (evaluate model on training batch)
        outputs = model(input_ids=b_input_ids, attention_mask=b_input_mask)

        hidden_layers = outputs[2]  # get hidden layers

        hidden_layers = torch.stack(hidden_layers, dim=1)  # stack the layers

        hidden_layers = hidden_layers[:, -4:]  # get the last 4 layers

    logits = kim_model(hidden_layers)

    loss = criterion(logits.view(-1, 2), b_labels.view(-1))

    # accumulate validation loss
    total_test_loss += loss.item()
```

```
        # calculate preds
        _, predicted = torch.max(logits, 1)

        # move logits and labels to CPU
        predicted = predicted.detach().cpu().numpy()
        y_true = b_labels.detach().cpu().numpy()

        # calculate f1
        total_test_f1 += f1_score(predicted, y_true,
                                  average='weighted',
                                  labels=np.unique(predicted))

        # calculate accuracy
        total_test_accuracy += accuracy_score(predicted, y_true)

        # calculate precision
        total_test_precision += precision_score(predicted, y_true,
                                                average='weighted',
                                                labels=np.unique(predicted))

        # calculate recall
        total_test_recall += recall_score(predicted, y_true,
                                                average='weighted',
                                                labels=np.unique(predicted))

    # report final accuracy of test run
    avg_accuracy = total_test_accuracy / len(dataloader)

    # report final f1 of test run
    avg_test_f1 = total_test_f1 / len(dataloader)

    # report final f1 of test run
    avg_precision = total_test_precision / len(dataloader)

    # report final f1 of test run
    avg_recall = total_test_recall / len(dataloader)
```

```
    # calculate the average loss over all of the batches.
    avg_test_loss = total_test_loss / len(dataloader)

    # capture end testing time
    training_time = format_time(time.time() - total_t0)

    # Record all statistics from this epoch.
    test_stats.append(
        {
            'Test Loss': avg_test_loss,
            'Test Accur.': avg_accuracy,
            'Test precision': avg_precision,
            'Test recall': avg_recall,
            'Test F1': avg_test_f1,
            'Test Time': training_time
        }
    )
    # print result summaries
    print("")
    print("summary results")
    print("epoch | test loss | test f1 | test time")
    print(f"{epoch+1:5d} | {avg_test_loss:.5f} | {avg_test_f1:.5f} | {training_time:}"

    return None

# instantiate BERT model with hidden states
#model = BertModel.from_pretrained('bert-base-uncased', output_hidden_states=True).cu
model = BertModel.from_pretrained('allenai/scibert_scivocab_cased', output_hidden_sta
#model = BertModel.from_pretrained('microsoft/deberta-large', output_hidden_states=Tru
#model = BertModel.from_pretrained('roberta-large', output_hidden_states=True).cuda()

# instantiate CNN config

class config:
    def __init__(self):
```

```
        config.num_classes = 2  # binary
        config.output_channel = 16  # number of kernels
        config.embedding_dim = 768  # embed dimension
        config.dropout = 0.4  # dropout value
        return None


# create config
config1 = config()

# instantiate CNN
kim_model = KimCNN(config1).cuda()

# set loss
criterion = nn.CrossEntropyLoss()

# set number of epochs
epochs = 4

# only train the last 4 layers; saves ~600mb of GPU mem and 30s of compute
BERT_parameters = []
allowed_layers = [11, 10, 9, 8]

for name, param in model.named_parameters():
    for layer_num in allowed_layers:
        layer_num = str(layer_num)
        if ".{}.".format(layer_num) in name:
            BERT_parameters.append(param)

# set optimizer
optimizer = AdamW([{'params': BERT_parameters, 'lr': 2e-5}], weight_decay=1.0)


# set LR scheduler
total_steps = len(train_dataloader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer,
```

```
                                                num_warmup_steps=0,
                                                num_training_steps=total_steps)

# create gradient scaler for mixed precision
scaler = GradScaler()

# create training result storage
training_stats = []
valid_stats = []
best_valid_loss = float('inf')

# for each epoch
for epoch in range(epochs):
    # train
    train(model, train_dataloader, optimizer)
    # validate
    validating(model, valid_dataloader)
    # check validation loss
    if valid_stats[epoch]['Val Loss'] < best_valid_loss:
        best_valid_loss = valid_stats[epoch]['Val Loss']
        # save best model for use later
        torch.save(model.state_dict(), 'bert-cnn-model1.pt')  # torch save
        model_to_save = model.module if hasattr(model, 'module') else model
        model_to_save.save_pretrained('./model_save/bert-cnn/')  # transformers save
        tokenizer.save_pretrained('./model_save/bert-cnn/')  # transformers save

# organize results
#pd.set_option('precision', 3)
df_train_stats = pd.DataFrame(data=training_stats)
df_valid_stats = pd.DataFrame(data=valid_stats)
df_stats = pd.concat([df_train_stats, df_valid_stats], axis=1)
df_stats.insert(0, 'Epoch', range(1, len(df_stats)+1))
df_stats = df_stats.set_index('Epoch')
df_stats

# test the model
```

```
test_stats = []
model.load_state_dict(torch.load('bert-cnn-model1.pt'))

testing(model, test_dataloader)

df_test_stats = pd.DataFrame(data=test_stats)
df_test_stats
```