NLP-Based Automated Conspiracy Detection for Massive Twitter Datasets

Large-scale inference with Graphcore Intelligence Processing Units

Rohullah Akbari

Thesis submitted for the degree of
Master in Data Science
60 credits

Department of Mathematics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Spring 2023

NLP-Based Automated Conspiracy Detection for Massive Twitter Datasets

Large-scale inference with Graphcore Intelligence Processing Units

Rohullah Akbari

NLP-Based Automated Conspiracy Detection for Massive Twitter Datasets

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The COVID-19 pandemic, which affected societies worldwide, and many were compelled to shut down, also affected social media platforms like Twitter. The COVID-19-related misinformation on Twitter covered various topics and contained many competing narratives, including conspiracy theories. This thesis aims to analyze such conspiracy theories using the manually labeled nine conspiracy categories from the COCO dataset, a multilabel multiclass text-based dataset. The first part of the thesis focuses on developing machine-learning models to detect these categories. Classical approaches such as Bag-of-Words, TF-IDF, and N-gram variations were used as baseline models. In addition, various pre-trained Transformer models, including general and domain-specific models, were explored using multi-task and ensemble learning methods. Among these models, COVID-Twitter-BERT (CT-BERT), pretrained on a large corpus of Twitter messages related to COVID-19, achieved the best performance on the dataset.

The second half of the thesis focuses on conducting a large-scale inference of 2.5 billion tweets, which were reduced to approximately 381 million tweets after removing retweets and duplicates. First, we evaluate the speed performance of Graphical Processing Units (GPUs) and Graphcore Intelligence Processing Units (IPUs) for NLP-based inference tasks to find the fastest processing unit. The experiment results revealed that the IPUs performed relatively faster than the GPUs for the inference task and were, therefore, used to perform the inference on all tweets, which produced a substantial dataset annotated with nine conspiracy theories, each comprising three subcategories: promoting, discussing, and non-conspiracy.

Acknowledgments

This thesis represents the completion of nearly a year's worth of effort, and I attribute its completion to the supervision and guidance provided by my supervisors, Johannes Langguth, Daniel Thilo Schroeder, Konstantin Pogorelov, and Andrey Kutuzov. I am immensely grateful to each of them for their valuable time and assistance throughout the thesis. I would like to extend a special appreciation to Johannes for granting me the opportunity to work on this super cool thesis and for our numerous insightful discussions on the topic.

I am deeply grateful to my family for their unwavering support. Their constant encouragement and presence have been invaluable, and I extend my heartfelt appreciation to them.

As I submit this thesis, it marks the end of my time at the UiO campus. Over the last five years, the campus has served as my second home, and bidding farewell feels weird. I sincerely thank my fellow students and friends, Philip, Haiat, Shamshad, and Aleks. Our shared moments of lunches, meetups, discussions, and many, many late-night study sessions have enriched my academic journey and created lasting memories.

Contents

List of Figures

List of Tables

Acronyms

NLP Natural Language Processing 35

BoW Bag of Words 14

SVM Support vector machine 16

LogReg Logistic Regression 19

NNs Neural networks 26

NN Neural network 33

FFNN Feedforward neural network ix

ML machine learning 55

MTL Multi-task learning 35

BERT Bidirectional Encoder Representations from Transformers . . . 40

RoBERTa Robustly Optimized BERT Approach 41

CT-BERT COVID-Twitter-BERT i

MCC Matthews correlation coefficient 49

COCO An annotated Twitter dataset of COVID-19 conspiracy theories ix

Chapter 1

Introduction

1.1 Motivation

Before the internet, print media, radio, and television were the primary news and knowledge sources. However, the digital revolution drastically transformed societies, providing unparalleled access to global information and enabling real-time communication, eliminating time and space barriers. In this digital age, social media platforms like Facebook, Twitter, and Instagram have become virtual hubs for community growth and idea exchange, surpassing pre-internet limitations and opening up vast resources of information and connections. Nevertheless, as we traverse the digital landscape, we must address its inherent challenges, including concerns about information reliability, impacts on mental health, and potential misuse of disinformation and propaganda.

The COVID-19 pandemic is a global pandemic that was first detected in Wuhan City, Hubei Province of China, on 31 December 2019, where the WHO China Country Office was informed of cases of pneumonia of unknown etiology[1]. Due to the rapid transmission of COVID-19, many societies were compelled to enforce lockdown measures. Furthermore, similar to the swift global dissemination of the virus, social media posts concerning the virus on platforms such as Twitter and Facebook propagated rapidly as well. Unfortunately, a significant amount of misinformation was spread on these platforms at the start of the pandemic, including conspiracy theories, fake cures, and misleading information about the virus. Some examples of these misinformation types were unverified treatments, conspiracy theories about an intentional virus outbreak, and political propaganda, often to promote a particular political agenda or blame a particular party or country for the outbreak.

The term *digital wildfires* often refers to the rapid dissemination of misinformation across social media platforms, particularly during crises or breaking news events. Analogous to the swift spread of a wildfire through dry vegetation, misinformation can propagate quickly through social media,

[1] https://www.who.int/emergencies/disease-outbreak-news/item/2020-DON229

fueled by user engagement such as shares, likes, and comments. The quick proliferation of inaccurate, counterfactual, or deliberately misleading information, which can quickly infiltrate public awareness and yield severe real-world consequences, is considered one of the top global risks of the 21st century [106]. Nevertheless, only a minuscule fraction of the pervasive misinformation on the internet results in harmful real-world actions. One such example is the burning of 5G towers due to the misguided belief that these towers contribute to the spread of viruses [85]. Identifying potential digital wildfires before they inflict significant real-world damage is crucial. However, given the sheer volume of text on social media, such as the 500 million daily tweets on Twitter, only systems capable of automatically detecting misinformation narratives can reliably fulfill this role.

During the pandemic, a wave of COVID-19-related conspiracy theories emerged alongside the misinformation pervading social media. Despite the efforts of major social networks, an overload of fabricated facts, baseless theories, and seemingly credible posts inundated online media sources. Such rumors and rapidly spreading inaccuracies quickly infiltrated public awareness, leading to significant real-world consequences. Heightened public focus on this issue has prompted content moderation and particular limitations on freedom of speech to deter manipulation of pandemic-related public opinion. Therefore, addressing the deceptive and misleading content within the COVID-19 *infodemic* is crucial. In this thesis, we aim to distinguish between content free of misinformation and content tainted with misinformation by analyzing, with building machine learning and NLP-based models, the COCO dataset[52]. The COCO dataset is a multi-label, multi-class dataset comprising twelve distinct conspiracy categories, each with three unique subcategories.

We aim to develop a robust language-based model to detect conspiracy on large amounts of Twitter data with the help of pre-trained *transformer*-based models such as *BERT* [23, 99]. These transformer-based models, which leverage self-attention mechanisms and parallel processing to effectively capture long-range dependencies and complex linguistic patterns within the text, have transformed the field of natural language processing (NLP) and have a significantly advanced state-of-the-art in various NLP tasks by enabling more profound understanding of context and semantics in textual data. As a result, they have set new benchmarks and reshaped the research landscape in NLP. With the pandemic, several pre-trained transformer models have been developed to tackle the misinformation on Twitter. Among them is the CT-BERT and BERTweet [69, 71]. We intend to fine-tune and evaluate these and other pre-trained transformer-based models to identify the one that optimally suits the COCO dataset.

The second part of the thesis is about making a large-scale inference. We will evaluate different processing units, such as GPUs and Graphcore

IPUs[2], to find the one which is best suited for large-scale inference. GPUs, which are the state-of-art processing units for deep learning models, are designed for parallel processing, which makes them particularly effective for tasks that can be broken down into many smaller tasks that can be performed simultaneously. On the other hand, IPUs are a type of specialized processing unit designed to accelerate complex machine-learning tasks. They are built with a unique architecture that includes thousands of parallel computing cores, advanced on-chip memory, and optimized software to achieve high performance and efficiency for artificial intelligence applications. The inference aims to annotate a large-scale Twitter dataset comprising 2.5 billion tweets. Owing to the massiveness of the dataset, we require a processing unit capable of conducting inference swiftly to ensure timely completion.

1.2 Problem Statement

Although there has been much research on fake news and conspiracy theory detection related to COVID-19, to the author's knowledge, no work has been conducted on datasets featuring a high degree of specificity, such as multiple categories of conspiracies for in-depth analysis of conspiracy theories. In addition, no other work has made a large-scale inference at the size of 2.5 billion tweet texts. With these points in mind, we formulate the following research question:

RQ1 Which machine-learning model is best suited for creating a multi-class multi-label conspiracy theory detection based on the COCO dataset?

1a) Some conspiracy categories are known to be more keyword-based than others in the COCO dataset. How big is the gap between the text vectorization methods like the *TF-IDF* approaches compared to the large pre-trained models for these categories?

RQ2 How does the performance of IPUs compare to other hardware solutions, such as GPUs, when conducting large-scale inference tasks for conspiracy theory detection using large pre-trained models?

2a) What is the distribution of conspiracy theories on big data? How do the different conspiracy categories evolve through time?

We aim to solve these research questions with the following research objectives:

Objective 1: Investigate the performance of keyword-based methods on the COCO dataset, including TF-IDF and BOW approaches.

Objective 2: Investigate the performance of general and domain-specific pre-trained transformer models.

[2] https://www.graphcore.ai/products/ipu

Objective 3: Explore the impact of advanced learning methods, such as deep ensemble methods and multi-task learning, on the best-performing model from Objective 2. Additionally, assess the effects of augmented techniques on the datasets.

Objective 4: Develop deep learning models on IPUs and measure their training and prediction times. In addition, run the inference on big data.

1.3 Main Contributions

This thesis contributes to the field of fake news and conspiracy theory detection. We proposed four research objectives to answer the research questions. In section, we will briefly describe the main contributions behind this thesis by sequentially repeating and explaining how these were met.

Objective 1: *Investigate the performance of keyword-based methods on the COCO dataset, including TF-IDF and BOW approaches.*

We support this objective by analyzing the dataset that was initially scraped tweets using COVID-19-related and was annotated with conspiracy theory-related synonyms, resulting in specific keywords in certain conspiracy categories. Given a category, discussing it without mentioning its name is nearly impossible. For instance, it isn't easy to discuss the **New World Order** category without referring to it directly. Similarly, discussing the **Population Reduction** category, which supports the conspiracy theory that COVID-19 was created to decrease the world's population, usually requires mentioning Bill Gates and George Soros. As a result, we believe that word count-based methods, such as *TF-IDF* and *BOW*, can help establish a robust baseline model for the COCO dataset.

Objective 2: *Investigate the performance of general and domain-specific pre-trained transformer models.*

This objective aims to investigate the performance of pre-trained transformer models on the COCO dataset and identify the top-performing model. We have chosen two general pre-trained models and four domain-specific ones pre-trained on similar data to achieve this. We assess the models using two evaluation approaches, One-for-All and One-for-One. Furthermore, we conduct experiments with various loss functions to refine our evaluation of the best model.

Objective 3: *Explore the impact of advanced learning methods, such as deep ensemble and multi-task learning, on the best-performing model from Objective 2. Additionally, assess the effects of augmented techniques on the datasets.*

The primary goal of this endeavor is to enhance the model's performance beyond its previous achievements. To accomplish this, we employ sophisticated machine-learning strategies, including multi-task learning and deep ensemble techniques. Additionally, we are exploring using augmentation methods and a fine-grained version of the dataset. Upon completion, we will have refined our model to its optimal performance on the dataset, making it ready for large-scale data inference.

Objective 4: *Develop deep learning models on Graphcore's IPUs and measure their training and prediction times. In addition, run the inference on big data.*

Ultimately, we implement the selected model on Graphcore's IPUs to assess their training and prediction times. This objective aims to construct an equivalent model on Graphcore's IPUs for large-scale data inference. Additionally, we compare the inference time of the IPUs with that of other state-of-the-art GPUs used for deploying deep learning models. We will have inferred big data and their corresponding predicted labels upon achieving this objective.

1.4 Outline

This thesis is divided into eight chapters. The following summary gives a brief introduction to the next chapters.

Background Chapter 2 presents the background, describing the theoretical framework and previous work supporting this thesis's methods.

Dataset Chapter 3 introduces and discusses the COCO dataset, where we will present the conspiracy categories in detail and, in addition, provide the labeling instruction used to label those categories. We will also discuss *Matthews correlation coefficient* (MCC) as the evaluation score of the dataset. Finally, we present the alternative versions of the dataset by feature engineering of the labels and other COVID-19-related open-source datasets.

Experiments with classical approaches Chapter 4 describes the execution, results, and discussion of the *classical* approaches such as the TF-IDF, BOW, and *N-gram*.

Experiments with transformer-based approaches Chapter 5 describes the execution, results, and discussion of the transformer-based approaches. Here we also introduce the One-for-All and One-for-One approaches. In addition, we also experiment with different activation functions.

Experiments with other datasets and advanced learning techniques Chapter 6 is about further experimenting with the best model with more advanced techniques such as deep ensemble learning, data augmentation

techniques, alternative versions of the dataset, and more advanced multi-task learning.

Experiments on intelligence processing units Chapter 7 introduces Graphcore's IPUs and motivates why these processing units are a good fit for machine learning. This chapter also gives an overview of implementing deep learning models on IPUs. Finally, we also describe the setup of the BERT model and the time comparison of IPUs with GPUs.

Inference on big data Chapter 8 details big data and motivates the reason for inference. Furthermore, this chapter provides results and a discussion of the inference.

Conclusion & Future work The results from the previous chapter are put into context, and final conclusions are drawn. The results were interpreted considering the known information, and connections between research questions and the results are also provided. Finally, the results were also discussed in a broader context and a discussion regarding future work is provided.

Chapter 2

Background

Fake news is a term used to describe false stories that appear to be news spread on the internet or through other media to influence political views or as a form of amusement. However, fake news can take on various forms. The four most common subcategories are:

- Political misinformation: False or misleading information about political candidates, elections, or policies. An example of this category is the 2016 United States presidential election, where it was found a large amount of fake news on Twitter [37].
- Health misinformation: False or unverified information related to health, such as cures or disease treatments. The use of hydroxychloroquine as a cure for COVID-19 is an example of this category [44, 108].
- Financial fraud detection: Financial scams and get-rich-quick schemes deceive individuals with fraudulent promises of substantial and effortless financial gains. Examples include pyramid and Ponzi schemes, forex and crypto scams, work-from-home frauds, and fake investment programs. [55, 63, 105].
- Scams: False or misleading information designed to deceive people into giving up their personal information or money.

This thesis will broadly study COVID-19-related conspiracy detection. Conspiracy theories can range from mildly implausible to entirely absurd and are typically spread through social media platforms. This chapter will provide an overview of previous work on fake news and conspiracy detection, as well as machine learning and NLP-based methods used to detect and combat them.

2.1 Previous work

This section will provide an overview of the various types of fake news detection. We will first present the general types of fake news detection and subsequently provide a more detailed discussion of domain-specific types, such as Twitter-based COVID-19-related fake news and

conspiracy theories. However, it is essential to note that this section does not comprehensively explain the machine learning and natural language processing algorithms used in fake news detection but briefly mentions them. The subsequent section will elaborate on the technical details of these algorithms. Within the general realm of fake news analysis, we will examine fact-checking-based, source-based, and context-based fake news analysis. We will discuss the differences between these analyses and the methodologies employed to combat them.

2.1.1 Fact-checking-based fake news

Fact-checking-based fake news analysis is one of the most studied fields of all fake news analyses. This analysis adopts a knowledge-based perspective to verify the accuracy of claims presented in a news article. It involves cross-referencing the claims with multiple sources to ensure their factual correctness. By comparing the information with reputable and reliable sources, one can assess the validity and reliability of the claims made in the article. This method promotes a more accurate understanding of the news and helps to identify and combat misinformation or fake news. Fact-checking can also involve identifying sources of misinformation or propaganda. This method was initially developed in journalism. Journalists aimed to assess news authenticity by comparing the knowledge extracted from to-be-verified news content (e.g., its claims or statements) with known facts [114].

Fact-checking can help identify false or misleading claims and provide readers with accurate information. Fact-checking has traditionally been performed with two common methodologies; manual and automatic fact-checking. Trained fact-checkers typically do manual fact-checking. Among the trained fact-checkers, there exist two different parts. The first part consists of experts, relying on domain experts as fact-checkers to verify the given news contents. Expert-based fact-checking is often conducted by a small group of highly objective fact-checkers. However, despite this highly accurate method, it is costly and poorly scalable. Lately, many websites have emerged to allow expert-based fact-checking better serve the public. Some of these websites are *PolitiFact*[1], *The Washington Post Fact Checker*[2], *FactCheck*[3] and *Snopes*[4]. Common for these websites is that they comment on American politics and verify statements, claims, speeches, and news. In contrast to expert-based fact-checking, which is a highly costly procedure and not scalable, crowd-sourced fact-checking solves the problem by being more affordable and scalable by having a team of average-level domain-expert to annotate the news. *Fiskkit*[5] is an example of this category, where users can upload articles, provide ratings for sentences within articles, and

[1] https://www.politifact.com/
[2] https://www.washingtonpost.com/news/fact-checker/
[3] https://www.factcheck.org/
[4] https://www.snopes.com/
[5] https://fiskkit.com/

choose tags that best describe the articles.

With the rise of machine learning and natural language processing, fact-checking-based analysis has been converted into an automatic procedure. Due to the huge amount of newly created information available on the internet, especially on social media, having good language models is necessary to tackle scalability. Traditionally, fact-checking analysis has been performed with Information Retrieval (IR) [62, 103]. IR is a field of study that deals with searching for and retrieving relevant information from large collections of unstructured or semi-structured data, such as text documents, images, or videos. It uses various techniques and algorithms, such as indexing, querying, ranking, and relevance feedback, to effectively search and retrieve information based on user queries. IR techniques can be used to quickly search through large amounts of data to find evidence that can be used to verify or refute claims made in a news article.

Despite the utility of IR in fact-checking tasks, transformer models have emerged as prominent candidates, seeking to boost the performance in these domains. Transformer models like BERT [23] have shown promise in fact-checking tasks [100, 107]. In fact-checking tasks, BERT can be used to identify the claims made in a news article and evaluate their accuracy by comparing them to relevant information from other sources. The standard way to use BERT for fact-checking is by fine-tuning the model on a large dataset of labeled articles and then predict on unseen articles. Another approach is to use BERT to identify key phrases and entities in a news article, then cross-reference them with other sources to verify their precision.

Fact-checking COVID-19

One example of fact-checking tasks based on COVID-19 is the FakeCovid, a dataset containing 5182 fact-checked news articles for COVID-19 collected from 4th of January 2020 to 15th of May 2020 [87]. The articles were collected from 92 different fact-checking websites, where each has been manually labeled into 11 different categories, and the whole dataset is in 40 languages from 105 countries. Typical for these websites is verifying the correctness of misinformation spread across several topics. As for the fact-checking process, they manually verify the news article's authenticity and perform a contextual analysis. The fact-checking website assigns a class for each fact-checked article. Each fact-checking website has a set of classes designed by them; for instance, the classes may be "true", "false", or "partially false". In addition, each data sample has been manually annotated into different categories describing several topics, for example, origin, virus, or international response. The category attribute has been used to denote the article's topic. The dataset consists of the following attributes:

- **FCID** The identifier of each data sample.

- **Source** of articles.
- **Title** from the collected link.
- **Published Date**
- **Content of articles** The content of the articles in text form.
- **Class** from the fact-checking website
- **Social media link**
- **Fact checking website**
- **Country** of the post.
- **Category** Manually labeled category describing the topic.
- **Language** of the content.

2.1.2 Style-based analysis

Style-based fake news analysis is an approach to identifying fake news that relies on analyzing the writing style of a news article or social media post. The intuition and assumption behind style-based methods are that malicious entities prefer to write fake news in a special style to encourage others to read and convince them to trust. The idea behind style-based fake news detection is that fake news spreaders often use a different writing style than real news spreaders, which can be used to identify them. One advantage of style-based fake news detection is that it can be effective even when the content of a news article is difficult to verify or there are no external sources of information available. Style-based detection can also identify patterns of language use that suggest an article is fake or misleading. However, style-based fake news detection has some limitations, e.g., some fake news articles may be written in a similar style to real news articles, making them difficult to detect using this approach alone, and it also requires a large dataset of labeled articles for training, which can be difficult to obtain for certain languages or domains.

The traditional method for detecting fake news based on their style is to extract stylometric features such as sentence length, word frequency, and punctuation usage. These features are used to create a profile of the writing style of each news article. These features are then driven through the traditional machine learning models, such as XGBoost, random forest, etc. One example of this approach is based on training two classifiers, a neural network and a model based on stylometric features, on a corpus of 103,219 documents from 223 online sources labeled by media experts, devising realistic evaluation scenarios [81].

2.1.3 Source analysis

Source analysis is an approach to detect fake news that involves evaluating the credibility and reputation of the source of a news article or social media post. The idea is that reputable sources are more likely to publish accurate information than less reputable sources. Source analysis can involve several different factors, such as the history of the source, the editorial policies of the source, and the expertise of the authors. For example, a news article from a well-established and reputable news organization like The New York Times or BBC is generally considered more trustworthy than an article from a relatively unknown website or blog. Source analysis can be a helpful approach to detecting fake news, as it can help identify articles that are likely to be inaccurate or misleading based on the reputation of the source. However, it is important to note that even reputable sources sometimes make mistakes or publish inaccurate information. Therefore, source analysis should be combined with other approaches, such as fact-checking, to improve the accuracy of fake news detection.

2.1.4 Context analysis & Social Media

Lastly, we present fake news analysis based on the context of news articles or social media posts. It involves analyzing the context in which a news article or social media post is posted to identify patterns or inconsistencies that suggest it is fake or misleading. Context analysis considers factors such as the language used and the targeted audience. The language used in a news article can indicate its accuracy or bias. For example, sensational or hyperbolic language may suggest that the article is intended to manipulate or mislead the reader.

Social media platforms present unique challenges for detecting fake news because of the large volume of content and the rapid speed at which it can be shared. Context analysis can be helpful because it allows for evaluating patterns and trends in the information shared on social media. Over recent years, the growth of online social media has dramatically streamlined how people communicate. Social media users share information, connect with others, and stay informed about trending events. Nevertheless, much recent information on social media is doubtful and, in some cases, intended to mislead.

One example of fake news on social media is the 2016 US presidential election. Twitter was heavily used as a platform for sharing news and opinions. One study found that 25% of 171 million tweets spread either fake or extremely biased news [13]. Many of these fake news stories were designed to be sensational or to play to people's fears and biases, and they often spread quickly through social media networks. Twitter and other social media companies have since taken steps to combat the spread of fake news on their platforms, including implementing fact-checking tools. However, the influence of fake news on social media remains a concern, and it will

likely continue to be a topic of study and debate in future elections.

To combat the spread of fake news, various efforts have been made. One such effort is the Fake News Challenge (FNC-1)[6], which brought together academic researchers and industry professionals to explore and develop techniques for automatically identifying fake news articles. FNC-1 involved a dataset of news articles labeled as either *fake* or *real*, and participants were tasked with creating machine learning models that could accurately distinguish between the two. The challenge gained significant attention from the NLP community, with 50 academic and industry teams participating. Another approach to tackling the fake news problem is a multimodal attention network proposed by Vo et al. [104]. Their framework aims to prevent the spread of fake news by ranking fact-checking documents based on relevancy. By considering the content of an original tweet that may contain misinformation, their algorithm can directly warn fake news posters and online users, including the poster's followers, about the misinformation and discourage them from spreading fake news. The framework uses text and images to search for fact-checking articles and promotes verified content on social media.

2.1.5 COVID-19 fake news on Twitter

Till now, we discussed the non-COVID-19 fake news strategies. This section will dive deeper into previous work on COVID-19-related fake news analysis. The common theme in tweets during the pandemic were fake news narratives, conspiracy theories, and hate speech toward a political entity and country. The term *infodedemic* was used by researchers in this period to describe the overload of misinformation that makes it difficult for people to find accurate and trustworthy information. Infodemics can be harmful because they can spread false information and conspiracy theories, undermining public trust in science and public health interventions. Several works have been released during the pandemic to combat infodemics. The *FakeNews: Corona Virus and 5G Conspiracy task* at MediaEval 2020[7] introduced another dataset that focused on the classification of tweet texts and retweet cascades for the detection of fast-spreading misinformation on Twitter [78, 79].

El-Patwa et al. [74] introduced the *Fighting an Infodemic*, an annotated dataset of 10,700 social media posts and articles of real and fake news on COVID-19. Das et al. [20] proposed a heuristic-driven ensemble framework at the CONSTRAINT[8] COVID-19 Fake News Detection in English challenge. Their suggested model was based on an ensemble model consisting of pre-trained Transformer models. In addition, Glazkova et al. [31]

[6] http://www.fakenewschallenge.org/

[7] https://multimediaeval.github.io/editions/2020/

[8] First Workshop on Combating Online Hostile Posts in Regional Languages during Emergency Situation Collocated with AAAI 2021: https://www.lcs2.in/CONSTRAINT-2021/

deployed CT-BERT [69] and various ensembles of it in the same challenge.

Elhahad et al. [25] introduced a misleading-information detection model related to COVID-19 that relied on the World Health Organization, UNICEF, and the United Nations sources of information, as well as epidemiological material collected from a range of fact-checking websites. Their proposed model is based on data validity since the data were obtained from reliable sources and consisted of ten distinct machine learning algorithms with seven feature extraction techniques. *TF-IDF* was used to embed the textual data into numerical data. The final prediction was based on a *voting ensemble* of these machine learning models: whether a tweet is real or misleading. Like this work, Al-Rakhami et al. [83] proposed an ensemble-learning-based framework for verifying the credibility of tweets. The proposed approach classified the tweets into categories of *credible* or *non-credible* and is based on various features, including tweet- and user-level features.

Efforts were made to automate COVID-19 fake news detection by designing a pipeline based on fact-checking algorithms and textual entailment [101]. Their approach consists of two models; the first model leverages a novel fact-checking algorithm that retrieves the most relevant facts concerning user claims about particular COVID-19. The second model verifies the claim's truth level by computing the textual entailment between the claim and the true facts retrieved from a manually curated COVID-19 dataset. The dataset they used is based on a publicly available knowledge source of more than 5000 COVID-19 false claims and verified explanations, a subset of which was internally annotated and cross-validated to train and evaluate our models.

Despite limited resources in other languages, Kar et al. [49] proposed an approach to detect fake news about COVID-19 early on from social media, such as Twitter, for multiple Indic-Languages except for English. In addition, they also proposed a BERT-based model augmented with additional relevant features extracted from Twitter to identify fake tweets. To expand the approach to multiple Indic languages, the alternative to mBERT [58] based model is fine-tuned over a created dataset in Hindi and Bengali. In addition to the Indic Languages, several efforts were made for the Arabic languages [39]. Alqurashi et al. [2] proposed the largest Arabic Twitter Dataset on COVID-19. Haouari et al. [39] proposed the ArCOV19-Rumors dataset, a COVID-19 Twitter dataset for misinformation detection, collected from 138 verified claims, mostly from popular fact-checking websites, and identified 9.4K relevant tweets to those claims.

2.2 Classical Approaches

This section will discuss the traditional techniques used in conspiracy detection tasks. These methods are called *classical* because they depend on text vectorization approaches that do not involve deep learning or pre-

trained language models. Text vectorization is vital for various natural language processing (NLP) tasks, including text classification, sentiment analysis, and language translation. Vectorization entails transforming text into a numerical format that machine learning algorithms can process. This procedure, also known as feature extraction or feature representation, requires the conversion of text into vectors or sequences of numbers that correspond to different linguistic properties of the text. We will present and explore two widely utilized text vectorization techniques: Bag of Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). The standard features utilized in these vectorization techniques are either indicators or counts. An indicator feature signifies the presence of a specific word in the text by assigning a value of 1 or 0. In contrast, the value of a count feature is contingent upon the frequency of occurrence of a specific word within the text. The works by Baarir et al. [5] and Poddar et al. [76] exemplify the application of these text vectorization methods for creating machine learning models aimed at detecting fake news.

Typically, machine learning models are used to learn text vectorization for various labels. These machine learning algorithms learn a set of weights for each feature and combine them into a linear function to predict the probability of a text belonging to a specific *class*. The weights indicate the importance of each feature in predicting the outcome. The support vector machine is among the popular machine learning models, which seeks to find a *hyperplane* that maximally separates the texts into their respective classes. A set of weights defines the hyperplane, and the SVM algorithm learns these weights by finding the optimal *margin* between the classes.

While the TF-IDF and BoW methods are effective for certain NLP tasks, they have several limitations. The primary drawback is their disregard for word order, which hinders the capture of context and semantic relationships between words. This can adversely affect the performance of machine learning models in tasks demanding linguistic context comprehension. Additionally, BoW is highly sensitive to frequently occurring yet potentially uninformative words. Although TF-IDF tries to counteract this by down-weighting common words across text samples, it may still struggle to fully recognize the significance of rare, informative words. Moreover, since both techniques depend on a predefined vocabulary from the training corpus, they cannot process words absent from the training set, leading to reduced performance when encountering new or unseen words. Despite these shortcomings, these methods serve as baseline models in various studies. In this section, we will explore these methods in greater detail, along with other machine learning models, such as support vector machines and logistic regression, while also discussing the technical aspects of machine learning.

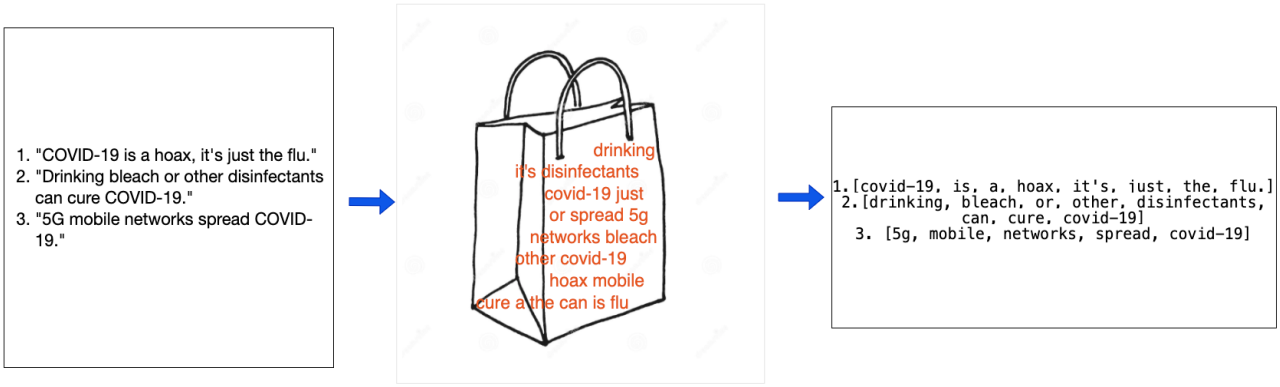


Figure 2.1: Graphic illustration of BoW; converting text samples into a list of word features. Note that the sentences used in this example are not real tweets.

2.2.1 The Bag of Words (BoW)

The BoW is a standard feature extraction method based on representing a text document as if it were a *bag of words*, i.e., an unordered set of words. This approach treats each word count as a feature. In order words, this method is based on counting the occurrences of each word in the text, and each data sample gets assigned a vector based on all of the word features. Each text sample is represented as sparse vocabulary vectors where the core elements of this representation are words.

A worked example

Figure 2.1 shows a simple example of the BoW representation, where it has three text samples that are being converted into a list of words. This process is known as *tokenization*, separating the words in a text. Furthermore, in such a process, the vocabulary is defined as the unique set of all *tokens*. The vocabulary in the example in Figure 2.1, is

$$\begin{aligned}\mathbf{V} = [&\text{it's, disinfectants, covid-19, just, or, spread, 5g, networks, bleach',}\\ &\text{'other, covid-19, hoax, mobile, cure, a, the, can, is, flu, drinking'}]\end{aligned} \tag{2.1}$$

The length of the vocabulary vector $\mathbf{V}$ is 20, and each text sample will be converted to a vector of the size. The final BoW representation of the samples in Figure 2.1 becomes

$$\begin{aligned}1 &= [0,0,0,1,0,1,0,0,0,1,1,1,0,0,0,0,0,1,1,1]\\ 2 &= [1,0,1,0,1,0,1,1,1,0,0,0,1,0,1,0,0,0,0,0]\\ 3 &= [0,1,0,0,0,0,0,0,1,0,0,0,0,1,0,1,1,0,0,0]\end{aligned}$$

The first element in all vectors represents the word **can**, and this word is only present in the second sample. Therefore, the second sample has been

assigned 1 on the first position, while the other vectors have been assigned 0s. Note that the representation, in this example, has been performed with an indicator not count.

2.2.2 The TF-IDF-based method

In this section, we present the utilization of TF-IDF weightings, a prevalent approach for assigning weights to word counts in term-document matrices. This technique employs the following equation to calculate the significance of a word within a text:

$$\text{tf-idf} = tf(t,d) \cdot idf(t,D) \tag{2.2}$$

The first term in Equation 2.2 describes the *term frequency* of a word in a document and is calculated by counting the raw instances of a word in a document.

$$\text{tf}(t,d) = \log(1 + \text{count}(t,d)) \tag{2.3}$$

The second term in Equation 2.2 describes the *inverse document frequency* of the word across a set of documents. In other words, it explains how rare a word is in the entire document set. This metric can be computed by dividing the total number of documents by the number of documents in which a specific word is contained.

$$\text{idf}(t,D) = \log\left(\frac{N}{count(d \in D : t \in D)}\right) \tag{2.4}$$

The TF-IDF value will be in the range of 0 and 1, where zero indicates that a word is significant and vice versa for the value of one.

2.2.3 Support Vector Machine

Support vector machine (SVM) is one of the most elegant methods for classification problems because of its robust and adaptable features. The SVM algorithm as we know it today was introduced in 1992 [12, 98]. The SVM algorithm performs the classification by constructing a *hyperplane* that separates the features. A hyperplane is a straight line in 2D and a plane in 3D. Theoretically, there could be infinite ways to separate the features; the SVM finds the optimal hyperplane, maximizing the distance between categories. This distance is called the *margin*. The data points on the margin are called *support vectors* (see Figure 2.2). With different kernels, SVMs can handle linear and nonlinear classification problems and are particularly helpful for classifying problems involving massive datasets. The SVM model finds the hyperplane in p-dimensional space, where p is the number of features. In such dimensional space, a hyperplane is called *an affine subspace* of dimension $p - 1$ [94, p.13]. The support vectors are, in practice, the most difficult to classify and are the critical elements of the training set. Optimizing the hyperplane is an optimization problem that optimization techniques solve.

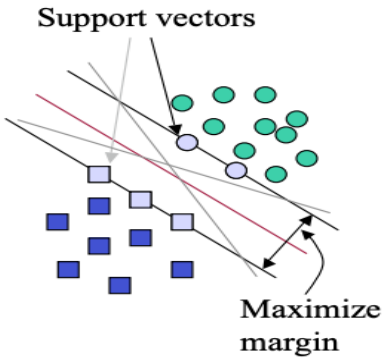


Figure 2.2: A straightforward hyperplane dividing the two classes serves as an example of an SVM model.

The simplest SVM

The most basic SVM is a linear classifier that employs a linear hyperplane. The hyperplane is defined by a weight vector w and an intercept term b, representing the variables we wish to optimize.

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{2.5}$$

where x is the training dataset with N samples. The hyperplane, in this case, separates the data points into two regions, one region for $\mathbf{w}^T\mathbf{x} + b > 0$ and the other for $\mathbf{w}^T\mathbf{x} + b < 0$.

The SVM model optimizes the hyperplane by maximizing the margin, which is the as minimizing [65, p.172]

$$\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 \quad \text{subject to } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \text{ for all } i = 1, \cdots, N \tag{2.6}$$

where $||w||$ is the Euclidean norm of $\mathbf{w}$. $\mathbf{x}^{(i)}$ and $y^{(i)}$ are the sample i from the training data. The $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b)$, is known as the *functional margin* [26], represents the condition that the data points are correctly classified. The term $\frac{1}{2}||\mathbf{w}||^2$, which is the same as $\frac{1}{2}\mathbf{w}^T\mathbf{w}$, is a regularization term that penalizes large values of $\mathbf{w}$ and helps prevent *overfitting*. The optimization problem above can be solved using *Lagrange* function. We introduce a set of non-negative Lagrange multipliers λ_i for each constraint $y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$, and form the Lagrangian [65, p.174]:

$$L(w, b, \lambda) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{m} \lambda_i[y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) - 1] \tag{2.7}$$

The Lagrange multipliers λ_i are only nonzero for the data points that lie on the margin or are misclassified. By differentiating Equation 2.7 with respect to the elements of w and b and setting equal to zero, we can obtain

the optimal values of $\mathbf{w}$ and b.

$$\begin{aligned}\frac{\partial L(w,b,\lambda)}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{m} \lambda_i y^{(i)} \mathbf{x}^{(i)} = 0 \\ \frac{\partial L(w,b,\lambda)}{\partial b} &= -\sum_{i=1}^{m} \lambda_i y^{(i)} = 0\end{aligned} \tag{2.8}$$

The equations above result in the saddle points of the 2.7.

$$\begin{aligned}\mathbf{w} &= \sum_{i=1}^{m} \lambda_i y^{(i)} \mathbf{x}^{(i)} \\ b &= y^{(j)} - \mathbf{w}^T \mathbf{x}^{(j)}\end{aligned} \tag{2.9}$$

where j is the index of any support vector with $0 < \lambda_j < C$, i.e., any support vector that lies on the margin boundary. This leads Equation 2.7 to

$$L\left(\mathbf{w}^*, b^*, \boldsymbol{\lambda}\right) = \sum_{i=1}^{n} \lambda_i - \sum_{i=1}^{n} \lambda_i y_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j, \tag{2.10}$$

Once the expression of 2.9 is learned, this expression is used to predict on a new point $\mathbf{z}$ [65, p.175]

$$\mathbf{w}^{*T} \mathbf{z} + b^* = \left(\sum_{i=1}^{n} \lambda_i t_i \mathbf{x}_i \right)^T \mathbf{z} + b^* \tag{2.11}$$

In other words, the inner product between the new data point and the support vectors must be computed to categorize a new point.

Non-linear classification and kernels

In many real-world problems, the data is not perfectly separable employing a hyperplane, i.e., the dataset is not linearly separable. The SVM model is, therefore, modified to allow some misclassifications by introducing the concept of so-called *slack variables* $\eta_i \geq 0$, and the constraints become $y_i(\mathbf{w}^T \mathbf{x_i} + b) \leq 1 - \eta_i$. Slack variables are non-negative variables that represent the degree of misclassification of each training example by allowing the SVM algorithm to find a hyperplane that still separates the classes but with some margin of error, but for the data points that are correct slack variables are set to be zero. The idea is to allow some examples misclassified while penalizing them for being on the wrong margin or hyperplane. The amount of penalization is controlled by a hyperparameter called C, which determines the trade-off between maximizing the margin and minimizing the misclassification; small C means we prize a large margin over a few errors, and large C means the opposite. Adding all these new components, the function that is minimized to obtain the ideal weights is [42, p.420]

$$\min_{\mathbf{w},b} \frac{1}{2} ||\mathbf{w}|]^2 + C \sum_{i=1}^{n} \xi_i \tag{2.12}$$

where $\sum_{i=1}^{n} \xi_i$ is the distance of all misclassified points from the correct boundary line.

At this point, the data is still not linearly separate, and that is when a *kernel* comes into the picture. A kernel $K(x_i, x_j)$ is a mathematical function that transforms the input data into a higher-dimensional feature space. In other words, kernels allow SVMs to effectively solve non-linear classification problems by implicitly mapping the input data to a higher-dimensional space where a linear separation boundary can be found. The kernel function represents a similarity measure between two input data points x_i and x_j in the original input space. It computes the dot product between the transformed feature vectors $\phi(x_i)$ and $\phi(x_j)$ in the high-dimensional space. The choice of the kernel function depends on the specific problem and the characteristics of the data. Some standard kernel functions are *polynomial*, *radial basis*, and *sigmoidal* [4, p. 174]. More details regarding the difference between these kernels can be found on the same page.

2.2.4 Logistic Regression

Logistic Regression (LogReg) stands out as a simple yet powerful machine learning algorithm, frequently serving as the fundamental classification method in various natural language processing tasks. Moreover, it can be thought of an simplied version of *neural networks*, by featuring a single layer that performs a linear transformation followed by a non-linear activation function. Functioning as a discriminative model, Logistic Regression prioritizes class differentiation and directly computes the class with the highest probability based on the provided textual features.

Logistic regression comprises N features or nodes. Throughout the training process, these nodes are multiplied with a weight matrix. The primary objective of the training is to learn the weights and the bias. Each weight signifies the importance of a feature by assigning it a positive value if it is crucial and a negative value if it is not essential. By employing the multiplication layer, we derive the following equation:

$$z = \left(\sum_{i=1}^{N} w_i x_i \right) + b = W \cdot X + B \tag{2.13}$$

where the X is the input features, e.g., TF-IDF representation of the textual data, W and B are the weights matrix and the bias term learned through the training. Depending on the number of classes in the dataset, the z is driven through different functions. In the case of the binary classification, the z is driven through the *sigmoid* function

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.14}$$

The output of the sigmoid function is between 0 and 1, and the sigmoid function separates the classes with of threshold of 0.5. For *multinomial*

logistic regression or multi-class problems, the z is driven through the *softmax* function. In this case, the dimension of Z is the same as the number of classes (K). Each of these components is then driven through the softmax function

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{K} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{K} \exp(z_i)}, ..., \frac{\exp(z_K)}{\sum_{i=1}^{K} \exp(z_i)} \right] \tag{2.15}$$

The softmax function, like the sigmoid, squashes the values toward 0 or 1. Among the K classes, the class with the highest probability will have the value of 1 and therefore be the most likely.

Learning logistic regression aims to approximate weights and bias values to fit the annotated classes. While training the LogReg, we encounter two components highlighting how well the model matches. The first component shows how close the predicted label is to the gold label, typically done by *accuracy* or *F1* scores. The second component measures the loss, i.e., the distance between the predicted value and the gold label. Different loss functions exist, but the most common is *cross-entropy* loss. While training the logistic regression model, we want the loss function to minimize to its minimum by using *optimization functions*. The job of the optimization function is to update the weights iteratively to minimize this loss function. We will discuss the loss and optimization functions more in-depth in Section 2.3 when we will introduce neural networks.

2.2.5 Bias and Variance

The generalization errors of machine learning models can be characterized by the concepts of *variance* and *bias*. The expected prediction error, or sometimes called *test* error, of any machine learning model, is given by [42, p.37]

$$E\left[(Y - \hat{f}(X))^2\right] = \underbrace{\sigma^2}_{\text{irreducible error}} + \underbrace{\text{Var}(\hat{f}(X))}_{\text{variance}} + \underbrace{(E[\hat{f}(X)] - f(X))^2}_{\text{bias}^2} \tag{2.16}$$

Equation 2.16 signifies the total expected prediction error, which consists of an *irreducible* error, variance, and bias. The expected prediction error always will have an irreducible error which cannot be controlled. However, the variance and the bias can be controlled. The bias term refers to the error that occurs when a model is not complex enough to grasp the genuine underlying relationship between the input features **X** and the target variable **Y**. Conversely, variance refers to the error that occurs when a model is too complex and has learned too much from the training data, including noise or random instabilities. In other words, the bias and the variance are inversely proportional, i.e., one cannot simultaneously minimize both. This leads to the well-known term *bias-variance tradeoff*. It is conceivable to swap bias and variance, resulting in a model with low bias but high variance or vice versa; nevertheless, it is not feasible to

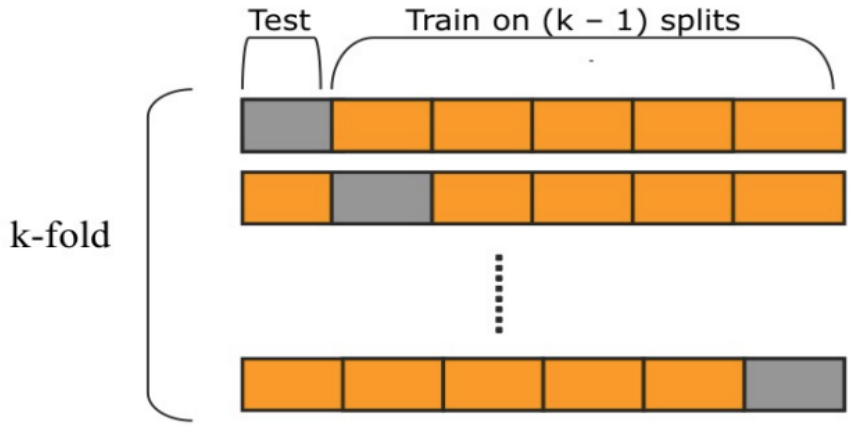


Figure 2.3: Sketch of the cross-validation process[9].

have both at zero because each model involves a tradeoff. However, this tradeoff is helpful for training machine learning models as we need to find a balance between them that gives us the best generalization performance. By understanding the bias-variance tradeoff, we can select the proper machine learning algorithm and optimize its parameters to achieve the best generalization performance. We can also use techniques such as *regularization* and *cross-validation* to balance bias and variance and prevent overfitting.

2.2.6 Cross-validation

Cross-validation, or $k-$fold cross-validation, is a method for dividing the training and test data into k *folds* with approximately the same size [48, p.71]. The key concept is to use the k-th fold for testing, while the other $k-1$ folds are used to train the model and compute the error rate on the test set. This procedure is repeated with another fold as the test set, again training on the other $k-1$ folds. The previous sampling process is repeated k times and averages the test set error rate from these k runs to get an average error rate. More precisely, the average error is given by [42, p.212]

$$\mathrm{CV}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, \hat{f}^{-k(i)}\left(x_i\right)\right) \tag{2.17}$$

where $\hat{f}^{-k(i)}$ is predicted response with observation i in the k fold. There is a bias-variance tradeoff in choosing the number of folds k. A smaller number of k results in a minor variance but a more significant bias. A larger number of k results in a more negligible bias but a larger variance [42, p.243].

[9]The image is taken from Qingkai's Blog: http://qingkaikong.blogspot.com/2017/02/machine-learning-9-more-on-artificial.html.

2.2.7 Hyperparameter tuning and model selection

Every machine learning algorithm has hyperparameters that must be tuned to attain optimal performance. Some examples of hyperparameters include kernel functions, the regularization parameter C for SVM, and the learning rate and number of iterations for LogReg. Additionally, each machine learning model must be tuned for each fold in k-fold cross-validation. Unfortunately, this tuning process can be computationally intensive and time-consuming, but it often results in only marginal improvements.

The tuning of the hyperparameters can be seen as an optimization problem, and the so-called *search algorithms* can be useful [111]. *Grid search* and *random search* are two examples of these search algorithms [9, 10]. Both search algorithms are *decision-theoretic* approaches. The grid search searches exhaustively for a fixed domain of hyperparameter values. The random search randomly selects hyper-parameter combinations in the search space, given limited execution time and resources. More details regarding the comparison and performance of these algorithms can be found at [111].

2.3 Deep learning

Deep Learning is a subset of machine learning focusing entirely on deep neural networks. Different variants of deep neural networks have been developed to solve various problems. Some examples are graph neural network (GNN) that has been designed for solving graph-related problems, recurrent neural network (RNN) to solve sequential data-related problems, and convolutional neural network (CNN) for solving image and video recognition-related tasks.

Recent advancements in natural language processing have been driven by deep learning techniques, particularly recurrent neural networks, and transformer models. RNNs, including advanced variants like LSTMs and GRUs, handle sequential data and improve models' ability to capture context and temporal dependencies [89]. Pretrained embeddings, such as word2vec, GloVe, and FastText, revolutionize word representation by capturing semantic and syntactic relationships. Together, RNNs and pretrained embeddings have significantly enhanced the performance of NLP models across various tasks. However, with the introduction of the transformer model, the RNNs have somewhat been overshadowed. Transformers have emerged as a highly effective and scalable architecture for NLP tasks. Unlike RNNs, transformers rely on self-attention mechanisms to capture dependencies within a sequence, enabling parallelization of computations and overcoming the limitations of RNNs in capturing long-range dependencies.

This section presents an overview of the widely-used neural network

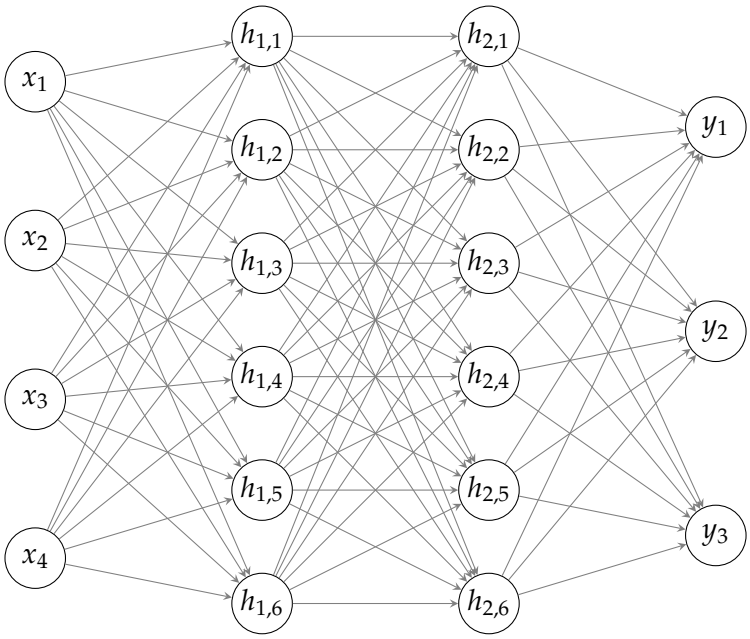


Figure 2.4: A simple feedforward neural network with an input layer, two hidden layers, and one output layer.

architecture, the *feedforward neural networks*, along with a comprehensive discussion of its components. We will delve into the details of critical elements in neural networks, such as *activation functions*, *loss functions*, *optimization functions*, and practical aspects concerning the training of these models. Additionally, we will briefly introduce advanced learning techniques, including *multi-task learning* and *ensemble learning*. While this section does not directly address conspiracy detection, it establishes a solid theoretical foundation for understanding transformer-based models, which will be discussed in the subsequent section.

2.3.1 Feedforward neural network (FFNN)

FFNN is the simplest form of a neural network and consists of multiple layers. A layer is a group of interconnected nodes or neurons that perform a specific computation on their inputs. The layers in an FFNN are acyclic, i.e., the outputs from nodes in each layer are passed to units in the next higher layer. There are three kinds of nodes in a simple FFNN: input nodes, hidden nodes, and output units (see Figure 2.4). The hidden layer takes a weighted sum of its inputs and then applies a non-linearity function. This kind of neural network is often called *fully connected*, i.e., each node within a layer receives the output of the preceding layer as its input. In addition, there is also a connection between all of the nodes of the last

layer to the current layer. As discussed in the section of LogReg, a single hidden layer has a weight vector $\boldsymbol{W}$ and a bias term $\boldsymbol{b}$ as its parameters. Since the FFNN is typically larger and contains more nodes than LogReg, we represent the weights as a matrix and the bias term as a vector. Each element in the weight matrix represents a weight between two connections from two nodes, e.g., the *ji*-th in $\boldsymbol{W}$ represents the weight between the ith input node to the jth output node.

The first step in a simple FFNN is to compute the intermediate vector $\mathbf{z}$ by performing a matrix multiplication between the weight matrix and the input data and then adding the bias term

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.18}$$

The output of this computation is a linear vector which is worthless since it cannot solve complex problems. To transform it into a non-linear vector, we use an *activation function*, to help the network determine whether a node should be activated. The following step applies the activation function

$$\mathbf{a} = \sigma(\mathbf{z}) \tag{2.19}$$

where the σ is the activation function, $\mathbf{W}$ has the dimensionality of $\mathbb{R}^{n_1 \times n_0}$, $\mathbf{x}$ has the dimensionality of $\mathbb{R}^{n_0}$, and $\mathbf{b}$ has the dimensionality of $\mathbb{R}^{n_1}$. The role of the output layer is to compute the final output layer by using the current $\mathbf{a}$, which is done the same way as the first output layer. In the final output layer, we also have a weight matrix $\mathbf{U}$, but it does not have any bias term in the final output layer:

$$\mathbf{v} = \mathbf{U}\mathbf{a} \tag{2.20}$$

where $\mathbf{U}$ has the dimensionality of $\mathbb{R}^{n_2 \times n_1}$, $\mathbf{h}$ has the dimensionality of $\mathbb{R}^{n_1}$, and this leads to $\mathbf{v}$ having the dimensionality of $\mathbb{R}^{n_2}$. The final output layer is driven through another activation function, $\mathbf{q}$, to convert the vector into the most probable classes

$$\hat{\mathbf{y}} = \mathbf{q}(\mathbf{v}) \tag{2.21}$$

Note that the activation function in the hidden layers and the output is not necessarily the same. For example, the standard technique uses the *ReLU* in the hidden layers and *softmax* in the output layer.

2.3.2 Training of feedforward neural networks

The goal of training the neural networks is to approximate the true $\mathbf{y}$ vector by learning the weights matrix $\mathbf{W}$ and the $\mathbf{b}$ for each of the layers of the neural networks. A neural network learns through *forward* and *backward propagation*. One iteration of the forward- and backward propagation is called an *epoch*.

Until now, we have only discussed a so-called forward pass in a neural network, i.e., feeding input data into the network and computing the output of

the network. The input data is multiplied by the weights of the network's neurons during a forward pass, and the results are then sent through an activation function to determine each neuron's output. But how does the network update its weights using the loss function and optimization techniques? The answer is *backpropagation*. The name arrives from the fact that the output from a forward pass propagates backward in the network by calculating the gradient of the network's parameters.

Until now, our discussion has primarily focused on the forward pass in a neural network, which involves feeding input data into the network and computing its output. During this process, the input data is multiplied by the weights associated with the network's neurons, and the results are passed through an activation function to determine the output of each neuron. However, to update the network weights using the loss function and optimization techniques, we employ a method called *backpropagation*. The term backpropagation stems from the fact that the output from the forward pass propagates backward through the network by calculating the gradient of the network's parameters.

The backpropagation mechanism moves through the network from the output to the input layer in reverse order using the chain rule from calculus. The method stores any intermediate variables (partial derivatives) needed to calculate the gradient concerning a few parameters. To explain the forward and backward propagation steps, we expand the feedforward neural network to L layers, where the l-th layer has n_l neurons. Furthermore, let $\boldsymbol{x} \in \mathbb{R}^{n_0}$ be the input vector, $\boldsymbol{y} \in \mathbb{R}^{n_L}$ be the output vector, and $\boldsymbol{\theta}$ be the vector of all weights and biases in the network. To compute the forward pass, we first calculate the weighted inputs and activation function for each layer l as follows:

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}$$
$$\boldsymbol{a}^{(l)} = \sigma(\boldsymbol{z}^{(l)})$$

Where $\boldsymbol{W}^{(l)}$ is the weight matrix connecting layer $l-1$ to layer l and $\boldsymbol{b}^{(l)}$ is the bias vector for layer l. Note that the activation in the index of 0 and L is

$$\boldsymbol{a}^{(0)} = \boldsymbol{x}, \boldsymbol{a}^{(L)} = \boldsymbol{y}$$

In addition, let the $J(\boldsymbol{\theta})$ be the loss between the network output $\hat{\boldsymbol{y}}$ and the target output $\boldsymbol{y}$. To backpropagate the loss, we start by calculating the error in the last layer L

$$\boldsymbol{\delta}^{(L)} = \nabla_{\boldsymbol{y}} J(\boldsymbol{\theta}) \odot \sigma'(\boldsymbol{z}^{(L)})$$

Where $\odot$ denotes element-wise multiplication and $\sigma'(\cdot)$ is the derivative of the activation function. Using the gradient of the loss function concerning the output and the derivative of the activation function applied to the preactivation data, this formula enables us to calculate the error term for the output layer. Then we compute the backpropagate error for each $l = L-1, L-2, \cdots, 1$

$$\delta^{(l)} = ((\boldsymbol{W}^{(l+1)})^T \delta^{(l+1)}) \odot \sigma'(\boldsymbol{z}^{(l)})$$

At this stage, we should have the error at each layer. By using an optimization function, we can update the weights and biases in the network by

$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \eta \nabla_{\boldsymbol{W}^{(l)}} J(\boldsymbol{\theta})$$

$$\boldsymbol{b}^{(l)} \leftarrow \boldsymbol{b}^{(l)} - \eta \nabla_{\boldsymbol{b}^{(l)}} J(\boldsymbol{\theta})$$

Where η is the learning rate and $\nabla_{\boldsymbol{W}^{(l)}} J(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{b}^{(l)}} J(\boldsymbol{\theta})$ are the gradients of the loss function concerning the weight matrix and bias vector for layer l, respectively, this process is repeated for several amounts epochs or until convergence.

2.3.3 Activation functions

Activation functions are critical to deep learning because they introduce non-linearity into the neural network, allowing the model to learn and represent more complex relationships in the input data. Without activation functions, a neural network would be a linear regression model. These functions are applied to the output of each layer in the network. These non-linear transformations help the network determine whether a node should be activated.

The sigmoid function

One of the most used activation functions is *sigmoid*. This function is rarely applied in the hidden layers; most of the time, it is applied in the last layer for binary classification problems. Sigmoid transforms any number in the domain of $\mathbb{R}$ into a domain of $(0, 1)$ [112, p. 184]:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.22}$$

Due to this property, the sigmoid is a good choice for binary classification. In addition, it is differentiable everywhere. The outputs domain of $(0, 1)$ also makes it easy to interpret it as probabilities. The sigmoid's major downside is the issue of *vanishing gradient*. Its gradient vanishes for significant positive and negative arguments. The network does not learn when the gradient reaches 0.

The softmax function

Softmax is another popular activation function used in Neural networks (NNs). Softmax takes a vector of real numbers as input and outputs a probability distribution over the classes by squashing them into a $(0, 1)$

domain for each vector element. The formula of softmax is given by [34, p.24]

$$softmax(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{2.23}$$

The sum of the produced output elements is always one, and the element with the highest probability gets selected as the most likely class according to $\hat{\mathbf{y}}$. Like sigmoid, the softmax function is typically employed in the last layer of a multi-class model. The softmax function appears in almost all the output layers of many deep learning architectures [6, 60].

Rectifier activation function (ReLU)

The rectifier activation function, or *ReLU*, is another widely used activation in neural networks. Despite its simple formula, this function has often been shown to produce outstanding results. It performs well, especially when combined with dropout regularization methods [34, p.46]. The formula for ReLU is given by:

$$ReLU(x) = \max(0, x) = \begin{cases} 0 & x < 0 \\ x & otherwise. \end{cases} \tag{2.24}$$

It returns 0 for negative values and the number itself for positive values. Despite looking like a simple activation function, it can significantly improve neural network learning. The reason is that ReLU produces sparse activations, i.e., only a subset of neurons in a layer will be activated, while the rest will be zero [33]. This helps the model avoid overfitting and motivates it to learn more robust features. In addition, ReLU is computationally more efficient compared to sigmoid since it avoids the computationally expensive numerical operations, which are done by using sigmoid or softmax. As mentioned above regarding the problem of vanishing gradient when using sigmoid, the ReLU function manages to overcome this problem. Sigmoid functions produce smaller gradients for more significant inputs, leading to the model not learning. The ReLU activation function has a simple derivative, meaning it has a large gradient over a wide range of input values, making it more effective in computations. The gradient does not vanish during backpropagation.

2.3.4 Loss functions

As discussed in the section of LogReg, the optimal goal of the training is to approximate the $\hat{y}$. We want it to be as close to the actual y and have a minimized *loss*. This procedure uses a loss function, a mathematical function designed for computing the distance between the output of the FFNN and the actual gold label.

Binary cross-entropy loss

Binary cross-entropy (BCE) is a simplified version of the cross-entropy loss function. As the name suggests, the BCE loss function deals with only binary classification problems. This function is based on **conditional maximum likelihood estimation**, i.e., we choose the weight and bias term parameters that maximize the log probability of the actual **y** labels given the observations. Cross-entropy loss is the negative log-likelihood loss. The expression for BCE loss can be easily derived by assuming that we have classes 0 and 1. Since we have only two classes, it follows a Bernoulli distribution, and we can express the probability that the model produces one observation as the following

$$p(y|x) = \hat{y}^y(1-\hat{y})^{1-y} \tag{2.25}$$

Working with small numbers has a big issue; the multiplication of small numbers converges toward zero. A logarithmic operation is usually performed on both sides to solve this problem.

$$\log p(y|x) = y \log \hat{y} + (1-y)\log(1-\hat{y}) \tag{2.26}$$

Inserting a negative sign in front of Equation 2.27 will give us the expression for BCE loss.

$$L_{BCE} = -\log p(y|x) = -\Big[y \log \hat{y} + (1-y)\log(1-\hat{y})\Big] \tag{2.27}$$

Cross-entropy loss

Cross-entropy (CE) loss is the default choice for most classification problems with neural networks. Furthermore, this loss function is commonly used for multi-class classification problems and is a good choice in natural language processing. The difference between BCE and CE loss is that we deal with vectors, not scalars. Therefore, we obtained the following expression for the CE loss for a single example.

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{K} \mathbf{y}_k \log \hat{\mathbf{y}}_k \tag{2.28}$$

2.3.5 Optimization functions

Most machine learning algorithms involve optimization techniques, and optimizing neural networks involves a non-convex optimization problem. The optimization aims to obtain the ideal weights or model parameters while training the neural networks. Furthermore, such optimization minimizes the loss function by altering the model's parameters. Various methods do this. This section will introduce the common choices for the optimization functions.

Gradient Descent (GD)

GD discovers the minimum of a loss function by using the direction in which the loss function's slope rises the most steeply, then moves in the opposite direction. More specifically, by starting with an arbitrary initial weight vector, the algorithm computes a vector W that minimizes the loss function L_{CE}. Then, step by step, it tries to modify the loss. At each stage, the weights vector changes in the direction that produces the most vertical descent along the loss surface. The step-wise process can be expressed mathematically as the following.

$$\theta_{t+1} = \theta^t - \eta \nabla L(f(x;\theta), y) \tag{2.29}$$

The η is the learning rate determining the algorithm's amount of movement. The $\nabla L(f(x;\theta), y)$ is the gradient of the loss function and is given by

$$\nabla L(f(x;\theta), y) = \left[\frac{\partial}{\partial w_1} L(f(x;\theta), y) \text{ ,..., } \frac{\partial}{\partial w_1} L(f(x;\theta), y) \right]^T \tag{2.30}$$

The gradient of a multi-variable function f is a vector in which each component expresses the partial derivative of f with respect to one of the variables.

Stochastic Gradient Descent (SGD) and Mini-batching

Training a neural network on a large dataset can be computationally and time inefficient. The data is typically divided into mini-batches, and the parameters are updated after each mini-batch. Stochastic Gradient Descent (SGD) is a widely-used method for training neural networks on large datasets. It calculates the gradient on a single batch of training data at each iteration, resulting in greater computational efficiency. However, this also means that the changes to the network parameters are often noisy and can fluctuate significantly. To address this issue, the gradient is computed over batches of training instances rather than a single instance.

In contrast, mini-batch training involves computing the gradient on a small batch of training data, with standard batch sizes being powers of two and ranging from 16 to 256 samples. While SGD can be viewed as a specific instance of mini-batch training with a batch size of 1, their convergence characteristics differ. SGD updates have greater variance than other algorithms, which can hinder convergence but also help avoid local minima. Mini-batch training is a trade-off between the stability of batch gradient descent and the efficiency of SGD, making it the most widely used optimization technique for neural network training due to its ability to strike a good balance between speed and stability [40, 51].

SGD optimization often produces promising results, but more elegant and advanced algorithms exist that optimize better. Most of these methods

are based on so-called *adaptive learning rates*, meaning that the optimizer automatically adjusts the learning rate during the training process based on the characteristics of the gradients encountered. The first algorithm that introduced adaptive learning rates was the **delta-bar-delta** algorithm [47]. This algorithm was based on increasing the learning rate if the partial derivative of the loss remains the same sign. Conversely, the learning rate will decrease if the partial derivative changes sign. Throughout recent years, it has been introduced many other adaptive optimizers which are more effective.

AdaGrad

AdaGrad is an adaptive optimization algorithm that relies on stochastic gradient descent (SGD). It adapts the learning rate of each parameter by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient [**AdaGradPaper**]. This algorithm treats the parameters with small partial derivatives of the loss to have a relatively moderate decline in the learning rate. The parameters with the largest partial derivative are treated with a rapid decrease in the learning rate. The central concept here is to adaptively scale the learning rate for each parameter depending on its gradient history, which aids in achieving faster convergence and higher performance in some circumstances, especially for problems with sparse gradients.

AdaGrad uses an adaptive learning rate, meaning that the learning rate for every parameter θ_i differs at every time step t based on the past gradients. At each time step t, the parameters are updated by

$$\begin{aligned} g_t &= \nabla_{\theta_t} J(\theta_t), \\ \theta_{t+1} &\leftarrow \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \end{aligned} \tag{2.31}$$

The $\nabla_{\theta_t} J(\theta_{t,i})$ is the gradient of the loss function with respect to the parameter θ_i at time step t. The $G_t \in \mathbb{R}^{d \times d}$ is the diagonal matrix where each diagonal element i, i is the sum of the squares of the gradients with respect to θ_i and ϵ is a smoothing term that avoids division by zero [84].

Root Mean Square Propagation (RMSProp)

Despite having an adaptive learning rate, AdaGrad struggles with nonconvex settings. The accumulation of squared gradients in the denominator is AdaGrad's direct weakness [84]. The accumulated sum keeps growing during the training since every added term is positive. The result is that the learning rate shrinks until it reaches a point where it becomes tiny, leading to becoming too small before arriving at a convex structure. Root Mean Square Propagation, or RMSProp[10], solves this issue by decreasing

[10]RMSProp is an unpublished method proposed by Geoff Hinton in Lecture 6e of his Coursera Class. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

its learning rate using an exponentially decaying average. The advantage of this is that it discards the history from the extreme past so that it can converge rapidly after finding a convex bowl [35, p.304].

The RMSProp method is based on removing the G_t in the dominator in Equation 2.31 and replacing it with a leaky average [112, p.567]

$$s_t \leftarrow \gamma s_{t-1} + (1-\gamma) g_t^2, \tag{2.32}$$

where the γ is the decay rate and controls the grade of adjusting/scaling. The final weight-updating equation becomes

$$\theta_{t+1} \leftarrow \theta_t - \frac{\eta}{\sqrt{s_t + \epsilon}} \odot g_t \tag{2.33}$$

Adadelta

The AdaDelta is another optimization method based on AdaGrad. Adadelta restricts the window of accumulated past gradients to a fixed size to reduce its bold decreasing learning rate. Like the RMSProp optimization function, Adadelta relies on a leaky average. The difference arises when updating the weights, where Adadelta uses a rescaled gradient [112, p.572]

$$g_t^{'} = \frac{\sqrt{\Delta\theta_{t-1} + \epsilon}}{\sqrt{s_t + \epsilon}} \odot g_t \tag{2.34}$$

where $\Delta\theta_{t-1}$ leaky average of the squared rescaled gradients and is updated by

$$\Delta\theta_t = \gamma \Delta\theta_{t-1} + (1-\gamma) g_t^{'2} \tag{2.35}$$

Finally, the equation for the parameters at time t is updated by

$$\theta_t = \theta_{t-1} - g_t^{'} \tag{2.36}$$

Adam

Adaptive Moment Estimation, or Adam, is yet another optimization function [50]. Adam can be seen as a combination of the RMSProp, and the *momentum* algorithm [80]. The momentum optimization approach employs a momentum vector to consider prior gradients while calculating the current gradient. Assisting models in navigating around loss surface plateaus and ravines hastens convergence in gradient decline.

By using the features of the momentum and the RMSProp, it is possible to combine their strengths and achieve better optimization performance. The central part of Adam is that it uses leaky averaging to estimate both the momentum and the second moment of the gradient [112, p. 575]

$$\begin{aligned} s_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ v_t &\leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t \end{aligned} \tag{2.37}$$

Both β_1 and β_2 are non-negative weighting parameters, often close to 1. In some scenarios, Adam is biased towards zero when both s_t and m_t are vectors of zero and when the decay rates are low [84]. By creating first and second-moment estimates that have been adjusted for these biases

$$\begin{aligned} \hat{s}_t &= \frac{s_t}{1-\beta_2^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_1^t} \end{aligned} \tag{2.38}$$

By using these equations, the weights can be updated by

$$\begin{aligned} g_t^{'} &= \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon} \\ \theta_t &\leftarrow \theta_{t-1} - g_t^{'} \end{aligned} \tag{2.39}$$

The gradient has been rescaled using the momentum $\hat{v}_t$, which is close to the RMSProp method, but RMSProp used the gradient.

2.3.6 Practical techniques

Initialization of Neural Networks

To optimize a neural network, there is a possibility that we might get stuck at a point where we are not able to improve the performance further. This is because the optimization is not straightforward. Starting from different initial points can result in different developments. Therefore, attempting numerous initialization techniques and choosing the most satisfactory technique is recommended. The variability in the results due to various random starting points cannot be predicted and depends on the specific neural network and data.
Constructing a neural network is necessary to decide the parameters for the hidden layers. Some distribution typically initializes the weights. The *Xavier initialization* suggests initializing a weight matrix [32] by

$$\mathbf{W} \sim U\left[-\frac{\sqrt{6}}{\sqrt{d_{\text{in}} + d_{\text{out}}}}, +\frac{\sqrt{6}}{\sqrt{d_{\text{in}} + d_{\text{out}}}}\right] \tag{2.40}$$

where the weight matrix has the dimensions $\mathbf{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ and the U represents the uniform distribution [11, p.161]. *He initialization* is another famous technique [43],

$$\mathbf{W} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{d_{\text{in}}}}\right) \tag{2.41}$$

where $\mathcal{N}$ is the Gaussian distribution [11, p.179]. This method is initialized by sampling from a zero-mean Gaussian distribution and works better than Xavier initialization in an image classification task.

Hyperparameter tuning

Hyperparameter tuning is one of the most important aspects of training NNs. The parameters of a neural network are the weights $\mathbf{W}$ and biases $\mathbf{b}$, which are learned through the optimization algorithm. Optimal values are tuned on an evaluation rather than the training set. In addition, we have some other hyperparameters:

- The number of hidden layers
- The number of hidden nodes per layer
- The choice of activation functions
- The choice of an optimization algorithm
- The learning rate
- The size of mini-batch

Overfitting in Neural Networks

The concept of overfitting is a general issue in machine learning, which occurs when the model parameters are adjusted to fit the training data too closely and can cause the model to capture the noisy samples in the training data. Consequently, an overfitted model will predict the training samples extremely well but incorrectly on new, unseen samples.

One of the most effective methods to prevent overfitting in Neural network (NN) is to deploy the *dropout* strategy during the training [45, 93]. The main idea of using dropout is to prevent the network from relying too much on specific weights. The dropout, as the name suggests, is based on dropping out of some of the nodes using the *Bernoulli* distribution [11, p.98]. The benefit of this distribution is that it randomly returns 0 and 1, which enables the network to cancel out some of the nodes. More specifically, the dropout is applied in each layer after the activation layer [34, p. 48]:

$$\begin{aligned} \mathbf{z} &= \mathbf{W}\mathbf{x} + \mathbf{b} \\ \mathbf{a} &= \sigma(\mathbf{z}) \\ \boldsymbol{m} &\sim \text{Bernouli}\,(r) \\ \mathbf{h} &= \mathbf{a} \odot \mathbf{m} \\ \mathbf{v} &= \mathbf{U}\mathbf{h} \\ \hat{\mathbf{y}} &= q(\mathbf{v}) \end{aligned} \tag{2.42}$$

where m is *masking vectors* and r is the parameter of the Bernoulli distribution.

Early stopping is another strategy to control overfitting in neural networks during training. The fundamental idea is to observe the validation loss

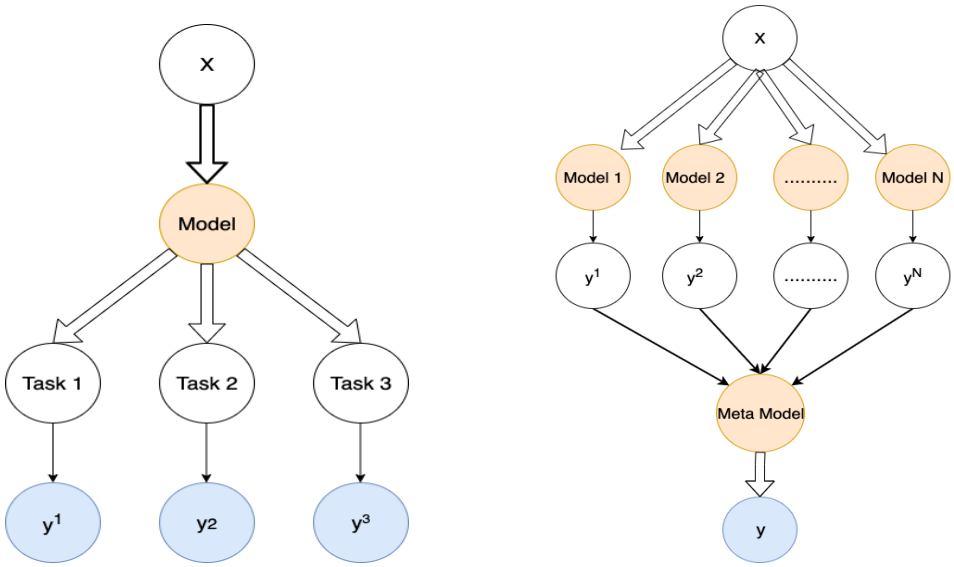


Figure 2.5: a) This figure illustrates the common situation where the tasks share a common input but involve different target variables. This approach is based on training one model to solve multiple *tasks*. b) The simplest scheme of an ensemble method, combining N models to decrease the variance of the model and obtain better performance.

while training the model and stop the training process before the model overfits the training data. The validation loss is calculated after each training epoch, and the training process is stopped when the loss quits enhancing after *patience criteria*, which is some small amount ϵ number of epochs. The motivation behind early stopping is that the validation loss will typically decrease during the initial training steps as the model learns to generalize. However, after some epochs, the validation loss may increase as the model overfit. Therefore, we may avoid the model overfitting by ending the training process before this occurs, enhancing the model's capacity to generalize to new, untried data. Another advantage of early stopping is the time saved on an *accelerator*, which may improve generalization in the case of noisy labels.

2.3.7 Advanced Learning Techniques

In this section, we will discuss two new advanced learning concepts that will help the learning of neural networks and make them generalize better.

Ensemble Learning

Till now, we have discussed the standard approach to model selection which consists of training a bunch of models based on the hyperparameters and evaluating each of these on the development set. Then, the model with the best evaluation metric is chosen, and the rest are discarded. However,

this approach does not work well in practice, and the model performing well on the development dataset does not necessarily perform well on the held-out dataset. Neural networks are a nonlinear method, meaning they can learn complex and nonlinear relationships from the input data. The disadvantage is that these models are liable to initial conditions, such as the terms of the noise in the training dataset, which may lead to overfitting. Generally, the neural networks have high variance and low bias, which is strange given that most of the NNs are trained on large datasets.

An effective way to tackle the very high variance is to deploy *ensemble* methods, also known as *ensemble learning* [38]. These methods are based on combining multiple models to obtain better performance and have been shown to perform better than any single classifier [24]. The difference in the models can be the model architecture, the use of different training data, different training regimes, different initialization schemes, etc. There are several methods for combining their predictions, such as *boosting* [30] and *bootstrap aggregating* [14].

The models, or the base learners, are generated sequentially in the boosting algorithm, where each model is trained to correct the errors of the previous model. The ensemble is built incrementally by training each model to emphasize the training instances misclassified by the previous models. The predictions are combined through a weighted majority vote. On the other hand, Bootstrap aggregating (bagging) is based on generating the base learners independently. This method creates multiple instances of the training data by sampling with replacement and training a separate model for each. The predictions are combined by voting.

Multi-task Learning

Multi-task learning (MTL), almost the opposite of ensemble learning, is based on training one model to solve multiple tasks [15]. Each task has its loss function, but the model weights are shared. Multi-task learning is usually deployed for closely related tasks such as *PoS*-tagging in Natural Language Processing (NLP) [72]. Similarly, multi-task learning has been applied to considerably fake news and misinformation detection tasks since most of these consist of closely related tasks [54, 57, 110]. As figure 2.5(a) shows, the parameters in the upper part are generic and shared across all the tasks, and the task-specific parameters are those in the figure's bottom. The multi-task learning method is built based on the assumption that a common collection of factors explain the variations in the input features. Each task is associated with a subset of these factors. This is why the MTL results in better generalization. Of course, this will happen only if some assumptions about the statistical relationship between the different tasks are valid, meaning that something is shared across some tasks.

2.4 Transformer

Recently, a novel model called Transformer has emerged, addressing numerous challenges in natural language processing [99]. This model has quickly gained traction and is now extensively utilized in the NLP community. Transformers have shattered multiple NLP records, advancing the state-of-the-art in various applications, such as machine translation, conversational chatbots, sentiment analysis, and more. The Transformer architecture relies exclusively on attention mechanisms to establish global dependencies between input and output, effectively transforming the landscape of NLP. A crucial aspect of text classification is ensuring that machines can comprehend the content. To achieve this, models must consider previous and subsequent inputs concerning the current input. Traditional recurrent neural networks (RNNs) have been effective in this task but are hindered by short-term memory limitations. This constraint means that RNNs can only consider a limited number of previous inputs when processing a current word. The Transformer addresses this issue by considering the entire text input using the attention mechanism. This enables Transformer to utilize the full context of a text when making predictions. This model comprises an encoder-decoder architecture. At a high level, the encoder maps an input sequence into a continuous abstract representation containing the learned information from the input. The decoder then processes this representation, generating a single output step by step while incorporating information from previous outputs. In this section, we dive further into the steps of the Transformer and explain how it works. Furthermore, we will present the most common types of Transformer, *BERT* and *RoBERTa*. Finally, we will also present the domain-specific Transformer model for the task of COVID-19-related conspiracy detection on Twitter.

2.4.1 The steps of the Transformer

In this section, we will describe each step of the architecture. Note that name of each step is related to Figure2.7.

The first step: Input Embedding involves passing the input through a word embedding layer. This layer is a lookup table to obtain a learned vector representation for each word. Since neural networks process information numerically, each word is mapped to a continuous-valued vector that captures its representation.

The second step: Positional Encoding involves incorporating positional information into the embeddings. Since the Transformer encoder layer does not contain recurrent networks, adding positional information to the embeddings is essential. This is achieved using positional encoding. The makers of Transformer devised an innovative method using the following functions:

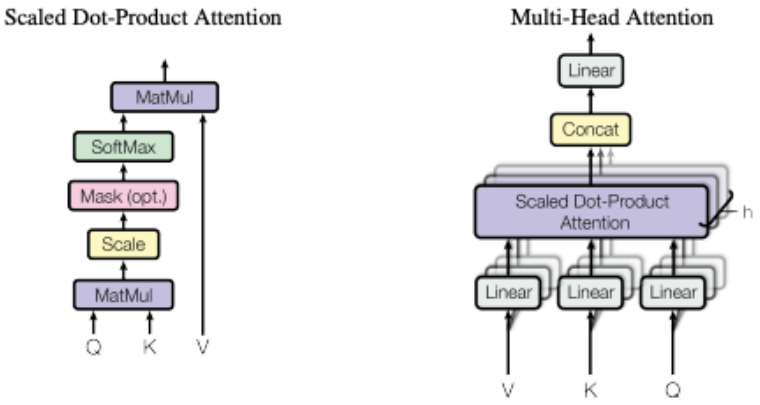


Figure 2.6: **Left**: Scaled dot-product attention computes the dot product of query and key matrices, scales it by the square root of the key dimension, and applies the softmax function to obtain weights. These weights are then used to compute the weighted sum of the value matrix. **Right**: Multi-Head Attention performs parallel attention computations, called *heads*, over different linear projections of the input queries, keys, and values.

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{dmodel}}}\right) \tag{2.43}$$

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{dmodel}}}\right) \tag{2.44}$$

These functions generate vector representations based on whether a word is in an odd or even time step. These vectors are then combined with the embedding vectors through an addition operation. Consequently, this process facilitates the flow of information about word positions, helping the network understand the input order.

The third step: Encoder layer This layer, displayed as the left block of Figure 2.7, maps all input sequences into a continuous abstract representation that holds the learned information for that entire sequence. This layer contains two sub-module; *multi-head attention* followed by a feedforward network, as shown in Figure 2.6. There are also residual connections around each sub-module, followed by a layer normalization. The multi-head attention module applies a specific attention mechanism called *self-attention*. This mechanism allows the model to associate each word in the input with other words. To achieve self-attention, we feed the information into three distinct fully-connected layers to create the *query* (Q), *key* (K), and *value* (V) vectors. These vectors are then driven through the following equation to achieve attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.45}$$

The query and key vectors undergo a dot product matrix multiplication to produce a scoring matrix. The higher score, the more the focus, and in this way, the queries are mapped to keys. Then the scores get scaled down by dividing by the square root of the dimension of the queries and the keys. This allows for more stable gradients, as multiplying values can have exploding effects. Next, these values are driven through a softmax function to get the attention weights. By doing the softmax, the higher scores get heightened, and the lower scores get depressed, allowing the model to be confident about which words to attend to. These attention weights are then multiplied with the value vector to an output vector.
The final output vector is then driven through a linear layer to process. To make this a multi-head attention computation, we must split the query, key, and value vectors into adding vectors before applying self-attention. The following equation does this.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.46}$$

where

$$\text{head}_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

These split vectors go through the same self-attention process individually. Each self-attention process is called a head, and each head's output vector gets concatenated into a single vector before going through the final linear layer.

The output vector of the multi-head attention module is added to the original input. This is called residual connection. The output of the residual connection goes through layer normalization. The normalized residual output gets fed into a pointwise feed-forward network for further processing. The pointwise feed-forward network is a couple of linear layers with a ReLu activation function in between the layers. The output of this feed-forward model is attached to the input of the same model. The residual connections help the network train by allowing gradients to flow through the networks directly. The layer normalization stabilizes the network, substantially producing the necessary training time. Finally, a pointwise feed-forward layer is used to process the attention output further, potentially giving it a richer representation.

The fourth step: Decoder layer The main job of the decoder, the right block of Figure 2.7, is to focus on the appropriate words in the input during the decoding process. We can stack the encoder N times to further encode the information where each layer can learn different attention representations. Therefore, potentially boosting the predictive power of the Transformer network. The decoder has the types of modules as the encoder. It has one

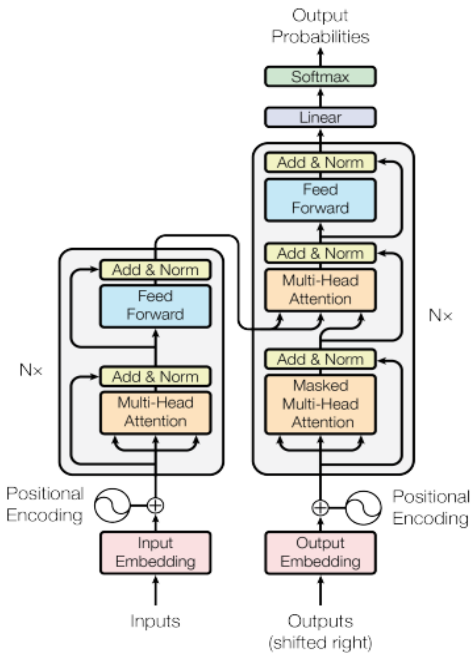


Figure 2.7: The Transformer model's architecture: an encoder and a decoder, which contain multiple layers of self-attention and feedforward neural networks.

Masked multi-head attention, one multi-head attention, a pointwise feed-forward layer with residual connections, and layer normalization after each sub-layer. These sub-layers behave similarly to layers in the encoder, but each multi-head attention layer has a different task. It is capped off with a linear layer that acts as a classifier and a softmax function to get the word probabilities. The decoder is autoregressive; it takes in the list of previous outputs as inputs, and the encoder outputs contain the attention information from the original input. The decoder stops decoding when it generates an end token as output.

The input goes through an embedding layer in a positional encoding layer to get the positional embeddings. The positional embeddings get fed into the first multi-head attention later, which computes the attention scores for the decoder's input. This multi-headed attention layer operates slightly differently since the decoder is autoregressive and generates the sequence word-by-word; we need to prevent it from being conditioned into future tokens. Therefore, we apply a look-ahead mask to prevent the decoder from looking at future tokens. The mask is added before computing the softmax and after calling the scores. In this case, the mask is a matrix the same size as the attention scores filled with values of zeros and negative infinities. Taking the softmax of this matrix removes the scores for the future words and leaves us scores for the previous words. This masking is the only difference in how the first multi-head attention is in the decoder.

Other than that, we still have multiple heads to which the masks are being applied before getting concatenated and fed through a linear layer for further processing. The output of the first multi-head attention layer is a masked vector with information on how the model should attend to the inputs of the decoder.

In the second multi-head attention, the output of the encoder is the queries and the keys, and in the first multi-headed attention layer, outputs are the values. This process matches the input of the encoder to the input of the decoder allowing the decoder to decide which encoder input is relevant to put focus on. The output of the second multi-headed attention goes through a pointwise feed-forward layer for further processing. The output of the final pointwise feed-forward layer goes through a final linear layer that accesses a classifier. The classifier is the most significant number of classes you have defined. The output again gets fed into a softmax layer. The softmax layer produces probability scores between 0 and 1 for each category, and the class with the highest probability gets picked out. The decoder can be stacked in N layers, where each layer takes inputs from the encoder and the layers before it by stacking layers. The model can learn to extract and focus on different combinations of attention from its attention heads, potentially boosting its predictive power.

2.4.2 The BERT model

Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained language model based on the transformer architecture [23]. The Transformer was initially created for the task of neural machine translation. The encoder tries to understand the sentence and the words. The decoder aims to generate an output sequence that maintains the linguistic coherence and context required for the given task. Both of these units have some understanding of a language; these can be separated, and we can build different models. The *GPT* model is based on stacking the decoder modules [82], while the BERT is based on a stack of encoders [23]. BERT considers both the left and right context of each word in a sentence, while other transformer-based models may be unidirectional, meaning they only consider each word's left or right context. In contrast, BERT employs bidirectional self-attention, and the GPT Transformer uses constrained self-attention where every token can only attend to the context to its left. This bidirectional approach allows BERT to capture a broader understanding of the context.

Pre-training of BERT

There are two main steps in the BERT framework; pre-training and fine-tuning. The pre-training of BERT is done by training the model on unlabeled data over different pre-training tasks. Two unsupervised learning tasks have accomplished the pre-training of BERT; *masked language model*

(MLM) and *next sentence prediction* (NSP). MLM is based on randomly masking a percentage of the input tokens, simply replacing the words with a [MASK] token. The goal is to predict the original vocabulary id of the masked word based on its context. The 15 percent of all tokens have been preprocessed, where 80 percent are masked, 10 percent are left unchanged, and a randomly selected vocabulary token replaces 10 percent. On the other side, the goal of the NSP task is to train a model that understands sentence relationships. A binary classification did this, choosing two sentences *A* and *B*; the goal was to predict whether sentence *B* follows sentence *A*.

The architecture of BERT is based on a multi-layer bidirectional transformer encoder based. The common sizes of BERT are:

- BERT $_{\text{BASE}}$ (L = 12, H = 768, A = 12, Total Parameters=110M)
- BERT $_{\text{LARGE}}$ (L = 24, H = 1024, A = 16, Total Parameters=340M)

Note that the *L* stands for Transformer blocks, *H* stands for hidden size and *A* stands for numbers of attention heads. The pre-training was done using data from the BooksCorpus (800M words) [116] and English Wikipedia (2,500M words), which totals 16GB of uncompressed text.

Fine-tuning

During the fine-tuning, the BERT model is started with the parameters learned from the pre-training. Then, these parameters are fine-tuned using labeled data from specific tasks. Fine-tuning is simple since the self-attention mechanism in the transformer allows BERT to model many downstream tasks by swapping out the appropriate inputs and outputs. The updated parameters for each task have separated fine-tuned models even though they were initialized with the same pre-trained parameters.

2.4.3 The RoBERTa model

Robustly Optimized BERT Approach (RoBERTa) is an extension of the BERT model [61]. The authors of RoBERTa intended that the BERT was significantly undertrained and, therefore, proposed suggestions to improve the BERT model. The most significant differences between these models are pre-training corpus, training duration and strategies, and masking strategy.

Larger pre-raining copus BERT is pre-trained on the corpus of text data primarily sourced from the BooksCorpus and Wikipedia. On the other hand, RoBERTa was pre-trained on a much larger and more diverse corpus of text data, which totaled over 160GB of uncompressed text, while the training data of BERT was at a total of 16GB. The RoBERTa is also trained on the BooksCorpus. However, in addition, it was trained on the *CC-NEWS*, which was collected from the English portion of the CommonCrawl News dataset [**CommonCrawl-News-dataset**], *OpenWebText* which is an open-source recreation of the WebText corpus[**OpenWebText**] and *Stories* which

is a dataset containing a subset of CommonCrawl data filtered to match the story-like style of Winograd schemas [**Stories**].

Training procedure RoBERTa modifies the pre-training process used by BERT in several ways, including *dynamic masking*. In the original BERT model, the input text is randomly masked during pre-training, and the model learns to predict the masked words based on the context delivered by the surrounding words. In RoBERTa, a different masking strategy is used, where the text is masked dynamically during pre-training. BERT uses a static masking strategy, where the masked tokens stay the same throughout training. The model has to predict the original word based on the context provided by both the surrounding words and the unmasked words within the same training example. Secondly, there was no NSP process. RoBERTa removes this task, as researchers found it did not contribute significantly to downstream performance. Lastly, the hyperparameter optimization was done differently. The RoBERTa employs a more exhaustive search for optimal hyperparameters than BERT. This includes larger batch sizes, longer training (more steps), and adjustments to the learning rate. These training strategies help improve the efficiency and effectiveness of the pre-training process, allowing RoBERTa to learn more detailed and nuanced language representations.

Performance Due to these differences, RoBERTa generally outperforms BERT on various natural language understanding benchmarks like GLUE, SuperGLUE, and SQuAD. In addition, the improvements in RoBERTa lead to better performance on downstream tasks, such as sentiment analysis, question-answering, and named entity recognition.

2.4.4 Domain-specific pretrained models

Fake news and conspiracy detection are essential topics, which is why it has been developed and pre-trained various models that make these types of detection easier. In our case, the unique dataset is based on COVID-19-related tweets. Firstly, it has a lot of COVID-19-related abbreviations which can only be understood by pre-trained language models that are pre-trained on similar datasets. Secondly, the datasets contain tweets with typically short lengths, frequent use of informal grammar, and irregular vocabulary, e.g., abbreviations, typographical errors, and hashtags. Therefore this might lead to a challenge in applying existing language models pre-trained on large-scale conventional text corpora with formal grammar and regular vocabulary to handle text analytic tasks on Twitter data. To tackle both of these challenges, we are obligated to deploy domain-specific pretrained models. Therefore, we are interested in models pre-trained on large-volume datasets on general and COVID-19-related tweets. This section will present the most popular pre-trained models trained on COVID-19-related tweets. Most of these models are based on bidirectional Transformer models such as BERT and RoBERTa.

COVID-Twitter-BERT

CT-BERT[11] is a pre-trained model that was trained on 22.5 million unique COVID-19 related tweets, which resulted in 0.6 billion words [69]. The base model on which CT-BERT is based is the BERT-Large uncased. Note that the BERT-Large has two different versions. The first one is trained with text and is processed in its original case, meaning that uppercase and lowercase letters are preserved. Reversly, the second is uncased, where all text is converted to lowercase before processing. This means that the distinction between uppercase and lowercase letters is lost, and the model would treat words such as "Car" and "car" as the same token. As mentioned in the sections above, the BERT is trained on Wikipedia articles and book corpus, and it still contains little information about any domain-specific knowledge about COVID-19 or tweets.

The CT-BERT is trained on a corpus of 160M tweets related to COVID-19 collected between January 12 to April 16, 2020. Before training, the original corpus was cleaned for retweet tags. Furthermore, each tweet was driven through a preprocessing process that included replacing all Twitter usernames with a common text token, the URLs with a common URL token, and all emoticons with textual ASCII representations. Finally, the retweets, duplicates, and closely duplicate tweets were removed from the training data. These abbreviations of the data resulted in almost 1/7 of the training data of BERT. Furthermore, the training process was carried out with a distributed training system on a TPU v3-8 (128GB of RAM) for 120 hours.

According to the authors of CT-BERT, the re-training, or domain-specific training, of BERT-Large resulted in improvements in previous datasets. The CT-BERT overtook the performance of the BERT-Large in various tasks such as Twitter Sentiment SemEval (SE) [70], Stanford Sentiment Treebank 2 [92], and additional COVID-19 classification tasks. In addition, CT-BERT has outperformed the BERT-Large, and RoBERTa-Large models on other COVID-19-related classification problems [31].

BERTweet

BERTweet[12] is a pre-trained language model for English Tweets, which is based on the original BERT model but has been trained with the RoBERTa pre-training procedure [71]. According to the authors, BERTweet was the first large-scale language model for English Tweets, trained on 850 million tweets (16 billion tokens).

The training dataset of BERTweet is a concatenation of two corpora. The first is a 4 TB of tweets streamed from January 2012 to August 2019 on

[11]The CT-BERT model is available at Hugging Face: https://huggingface.co/digitalepidemiologylab/covid-twitter-bert-v2

[12]The BERTweet is available at Hugging Face. Note that BERTweet has a general and a COVID-19-related model. All of them are available here: https://huggingface.co/vinai

Twitter and has been preprocessed through the same techniques as for CT-BERT; converting usernames and URLs into their respective standard token, converting emojis into their textual meaning and filtering out retweets. Additionally, tweets with less than ten tokens and more than 64 tokens were removed. This preprocessing technique resulted in the first corpus of 845M English Tweets. The second was streamed from January 2020 to March 2020 and was explicitly related to COVID-19. This dataset ended up with 5 million tweets after the processing techniques. In total, BERTweet was trained for four weeks on 8 V100 GPUs. Similar to CT-BERT, the authors of BERTweet show that the BERTweet model outperforms models like RoBERTa-Base on various NLP tasks such as POS-tagging, named-entity recognition, and text classification.

Chapter 3

The datasets

3.1 The COVID-19 Conspiracy dataset

The COCO is a manually annotated dataset concerning conspiracy categories. The dataset was proposed by Langguth et al. [52] and contains twelve categories, but due to the limited samples, we have decided to work with nine of these categories. Despite category descriptions in the paper, we include them here for clarity and to ease the discussion in the following chapters.

1) **Suppressed Cures** This category collects narratives proposing that effective medications for COVID-19 were available but whose existence or effectiveness has been denied by authorities, either for financial gain by the vaccine producers or some other harmful harm intent, including ideas from other conspiracy categories listed below.

2) **Mind Control and behavior control** In this category, we collected narratives containing the idea that the pandemic is being exploited to control the behavior of individuals, either directly through fear, through laws that are only accepted because of fear, or through techniques which are impossible with today's technology, such as mind control through microchips.

3) **Anti Vaccination** We collect all statements suggesting that the COVID-19 vaccines serve some hidden nefarious purpose or can kill people to control the population numbers. This category does not include concerns about vaccine safety or efficacy or concerns about the trustworthiness of the products since these are not conspiracies, even though they may contain misinformation.

4) **Fake Virus** One of the prominent narratives early in the pandemic was that there is no COVID-19 pandemic and that reports about it are meant to deceive the population either to hide either death caused by 5G wireless equipment or later by vaccines or that the pandemic is just an over-dramatization of the annual flu season with the intent of controlling the behavior of the population.

5) **Intentional Pandemic** This straightforward narrative posits that the pandemic results from purposeful human action pursuing some illicit goal. The actor and the goal vary widely, but they all share the ability to produce a perceived culprit for the situation. Note that this is distinct from the assertion that COVID-19 is a bio-weapon or was created in a laboratory since this does not preclude the possibility of it being released accidentally.

6) **Harmful Radiation or Influence** This class of conspiracy theories bundles all ideas that connect COVID-19 to wireless transmissions, especially from 5G equipment. This was done by claiming, for example, that 5G is deadly and that COVID-19 is a cover-up or that 5G allows mind control via microchips injected into the bloodstream. As 5G misinformation has already been studied separately, it was not the focus of this dataset.

7) **Population Reduction** Often linked to statements by Bill Gates, conspiracy theories on population reduction or population growth control suggest that either COVID-19 or vaccines are used to reduce population size by killing people or rendering them infertile. In some cases, this is directed against specific ethnic groups. These narratives often use the term" population control" in the sense of population size control, which must be distinguished from population behavior control covered in other conspiracy theories.

8) **New World Order (NWO)** is a preexisting conspiracy theory that deals with the secret emerging totalitarian world government. In the context of the pandemic, this usually means that COVID-19 is being used to bring about this world government through fear of the virus, taking away civil liberties, or some other implausible ideas such as mind control.

9) **Satanism** This category collects narratives in which the perpetrators are some kind of Satanists. Such conspiracy theories may involve harm or sexual abuse of children, such as the idea that global elites harvest Adenochrome from children. Many of these ideas predate COVID-19, but they have been reinterpreted in the new context of the pandemic.

Each category has three subcategories, meaning the total number of distinct categories is 27 (see Table 3.2). These three subcategories are:

- **Promotes/Supports Conspiracy** class contains all tweets that promote, support, claim, insinuate some connection between COVID-19 and various conspiracies, such as, for example, the idea that 5G weakens the immune system and thus caused the current coronavirus pandemic; that there is no pandemic and the COVID-19 victims were harmed by radiation emitted by 5G network towers; ideas about an intentional release of the virus, forced or harmful vaccinations, the

vaccine contains microchips, or the virus is a hoax, etc. The crucial requirement is the claimed existence of some causal link.

- **Discusses Conspiracy** class contains all tweets that mention the existing conspiracies connected to COVID-19 or negate such a connection negatively or sarcastically.
- **Non-Conspiracy** class contains all tweets not belonging to the previous two classes. Note that this also includes tweets that discuss the COVID-19 pandemic itself.

3.1.1 Data Exploration

The training dataset consists of 1913 tweets, while the test dataset consists of 830 samples. The dataset is heavily unbalanced regarding the number of samples per class, reflecting the distribution of tweet topics and people's opinions. Figure 3.1 demonstrates the class distribution for each category, which shows that all categories contain a low percentage of 2's and 3's. Most samples are 1's, whereas in most cases, the percentage of 1's is more than 90 percent.

The dataset's top 10 bigrams and trigrams can be seen in Table 3.1. Some N-grams, such as "bill gate" and "new world order", could be essential for classifying the conspiracies. Based on this, it is crucial to consider N-grams when constructing the TF-IDF and **BOW!** (**BOW!**) approaches.

((a))

Bi-grams	
deep state	215
world order	101
new world	100
bill gates	97
population control	78
corona virus	56
false flag	45
covid 19	41
covid vaccine	40
mark beast	33

((b))

Tri-grams	
new world order	99
population control bill	8
bill gates wants	7
qr code system	6
deep state conspiracy	6
soros bill gates	6
create new world	5
one world government	5
new covid cases	5
bring population control	5

Table 3.1: Top 10 most common bigrams and trigrams in the training data of COCO dataset. These sequences have been generated after the removal of stopwords.

3.2 Evaluation metrics

The dataset is highly imbalanced, as shown in Figure 3.1, and therefore we must consider a powerful evaluation metric when building machine

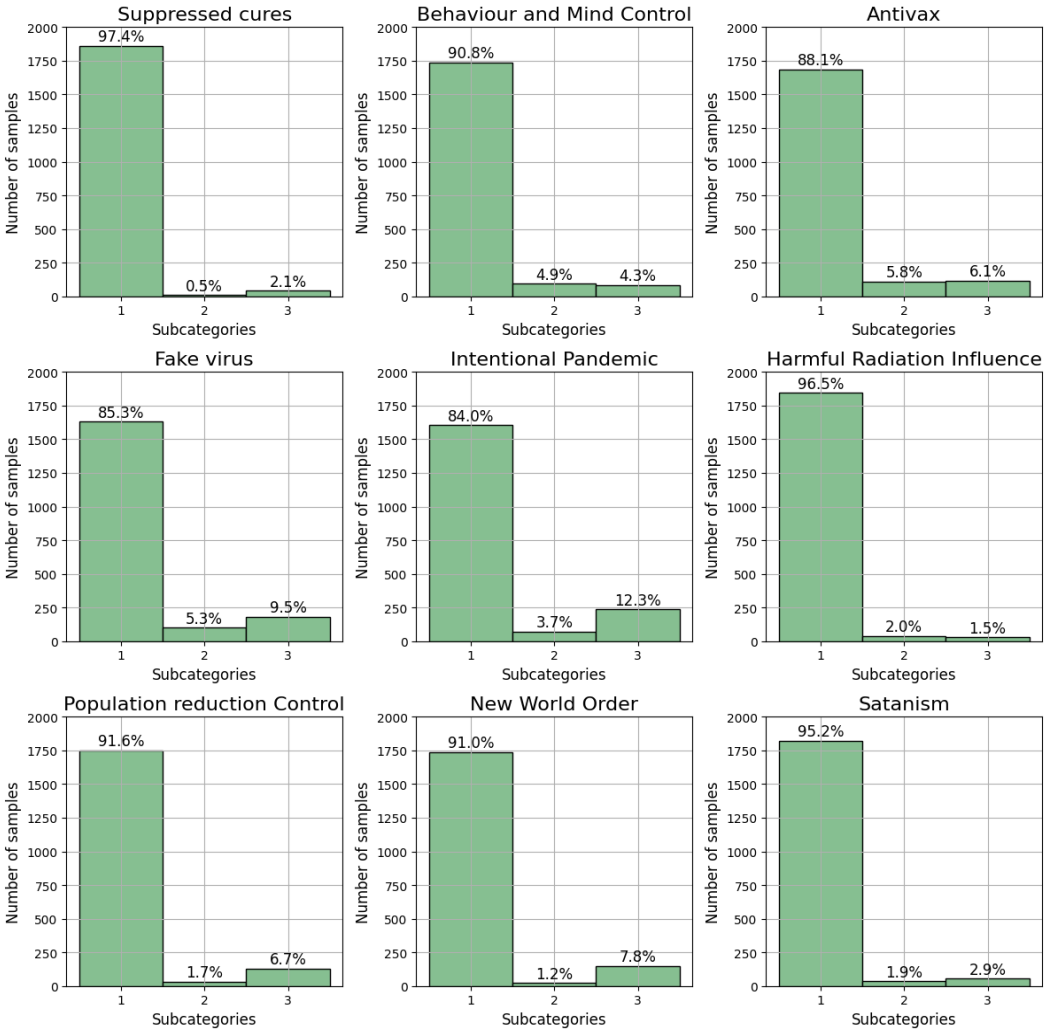


Figure 3.1: The class distribution of the nine conspiracy categories demonstrates the highly imbalanced number of samples per class in COCO dataset. Note that the 1's, in the x-axis, stands for **Non-Conspiracy**, 2's stands for **Discusses Conspiracy** and 3's stands for **Promotes/Supports Conspiracy**.

learning models on this dataset. The evaluation metric has to be insensitive to the imbalance of the categories. A solution to this is the Matthews correlation coefficient (MCC), first introduced in 1975, which is suited for multi-class classifiers for balanced and imbalanced datasets [36]. The reason why MCC is a good evaluation metric is that it constructs a high score when the prediction obtains good results concerning all four categories of the confusion matrix, such as the true positives (TP), false negatives (FN), true negatives (TN), and false positives (FP) [17]. Furthermore, it is proportionally both to the size of positive and negative elements in the dataset. The equation of MCC is given by

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FN)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.1}$$

Note that Equation 3.1 has some excellent properties, such as if the model predicts wrong all of the time, i.e., $TP = TN = 0$, the output becomes -1. While, in the case of a perfect model, where $FP = FN = 0$, Equation 3.1 outputs a value of 1. In addition, the 0 indicates that the model is no better than a random flip of a fair coin. MCC is also perfectly symmetric, so no class is more important than the other; switching the positive and negative will still get the same value.

Alternative to the MCC is the *F1-score*, a popular choice for most classification problems, and is the harmonic mean of *precision* and *recall*. The general expression of the F1-score is given by

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.2}$$

Precision is the proportion of true positives (TP) out of all detected positives, and recall is the number of true positives (TP) correctly classified. More specifically, they are defined as

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$

In a general setting, there exist different variants of the F1-score. Some popular examples are *micro*, *macro*, and *weighted* F1-scores. The micro version computes F1 scores considering total TP, TN, and FB and uses Equation 3.2 directly. On the other hand, the macro version computes F1 scores for each class and then computes their total average to represent the overall score. Note that the micro version does not consider the proportion for each label in the dataset. The weighted version modifies the micro version by computing F1 scores for each class and then averaging them by considering the proportion for each class in the dataset. Although the micro and weighted versions of the F1-score can be good evaluation metrics, both neglect the true negatives (TN) when computing the scores, which can be inaccurate when working with an imbalanced dataset.

In summary, the MCC and F1 scores serve as performance metrics for classification tasks, yet their distinct characteristics make one more appropriate than the other in specific situations. In some instances, the MCC score is considered superior to the F1 score because it accounts for all four confusion matrix values (true positives, false positives, true negatives, and false negatives). In contrast, the F1 score only considers true positives, false positives, and false negatives. Consequently, the MCC score offers a more balanced assessment of a classifier's performance, particularly in imbalanced datasets. Furthermore, with a range from -1 to 1, the MCC score allows for a more straightforward interpretation than the F1 score, which ranges from 0 to 1. Additionally, the MCC score is more sensitive to the classifier's performance across positive and negative classes, unlike the F1 score, which can be biased toward a specific class.

3.3 The alternative versions of the dataset

The dataset, introduced in the section above, consists of 9 categories, each containing three subcategories. The total number of classes is 27, which can be pretty significant. In this section, we will introduce two new versions of the dataset, which can be taught as a fine-grained version of the dataset. These new dataset versions are based on the same dataset, but we are feature-engineering the categories.

3.3.1 Misinformation Detection

We call the first version of dataset *misinformation detection* dataset. This version has converted all nine conspiracy categories into a single category called *misinformation*, and the goal of this dataset is to train and predict various COVID-19 Conspiracy theories by building a multi-class classifier that can flag whether a tweet promotes/supports or discusses at least one (or many) of the conspiracy theories. An example of this transformation can be:

$$1, 1, 3, 1, 3, 1, 1, 1, \text{tweet-text} \implies 3, \text{tweet-text}$$
$$1, 1, 1, 1, 1, 1, 2, 1, \text{tweet-text} \implies 2, \text{tweet-text}$$
$$3, 1, 1, 1, 3, 1, 1, 1, \text{tweet-text} \implies 3, \text{tweet-text}$$
$$1, 1, 1, 1, 1, 1, 1, 1, \text{tweet-text} \implies 1, \text{tweet-text}$$

Note that the $1, 1, 3, 1, 3, 1, 1, 1$ corresponds to the nine conspiracy categories in the order as mentioned in Section 3.1. Therefore, all data samples with at least one 2 or 3 value among the nine categories have been transformed into 2 or 3.

3.3.2 Conspiracy Recognition

We call the second version of dataset *conspiracy recognition* dataset. This dataset aims to offer a recognition dataset that can be used to build a detector that can detect whether a text in any form mentions or refers to any predefined conspiracy topics; in other words, detect the 2's and the 3's.

This dataset was constructed by recasting these, 2's and 3's, into values of 1's and the original 1's into 0's. An example of this transformation is:

$$1,1,1,1,1,1,1,1,1,\text{tweet-text} \implies 0,0,0,0,0,0,0,0,0,\text{tweet-text}$$
$$2,1,1,1,1,1,1,1,1,\text{tweet-text} \implies 1,0,0,0,0,0,0,0,0,\text{tweet-text}$$
$$1,1,1,1,3,1,1,1,1,\text{tweet-text} \implies 0,0,0,0,1,0,0,0,0,\text{tweet-text}$$
$$1,1,1,1,3,1,1,1,2,\text{tweet-text} \implies 0,0,0,0,1,0,0,0,1,\text{tweet-text}$$

Labels	**Tweet Text**
1,1,1,3,1,1,1,1,1	COVID-19 is a hoax, it's just the flu.
3,1,1,1,1,1,1,1,1	Drinking bleach or other disinfectants can cure COVID-19...
1,1,1,1,1,3,1,1,1	5G mobile networks spread COVID-19.
1,1,1,1,1,2,1,1,1	no, I don't think that 5G mobile networks spread COVID-19.

Table 3.2: Illustration of the data format in the COCO dataset. The **Labels** consists of nine integers, each representing the nine conspiracy categories. Their order is identical to the numbering in Section 3.1. Note that the text samples shown here are made-up, not real tweets.

3.4 Other COVID-19 related datasets

This section introduces other datasets that tackle the fake news and conspiracy related to the COVID-19 pandemic. According to the author's knowledge, three openly available datasets exist regarding this topic. Typical for these datasets is that they are extracted from social media platforms or fact-checking websites and are more or less related to the type of language and content.

3.4.1 Fighting an Infodemic: COVID-19 Fake News Dataset

The COVID-19 Fake News Dataset was created in 2021 with the aim of *fighting an infodemic* [74]. This dataset consists of 10700 text samples obtained from social media and articles labeled as real and fake news on COVID-19. The social media platforms used for the data collection were selected to be the ones actively used for social networking for counterpart communication and forwarding details, such as news, events, social phenomenon, etc. Real news and false statements on COVID-19 subjects that appeared on social media were collected. Fake claims were collected from various fact-checking websites, while the real news was collected from Twitter using verified Twitter handles. The creators of this dataset collected the data using the criteria of language that had to be English, and the content had to relate to COVID-19. The most frequent words in this dataset are:

- **Fake** coronavirus, covid19, people, will, new, trump, says, video, vaccine, virus.

- **Real** covid19, cases, new, tests, number, total, people, reported, confirmed, states.
- **Combined** covid19, cases, coronavirus, new, people, tests, number, will, deaths, total.

More details can be found in the dataset's paper [74].

3.4.2 The COVID-19 Category dataset

The COVID-19 Category dataset[1] is developed by the team of CT-BERT [69] and is a sub-sample of the data used for training CT-BERT. Like the rest of the training data, this sub-sample was collected between January 12 and February 24, 2020. This dataset is a binary set with the following labels

- **category_personal** describing a tweet text as a personal narrative.
- **category_news** describing a tweet text as actual news.

The annotation of the dataset was accomplished with *Amazon Mechanical Turk* service [2]. The final class distributions ended with 33.3 % (personal narrative) and 66.7% (news). However, the dataset is available in the form of tweet IDs and their labeled class, meaning to use the data, one must use the Twitter API to gather the tweets based on the IDs. Therefore, *Tweepy* [3], which is a Python library for accessing the Twitter API, was deployed to gather the tweets. Unfortunately, the majority of these tweets were removed and did not exist. In total, 3888 of 5157 was removed from Twitter.

[1]The dataset is available here: https://github.com/digitalepidemiologylab/covid-twitter-bert/tree/master/datasets/covid_category

[2]Amazon Mechanical Turk: https://www.mturk.com/

[3]https://www.tweepy.org/

Chapter 4

Experiments with classical approaches

In this chapter, we explore classical approaches, including Bag-of-Words (BOW), TF-IDF, and N-grams, to establish baseline models that will be compared with Transformer-based models in the subsequent chapter. Our primary objectives are to:

- Assess the performance of these classical approaches on the dataset.
- Identify the conspiracy categories that are most easily predicted using simple, non-neural text representation techniques.

We begin by outlining the experimental methodology, followed by a detailed presentation of the experiments conducted using all methods. Finally, a comparative analysis and discussion of the results are provided at the end of the chapter.

4.1 Experimental setup

This section contains the elements and descriptions of all the tools used to perform the experiments. In addition, we have the software, hardware, and various methods used among the tools.

4.1.1 The software

The programming language used in the whole thesis is Python. However, the version of Python differs from different chapters because of dependencies in various Python packages. In this chapter, we have used the version of 3.11.2[1]. According to the documentation, this version of Python is believed to be 10-60% faster than Python 3.10, and on average, we measured a 1.25x speedup on the standard benchmark suite. The fact that Python 3.11.2 is the fastest version made it quite reasonable to execute all of the methods in this version since we obtained optimization of the running time

[1]The documentation of Python and different versions can be found here: https://docs.python.org/3/.

concerning pre-processing and running of the models without any effort.

The machine learning framework used in this chapter is scikit-learn [75], with the version of 1.2.1, a free software machine learning library. This framework supports various classification, regression, and clustering algorithms and is designed to interoperate with the Python numerical and scientific libraries NumPy [41], SciPy [102], and Pandas [67]. In addition, scikit-learn also supports text vectorization methods such as BoW and TF-IDF.

4.1.2 The hardware

All experiments have been performed on Experimental Infrastructure for Exploration of Exascale Computing (eX3) [2] has more than 40 nodes with different hardware. The use of which nodes, or hardware, depends on the machine learning framework. We have implemented various algorithms from different frameworks in this thesis, which have required different hardware. In the case of scikit-learn and the classical approaches, we have deployed these algorithms on a single *AMD Epyc 7302P* that has 256 GB RAM.

Data split

The COCO dataset consists of one *dev* set, used to train and validate the model, and a holdout set. The holdout (test) set is not used for anything except to check the performance of the models when the models are finished training. To train a model, we divide the dev set into a training set, a validation set, and a test. We applied cross-validation methods to split the data.

Before applying the cross-validation method, we ensured a test dataset by splitting the dev set into a temporary training set and a test set using the *train_test_split* class from scikit-learn. We specifically applied the split with stratification to ensure that the temporary train and test set had the same number of categories. This test set was used to evaluate each model. The temporary training set was then driven through the 9-fold cross-validation method. The reason why we applied specific 9-folds was that we wanted to split data into seven equal folds. We used the *StratifiedKFold* to ensure each set in each fold has the same category distribution.

Evaluation metric

In both cases of the One-for-All and One-for-One, we apply the MCC[3] to evaluate the predictions on the category level. To obtain the performance of

[2] eX3 is an experimental cluster owned by Simula Research Laboratory. The official web page: https://www.ex3.simula.no/

[3] We used the *matthews_corrcoef* from scikit-learn to compute MCC: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html.

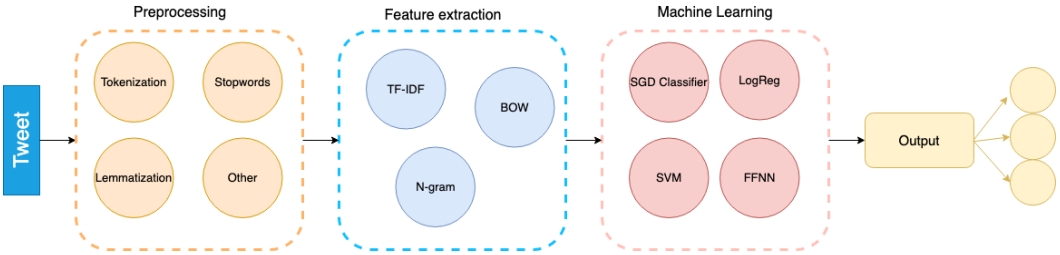


Figure 4.1: The steps of the classical approach; preprocessing, feature extraction, and machine learning model.

the whole approach from every category, we use the mean of every single MCC score.

$$\text{MCC}_{\text{Model}} = \frac{\text{MCC}_1 + \text{MCC}_2 + \cdots \text{MCC}_9}{9} \tag{4.1}$$

For the rest of the thesis, any mention of MCC will be in reference to $\text{MCC}_{\text{Model}}$, indicating a specific type of MCC and avoiding any potential confusion with other types.

4.2 Applied machine learning models

This section introduces the machine learning models executed to learn the text representation and their tunable parameters. As described in Section 2.2, machine learning (ML) models such as LogReg and SVM are popular choices. Note that every ML introduced here have implemented with the scikit-learn package. In addition, there following models were employed:

- Logistic regression (LogReg)
- Support vector machine (SVM)
- Feedforward neural networks (FFNN)

The SGD Classifier is a linear classifier with SGD-based learning implemented in scikit-learn. The backend models of this implementation consist of LogReg and SVM. The loss gradient is estimated for each sample at a time, the model is updated with a decreasing strength schedule, and SGD allows minibatch learning.

Table 4.1 displays the tunable and chosen parameter values. Note that the SGD classifier with *hinge* loss is equivalent to linear SVM, SGD with *log* loss is equivalent to LogReg, etc. The only difference between SVM, LogReg, and the SGD classifier is that the SGD classifier is implemented with the SGD algorithm.

These machine learning methods were executed in a *One-for-One* manner, executing nine different models for the nine categories. We consider every category to be completely independent of the other. For the prediction

Model	Parameter	Parameter values
LogReg	penalty	L1 , L2, Elastic net
LogReg	optimizer	lbfgs, liblinear, newton-cg, saga
LogReg	maximum iterations	100, 500, 1000
SVM	C	0.1, 1, 10
SVM	kernel	linear, poly, RBF, sigmoid
FFNN	number of layers	2,3,4,5
FFNN	hidden layer sizes	50,100,150
FFNN	activation function	identity, logistic, ReLu
FFNN	optimizer	SGD, Adam
SGD Classifier	loss	hinge, log, modified_huber squared_hinge, perceptron
SGD Classifier	penalty	L1 , L2, Elastic net
SGD Classifier	alpha	$1e-3, 1e-4, 5e-4, 1e-5, 5e-5$

Table 4.1: The chosen tuning parameters for the four machine learning models for learning text representation.

part, we combine the prediction of each model to achieve labels from every category.

4.3 Experiments with unigrams

This section introduces the baseline line of the text vectorization approaches,i.e., the BoW and the TF-IDF approaches. The experiments in this section have been performed with unigrams, the simplest form of N-grams, which are contiguous sequences of N items. An item can be a word, a character, or a part of a word. The unigrams are the simplest form of N-grams, but we still believe they can provide valuable information for conspiracy detection tasks. They do not capture any information about the order or sequence of words, which can limit their usefulness in certain contexts. Higher-order N-grams, such as bigrams, trigrams, or higher, can be more useful, as they can capture more complex patterns and relationships between words. We will study the higher N-grams in the next section.

Note that, in this section, the only preprocessing technique performed is the removal of the stopwords for both the BoW and TF-IDF approaches.

4.3.1 The BOW approaches and results

The BoW approach is implemented with *CountVectorizer* class from scikit-learn, which transforms a collection of text samples into a matrix of token counts. This implementation creates a sparse representation of the token counts. The representation created from this method is then fed to the machine learning models described in the section above. The benefit of

	Category	SGD Classifier	SVM	FFNN	LogReg
BOW	Suppressed cures	0.440	0.327	0.333	0.239
	Behaviour and Mind Control	0.454	0.377	0.421	0.349
	Antivax	0.455	0.482	0.398	0.379
	Fake Virus	0.351	0.400	0.312	0.308
	Intentional Pandemic	0.285	0.247	0.232	0.231
	Harmful Radiation/ Influence	0.689	0.622	0.581	0.541
	Population reduction	0.648	0.582	0.565	0.491
	New World Order	0.715	0.727	0.681	0.654
	Satanism	0.444	0.384	0.312	0.257
	Average	**0.498**	0.461	0.432	0.383
TF-IDF	Suppressed cures	0.759	0.394	0.362	0.00
	Behaviour and Mind Control	0.483	0.374	0.222	0.094
	Antivax	0.526	0.455	0.412	0.250
	Fake Virus	0.322	0.366	0.332	0.017
	Intentional Pandemic	0.271	0.131	0.211	0.082
	Harmful Radiation/ Influence	0.638	0.552	0.510	0.00
	Population reduction	0.713	0.561	0.540	0.183
	New World Order	0.70	0.711	0.690	0.306
	Satanism	0.465	0.445	0.378	0.00
	Average	**0.542**	0.443	0.380	0.104

Table 4.2: The MCC scores of the BoW and TF-IDF approach per category on the holdout dataset. These scores were generated after training the ML models from Section 4.2 with unigrams BoW and TF-IDF text representation. The scores have been rounded up to three decimals.

using CountVectorizer is its excellent properties, such as N-gram ranges. For example, the N-gram ranges vary in the range of n-values for different word N-grams, and the values of N must be between the minimum and maximum of N.

4.3.2 The TF-IDF approaches and results

Like the BoW approach, the TF-IDF approach was implemented with scikit-learn's *TfidfVectorizer* class. In addition, we implemented the TF-IDF approaches with the same N-gram ranges described in 4.3.1, and the stopwords were removed from the textual data.

4.4 Experiments with N-grams

Table 3.1 shows the ten most common bigrams and trigrams in the training data of the COCO dataset, and it is evident that N-grams such as "world order order", "population control," and "soros bill gates" are essential for classifying several categories. Therefore, to improve the results from previous sections, we experimented with different N ranges, such as unigrams, bigrams, and trigrams. In addition to that, we also experimented

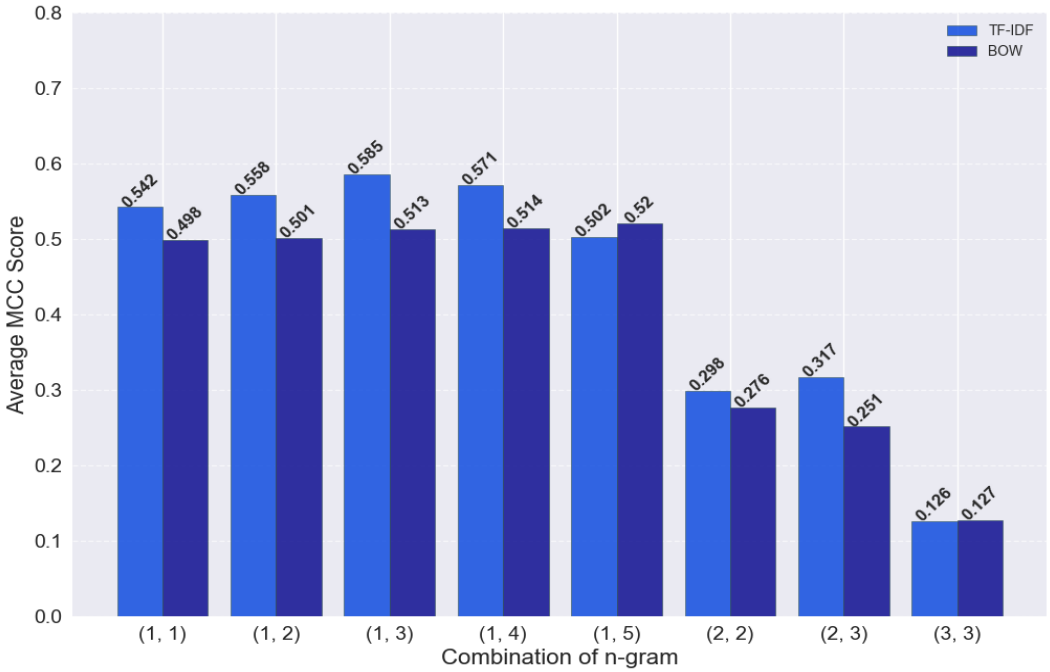


Figure 4.2: Comparison of the performance of the BOW and TF-IDF approaches with various N-grams.

with combining these N-grams, such as a combination of bigrams and trigrams. That was done by forwarding the *ngram_range* parameter to the CountVectorizer and similar to TfidfVectorizer. More specifically, we used the following N-gram ranges to work with:

- **(1,2)** Combination of unigrams and bigrams.
- **(1,3)** Combination of unigrams, bigrams and trigrams.
- **(2,3)** Combination of bigrams and trigrams.
- **(1,4)** Combination of unigrams, bigrams, trigrams, and four-grams.
- **(1,5)** Combination of unigrams, bigrams, trigrams, four-grams, and five-grams.
- **(2,2)** Bigrams.
- **(3,3)** Trigrams.

Furthermore, before executing these experiments, we removed the stop-words. No other preprocessing was done on the textual data.

4.5 Enhancing Text-based Model Performances: Strategies and Techniques

All of the experiments till now have been executed with no preprocessing of the textual data. In this section, we will introduce and explore

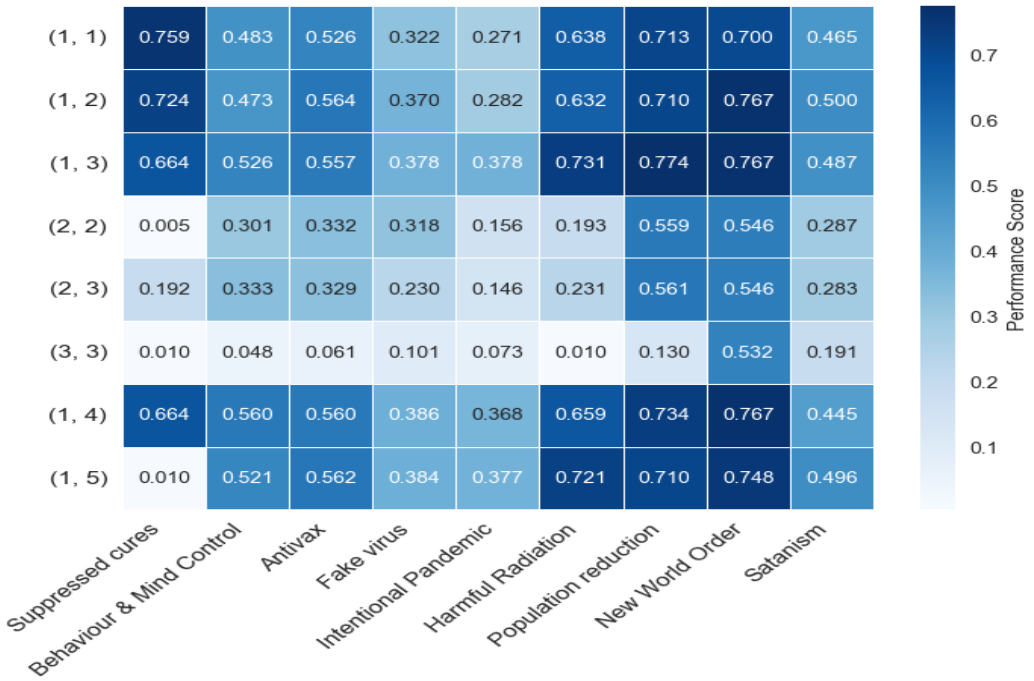


Figure 4.3: The TF-IDF performance comparison with the SGD classifier on different N ranges.

different preprocessing techniques to improve the performance of the models. We are working with the dataset from Twitter, which contains considerable noise. The most common type of noise is spelling errors, a common challenge when building NLP models based on tweets. Due to the character limit in tweets, people often use abbreviations, slang, and informal language, leading to misspellings and typos. In addition, the speed at which people write and send tweets can also contribute to spelling errors. Furthermore, people may use different dialects and accents, resulting in spelling and word usage variations. As a result, tweets can often contain misspellings and grammatical errors, making natural language processing tasks more challenging. This section will deploy different preprocessing techniques to tackle these challenges.

4.5.1 Spelling correction

There exist several open-source packages that tackle the issue of spelling correction, such as *textblob* [4] and *pyspellchecker*[5]. However, these packages are trained or based on general data, not specifically on Twitter data, leading to wrong word recommendations. In addition, the essential words in our case are the COVID-19-related conspiracy words such as *plandemic*, *qanon*, *bioweapon*, etc., which these packages failed to classify the correct words. To solve this issue, we built our spelling correction tool trained on

[4] https://textblob.readthedocs.io/en/dev/
[5] https://pypi.org/project/pyspellchecker/

a COVID-19 dataset.

The solution is based on Peter Norvig's work on spelling correction[6]. Norvig outlined a simple but effective approach to spelling correction using a probabilistic model based on word frequency and edit distance. Given a misspelled word, the model generates a list of candidate corrections by applying a series of edit operations to the misspelled word. For each candidate correction, the model calculates a score based on the frequency of the candidate word in a reference corpus and the edit distance between the candidate word and the original misspelled word. The model then selects the candidate word with the highest score as the most likely correction for the misspelled word.

We used the BERTweet dataset[7] as the corpus for the spelling correction model. This model was deployed for each training and test sample to check for every word and correct if it was misspelled.

4.5.2 Emojis to text

The tweets in the dataset also contain a lot of emojis. Therefore, we convert the emojis into their textual meaning to make the ML better understand the tweets. This is done by using the *emoji* [8].

4.5.3 Word segmentation

Another challenge regarding the noise in the dataset is the augmented words in hashtags or any other misspelling forms. Some examples are "PlandemicHoax" and "democRatsAreTheVirus". To separate these words, we deploy word segmentation techniques. Word segmentation is dividing a continuous sequence of text into individual words. It is an essential preprocessing step in many natural language processing (NLP) tasks. In addition, accurately segmenting text into individual words is essential for many downstream NLP tasks that rely on word-level features, such as word frequency, part-of-speech tags, or word embeddings. There are different techniques for word segmentation depending on the language and the type of text being analyzed.

The word segmentation process was performed with *ekphrasis*[9] package, which was first introduced in 2017 [8]. This package provides word statistics (unigrams and bigrams) from 2 big corpora, the English Wikipedia and a collection of 330 million English Twitter messages. The word segmentation implementation uses the Viterbi Algorithm [29], requiring word statistics to identify and separate the words in a string. We performed the word segmentation on every hashtag and replaced the segmented version.

[6] http://norvig.com/spell-correct.html
[7] https://github.com/VinAIResearch/BERTweet
[8] https://pypi.org/project/emoji/
[9] ekphrasis's Github page: https://github.com/cbaziotis/ekphrasis

4.5.4 Word normalization

Word normalization is a preprocessing technique in natural language processing that involves transforming words into their base or canonical form to reduce text complexity and improve the performance of NLP models. The process typically includes stemming, lemmatization, and case folding, with stemming removing suffixes to extract the root form of words, while lemmatization maps words to a standard dictionary form. By making text data more consistent, word normalization can enhance the performance and efficiency of text-based models. In our case, we performed the lemmatization of the text with the spaCy [46] package in Python.

4.5.5 Experimenting with the enhanced methods

We found that the TF-IDF approach with an N-range of $(1, 3)$ performed best among all other options, and this performance was achieved with the SGD classifier. We will integrate this approach with the suggestions in Section 4.5, aiming to improve the results from previous sections.

4.6 Results & Discussion

In this chapter, we conducted several experiments with non-deep learning-based approaches for COVID-19 conspiracy detection. The experiments were based on training classical machine learning models with word count-based methods such as BOW and TF-IDF. The first rounds of the experimentations were conducted with unigrams representation, but in later rounds, we also expanded the N-gram ranges to bigrams, trigrams, etc. Lastly, we carried out investigations with methods that would enhance the models' performance. In this section, we will discuss the results of all experiments.

4.6.1 Experiments with unigrams

The initial phase of experimentation focused on unigrams, with each machine learning model fine-tuned based on the parameters detailed in Table 4.1. The results for both vectorization methods are presented in Table 4.2. It was observed that the SGD Classifier yielded the most favorable results for both BOW and TF-IDF techniques, while the Support Vector Machine (SVM) emerged as a strong runner-up. In contrast, Logistic Regression (LogReg) exhibited the least desirable performance. Additionally, the TF-IDF method achieved the best overall MCC score of 0.542, outperforming the highest BOW score of 0.498.

Notably, the TF-IDF approach with the SGD classifier demonstrated excellent results for the **Suppressed Cures**, **Population Reduction**, and **New World Order** categories, with all of these categories achieving an MCC score higher than 0.700. This indicates that these models are primarily

keyword-based. However, the models faced difficulties in the **Fake Virus** and **Intentional Pandemic** categories, where the best MCC score obtained was approximately 0.322 or lower.

4.6.2 Comparison of N-gram ranges

Referring to Table 3.1, we hypothesize that expanding the N-gram range for text vectorizers may enhance the models' performance. With this consideration, we conducted experiments using various N-grams as detailed in Section 4.4. The results are presented in Figure 4.2. The TF-IDF method outperforms the BOW approach for every N-gram range, except for $(1,5)$, which can be disregarded due to the marginal difference. The highest overall MCC score across all categories was 0.585, achieved using the TF-IDF method with an N-gram range of $(1,3)$.

Based on the same figure, incorporating unigrams in the text vectorization process appears crucial; otherwise, the models' performance declines significantly. The combinations of $(2,2)$, $(2,3)$, and $(3,3)$ yielded an overall MCC score lower than 0.320. The performance of the TF-IDF method increases with the N-gram range until it reaches $(1,4)$. The combination of (1,3) includes unigrams, bigrams, and trigrams. The TF-IDF approach faces challenges with the $(1,4)$ range, as the dataset contains a limited number of four grams. The best BOW method registered an MCC score of 0.520, while the top-performing TF-IDF approach achieved an MCC score of 0.585.

Further examining the MCC scores, we analyzed the TF-IDF performance for each category. Figure 4.3 offers insights into the MCC scores presented in Figure 4.2 by displaying the individual category MCC scores. Our hypothesis was confirmed as the **Population Reduction**, and **New World Order** categories were comparatively easier to predict and are depicted in darker blue, except for the N-grams that exclude unigrams. Moreover, the **Harmful Radiation** category demonstrated relative ease in prediction, achieving an MCC score of 0.731 for the $(1,3)$ range. Conversely, the **Suppressed Cures** category encountered challenges with higher N-gram ranges, leading to diminished performance. Finally, as anticipated, the **Fake Virus** and **Intentional Pandemic** categories fared worse among all categories, but their performance improved slightly with the increase in the N-gram range.

4.6.3 Improving the model performances

To enhance the performance of the TF-IDF approach, we employed various techniques, including emoji-to-text conversion, word normalization, spelling correction, and word segmentation. However, implementing these methods did not significantly improve the TF-IDF performance. Table 4.3 reveals a slight decline in the performance of the original best model, TF-IDF, with a (1,3) N-gram range. The spelling correction and word segmentation techniques were the least impactful, suggesting that our dataset con-

Prepossessing method	MCC scores
Original (1,3)	0.585
Emojis to text	0.541
Word segmentation	0.527
Spelling correction	0.529
Word normalization	0.545

Table 4.3: The comparison of TF-IDF with (1,3) with different preprocessing methods.

tains distinct COVID-19 and Twitter-specific terminologies that traditional probabilistic correction models struggle to handle accurately.

4.7 Conclusion

This study explored various non-deep learning-based approaches for COVID-19 conspiracy detection, employing classical machine learning models alongside word count-based methods such as BOW and TF-IDF. Initial experiments using unigram representations demonstrated that the SGD Classifier performed best for both BOW and TF-IDF techniques, with the TF-IDF method achieving the highest overall MCC score. Subsequent experiments with varying N-gram ranges confirmed our hypothesis that expanding the N-gram range could improve model performance, with the TF-IDF method outperforming the BOW approach in most cases. The analysis of individual category performance revealed that specific categories, such as **Population Reduction** and **New World Order**, were comparatively easier to predict. In contrast, others, such as **Fake Virus** and **Intentional Pandemic**, struggled. However, an increase in the N-gram range led to slight improvements in these more challenging categories. In conclusion, our investigation of non-deep learning-based approaches for COVID-19 conspiracy detection highlights the importance of selecting appropriate N-gram ranges and vectorization techniques. While the TF-IDF method with the SGD Classifier performed best overall, understanding each category's unique challenges and characteristics is essential for optimizing performance across diverse conspiracy theories.

Chapter 5

Experiments with Transformer-based approaches

In this chapter, we are implementing Transformer-based approaches. The goal of this chapter is to answer the following questions:

1) How do the Transformer-based models do compared to the classical approaches?

2) Do the classical approaches outperform the Transformer models in some categories?

3) Among the pre-trained Transformer models, CT-BERT is the one that has been trained on domain-specific data and is known to perform best. Can the other models outperform CT-BERT?

We start with a pretrained simple BERT model, *bert-base-uncased*, and fine-tune this with a suitable neural network for conspiracy detection. Later on, we will experiment with more extensive and more advanced pretrained models, such as *roberta-large*, CT-BERT, BERTweet, and *Twitter-RoBERTa-Large*. We first present the technicalities around the experiments and then the experiments based on all methods. Finally, the results are compared and commented on at the end of the chapter.

5.1 Tools for Reproducible Research: Hardware and Software Considerations

This section contains the elements and descriptions of all the tools used to perform the experiments, such as the software, hardware, and various methods deployed.

5.1.1 The software

This section describes the software specifications and frameworks used to produce this thesis's results. Python 3.7.31 has been used to conduct all experiments in this chapter. One can easily organize, preprocess and visualize data, making Python a natural choice for many researchers. Combined

with open-source machine learning libraries such as PyTorch [73] and TensorFlow [66], Python has become a powerful tool for deep learning tasks.

This chapter uses PyTorch and Transformer [109] to deploy and train the selected models. PyTorch is an open-source machine learning library for Python primarily used for deep learning applications. It is developed and maintained by Facebook's AI Research lab and is designed to be flexible, fast, and easy to use. One of the critical advantages of PyTorch is its ability to work seamlessly with Graphical Processing Units (GPUs). PyTorch allows developers to easily leverage the power of GPUs for deep learning by providing GPU-accelerated tensor computations, which allow for faster and more efficient model training and inference. PyTorch also supports distributed computing, allowing developers to parallelize model training across multiple GPUs or machines.

The pretrained models exist on Hugging Face's website and can easily be accessed with the Transformer library. The Transformer library provides a high-level API for building, training, and using various deep learning models for tasks such as text classification, named entity recognition, question answering, and language generation. It is built on top of PyTorch and TensorFlow and includes pretrained models for everyday NLP tasks. In addition, the library is designed to make experimenting with different models and techniques accessible and includes tools for fine-tuning pretrained models on new data.

5.1.2 The hardware

The experimentation in this chapter has been carried out with an Nvidia Volta A100 on the eX3 computing cluster. PyTorch allows accelerating performance with GPUs. GPUs are specialized hardware designed for parallel processing. As a result, they are much faster than traditional CPUs for running certain types of computations, such as matrix multiplications.

5.1.3 Reproducibility

All experiments were executed with a fixed random seed to ensure the reproducibility of the results in this chapter and the rest of this thesis. The exact sequence of random numbers will be generated each time the code is executed, which leads to the same sequence of model parameters being initialized and the same sequence of training, validation, and test data being sampled. To seed everything, the following code was used:

```
def seed_everything(seed_value=2022):
    os.environ['PYTHONHASHSEED'] = str(seed_value)
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    torch.cuda.manual_seed_all(seed_value)
```

```
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

This code sets the seed value for random number generators in various libraries commonly used in deep learning, including Python's built-in random library, NumPy, and PyTorch. The results should be deterministic and reproducible by setting the same seed value each time the program is run. Additionally, it ensures that the CUDA backends in PyTorch are deterministic by setting the deterministic flag to `True` and the benchmark flag to `False`. All experiments were executed with a seed value of 2022, including the scikit-learn modules used for data splitting and model training.

5.2 The data split

The data split was performed using a 9-cross validation technique to split the data into training, validation, and test sets. However, the cross-validation technique was performed with a stratification method based on all categories. Stratification refers to dividing a dataset into homogeneous subgroups based on a specified criterion, which is all of the conspiracy categories in our case. Stratification ensures that each subgroup represents the overall population's attributes, making the sample more representative and reducing bias. For example, when splitting a dataset into training and testing sets, stratification ensures that each subgroup's relative proportion is maintained in both the training and testing sets. This is important because it can help reduce the risk of sampling bias, which can arise when the training and testing sets do not represent the overall population.

As mentioned before, the COCO dataset contains two sets; a *dev* and *holdout* set. The *dev* set is used to train and evaluate the modes, while the *holdout* set is only used in the final stages of the pipeline. More specifically, the *dev* dataset was splitted by the following points:

- Firstly, the *dev* set was splitted into a temporary training and test set using the *iterative_train_test_split* class from scikit-multilearn [1] with stratification. The split ratio was 90 and 10, respectively, for temporary training and testing sets.
- Secondly, the temporary training set was splitted into nine different cross-validation folds. We chose the number 9 to ensure that each split was almost the same size as the test set. Note that the test was not utilized for anything else and was set aside for evaluating each model's performance.

This resulted in 9-folds of training and validation sets and one single test set. The amount of samples in each category is shown in Figure 5.1.

[1] Scikit-multilearn is a library for multi-label classification that is built on top scikit-learn: http://scikit.ml/

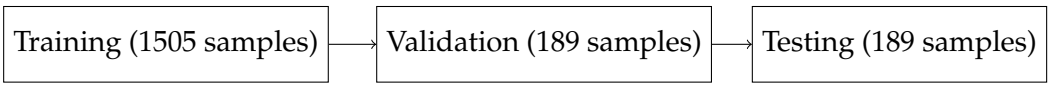


Figure 5.1: The number of samples in training, validation, and test set after 8-fold cross-validation.

5.2.1 Challenges in dataset acquisition

The dataset acquisition presented two significant challenges during experimentation with the COCO dataset. The first challenge pertained to the training dataset containing duplicate samples. The original training dataset had 1912 samples, of which 33 were duplicates. After removing the duplicates, we were left with 1879 samples in the training dataset.

The second challenge concerned the holdout dataset, which contained 830 samples. The presence of special punctuation marks in the dataset file posed a challenge while reading the data. The default `pandas.read_csv` module could only read 823 samples, leading to the loss of some lines during the file-reading process. To rectify this issue, we used the `error_bad_lines` argument with `quoting` in the same module, as shown below:

```
test_df = pandas.read_csv(
    path_to_data,
    quoting=csv.QUOTE_NONE,
    error_bad_lines=False
)
```

Regrettably, these errors were only discovered after several rounds of experimentation, which necessitated restarting our work. These errors directly impacted the final MCC scores of the models, leading to relatively lower MCC scores. This can be attributed to the model being trained on multiple duplicate samples, which introduced redundancy and compromised the diversity of the training data.

5.3 The chosen Transformer-based models

This section will introduce the Transformer-based models employed for conspiracy detection, and we will motivate why these models are good choices for the task of conspiracy detection. However, as shown in Figure 3.1, the dataset is highly imbalanced, and there are exceptionally few samples of 3's and 2's, meaning that the models have little data to learn these categories from. In addition, many domain-specific terminologies are used in these tweets, which could make the non-domain-specific models struggle to perform well. Therefore, we need a big pre-trained model that tackles the challenge of understanding the COVID-19-specific terminologies usually used on Twitter, the language of fake news spreaders, e.g., sarcasm, irony, hate speech, etc., in addition, to the standard

language used in tweets. Based on these criteria, we have chosen three types of pre-trained models for COVID-19-related conspiracy detection: the general models trained on various corpus, the Twitter-based models that have pre-trained on large amounts of tweets, and the models that have pre-trained on COVID-19-related tweets. Based on these criteria, we introduce the chosen pre-trained models for this task:

- **BERT-Uncased-Base**: This BERT model is relatively small, the most common pre-trained model among all the text classification problems, and serves as the baseline model for our case. Note that this model is uncased, meaning it does not distinguish between the words like "Car" and "car" and treats them evenly, which is a good quality when working with Twitter data since the data contains a significant quantity of capitalization errors, where words are not capitalized correctly.
- **RoBERTa-Large**: The RoBERTa model used in this study is a sizeable pretrained model trained on a diverse 160 GB text corpus. While it was not explicitly trained on COVID-19-related tweets, its extensive training corpus is expected to yield good performance. However, two concerns arise: the model's training data includes unfiltered internet content with potential biases, and it is cased, distinguishing between uppercase and lowercase letters, which can pose challenges when working with Twitter data.
- **BERTweet**: The BERTweet has two versions of the models; the first is a general BERT model trained on 850 million tweets, and the second is trained on 23 million tweets. Both versions are important as they could perform better with the COCO dataset because of their training dataset.
- **Twitter-RoBERTa-Large**: This model, based on RoBERTa-Large, was trained on a corpus of 154 million tweets.
- **CT-BERT**: This model was trained on 22.5 million tweets, which contain the words "Wuhan", "ncov", "coronavirus", "covid", or "sars-cov-2". Therefore, this model has shown good results in various COVID-19 text classification tasks and is a solid choice for our case.

5.4 The methods

This section will present the methodologies used to build a Transformer-based model for COVID-19 conspiracy detection. As discussed in Background Section, the Transformer is a state-of-the-art model for most NLP tasks. We will experiment with general pretrained models such as BERT and RoBERTa and further investigate the domain-specific pre-trained models such CT-BERT, BERTweet, and Twitter-Roberta.

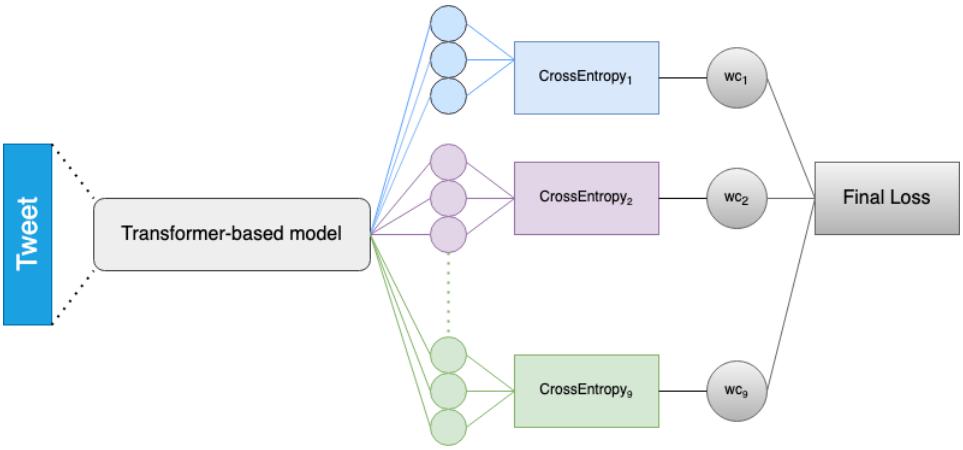


Figure 5.2: The One-for-All approach; training one Transformers-based model to predict all nine conspiracy categories simultaneously. Each color represents a conspiracy theory.

5.4.1 BERT-based Methods for Multi-Label Classification

Before setting up the experimentation on the dataset, the question of how to deal with labels arises. The labels in the dataset consist of a matrix rather than a single vector, making it a bit challenging to decide how to deal with it properly. As introduced in Section **??**, there are two common ways of dealing with this. The so-called One-for-One approach is based on building nine distinct classifiers for each of the nine vectors, i.e., conspiracy categories. On the other hand, the One-for-All approach suggests building one single model that is trained on all of the nine categories at once and then is used to predict all of the categories.

5.4.2 One-for-All: One Model to Rule Them All

As shown in Figure 5.2, the One-for-All is an efficient BERT-based multi-label multi-class classification model. This approach is based on training one Transformer-based model for classifying all conspiracy categories simultaneously. The model is fine-tuned for the nine categories with nine Cross Entropy loss functions, and the weighted losses for each of the nine categories are added to determine the overall loss, which is given by:

$$L = \frac{1}{N} \sum_{i}^{N=9} w_i L_i \tag{5.1}$$

As shown in Equation 5.1, the final loss is the weighted sum of the nine losses. The weights are computed by considering the number of samples in a specific category and dividing it by the numbers of each subcategory in that category. The advantage of such an approach is that a single model is fine-tuned to perform all the tasks simultaneously. Furthermore, the weights of all our loss functions are proportional to the inverse frequency of each class or sub-class they are related to.

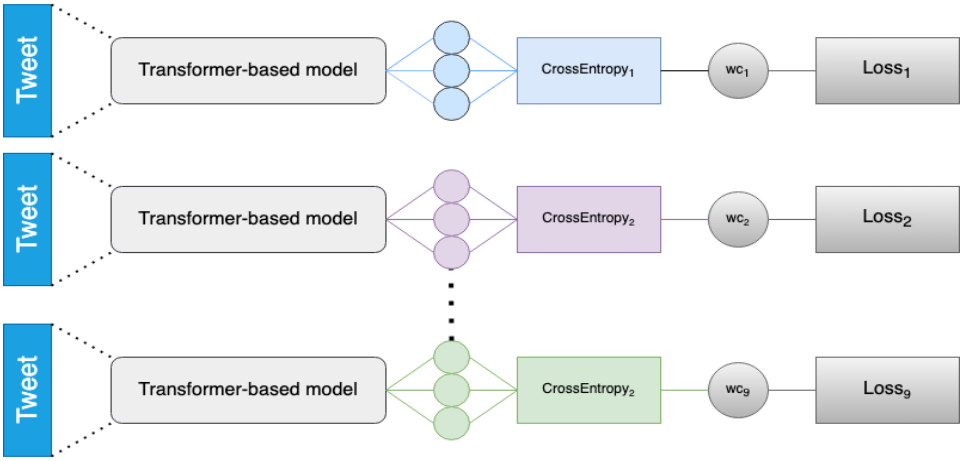


Figure 5.3: The One-for-One approach; Nine Models for Nine Mysteries. In contrast to the One-for-All approach, this method is based on training one Transformers-based model for each of the nine conspiracy theories.

One-for-All training, or fine-tuning, has been performed as a usual fine-tuning process but with minor modifications. First, each pre-trained Transformer model has been initialized with a superficial linear layer on the top of the model. Then, depending on the size of the model, it has chosen to give the linear layer a dimension of $[512, \text{Number-of-Classes}]$ for the base models and $[1024, \text{Number-of-Classes}]$ for larger models. Furthermore, we developed one separate class for each pretrained trained model, where the number of classes was 27, the total distinct number of categories. As briefly discussed above, for each of the nine categories, there is a single cross-entropy loss function that computes the loss of each epoch. During each epoch, the prediction and gold labels are extracted from the overall prediction of BERT using a single Transformer model for each of the nine categories. This extraction is performed where the corresponding logits for each category are obtained from a larger tensor based on the interval $[3i, 3i + 3]$. Furthermore, the label values for each category are retrieved from a separate tensor.

Finally, the model was deployed with the following steps about the 8-fold cross-validations set; note these steps are based on Figure 2.3:

```
For each fold in 9-folds:
    1) Train a One-for-All approach.
    2) At each epoch, evaluate the model on the
        corresponding validation set.
    3) Save the model with the highest MCC score
        on the validation score.
    4) Finally, evaluate the best model on the test dataset.
```

Among the 9-folds, the best model with the highest validation score was chosen.

5.4.3 One-for-One: Nine Models for Nine Mysteries

As shown in Figure 5.3, the One-for-One approach is based on considering each of the nine categories separately from each other and fine-tuning nine separate Transformer-based models. The weighted loss is dropped in this approach, and we are applying nine simple Transformers based on three class text classifications. Besides these differences, we are applying the methodologies introduced in Section 5.4.2 regarding the loss function and optimizers. However, the data split is performed differently. Since we are building nine separate models, the data has been divided differently concerning the stratification strategy. In this case, we have divided the data into 9-folds, where each fold has been stratified concerning each category. We ended up with 72 folds (9 categories times eight folds).

The training of the One-for-One approach was performed in the same manner as the One-for-All, but with small changes, and included the following steps:

```
For each category:
    For each fold in 9-folds for that category:
        1) Train a One-for-All approach.
        2) At each epoch, evaluate the model on the
            corresponding validation set.
        3) Save the model with the highest MCC score
            on the validation set.
        4) Finally, evaluate the best model on the test dataset.
```

Note this approach is more expensive in terms of computational power and time to fine-tune the models.

5.5 Experiments: setup and execution

The One-for-All and One-for-One approaches were employed with the same parameters except for the number of epochs. Since the One-for-All is a much larger model with a 27 number of parameters, we chose to work with 40 epochs. In contrast, we used only 20 epochs in the One-for-One approach since it has only three number of parameters. Furthermore, Table 5.1 displays the selected parameters for the experiments. In addition, all of the optimizers mentioned in Section 5.4.2 were implemented with the same decay rate, but the SGD and the RMSprop were implemented with an additional parameter `momentum`, which is used to accelerate the optimization process by accumulating the past gradients and adding them to the current gradient. The `momentum` was kept at 0.9 for all experiments.

All experiments were conducted with the `torch.optim.lr_scheduler.ReduceLROnPlateau`[2] scheduler from PyTorch library. Generally speaking, a

[2]The scheduler can be found here. https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

Parameters	Values
Batch Size	16
Number of epochs	40 (One-for-All) | 20 (One-for-One)
Max length of tokens	128
Learning rate	$2e^{-5}$
Weight Decay	0.001

Table 5.1: The table shows the chosen parameters to fine-tune the Transformer-based models.

scheduler adjusts the learning rate at specific intervals during training to influence the convergence rate toward the optimal solution and improve the model's overall performance. By gradually reducing the learning rate as the model approaches convergence, precise parameter adjustments are made to prevent the model from becoming stuck in sub-optimal or exceeding the optimal solution. The `ReduceLROnPlateau` monitors the model's performance metric and reduces the learning rate when the performance metric stops improving. The learning rate reduction occurs at predefined intervals, and the reduction size is typically a multiplicative `factor`. In addition, the `ReduceLROnPlateau` scheduler has the `patience` parameter, which specifies the number of epochs to wait before reducing the learning rate if there is no improvement in the validation loss. In other words, if the validation loss does not improve after patience epochs, the learning rate is reduced by a factor of a `weight_decay`. In our experiments, we chose the `patience` to be four and the `weight_decay` to 0.003 and `factor` to be at 0.3.

5.5.1 Experimenting with all pre-trained models

The first experiment consisted of deploying all of the selected Transformer-based models with the One-for-All approach. In this round, we used the `AdamW` optimizers from PyTorch with the learning rate from Table 5.1. This experimentation round aimed to determine which pre-trained model performed best for the COCO dataset and achieved the best MCC score.

5.5.2 Experiments with different optimizers and learning rates

The CT-BERT performed best in the previous section and were selected for further experiments with different optimizers and learning rates. Among the chosen optimizers, we have the following:

- SGD
- Adagrad
- Adadelta
- RMSProp

These optimizers, which were introduced in Section 2.3.5, have been implemented with PyTorch [3]. The optimal learning rate depends on various factors, such as the problem's complexity, the dataset's size, the model's architecture, and the optimization algorithm used. In this case, we applied the learning rates of 0.1, 0.01, and 0.001 to find the optimal performance.

The different learning rates affect optimizers differently. Based on some experimentation, we found that for some of the chosen learning rates, the model only learns a little. Therefore, we implemented the model with early stopping to reduce the running time of the models. Early stopping is a regularization technique used in deep learning to prevent the overfitting of a model. It involves stopping the training process of the neural network before it reaches the point of overfitting. The basic idea behind early stopping is to monitor the model's performance on a validation set during training. As the training progresses, the model's performance on the validation set typically improves up to a certain point and then starts to degrade due to overfitting. By monitoring the validation performance, we can detect when the model starts to overfit and stop the training process at that point. We implemented the early stopping with a technique called patience. Patience is a hyperparameter that controls the number of epochs the model can continue training without improving its validation performance. Training is stopped if the model fails to improve for several epochs. We chose the patience to be at 5.

5.5.3 Experiments with One-for-One

As shown in Figure 5.4, CT-BERT dominates all other pre-trained models predicting all nine categories. To further experiment with it, we apply the One-for-One method. Instead of training one model to predict all nine categories, we will develop nine models, one for each category. To do so, we neglect the weights, as earlier used in One-for-All, and the training is performed on shorter epochs. The best model was selected based on the overall mean MCC during One-for-All training. The problem with such an approach is that the performance of every category is not considered but rather the performance of every category. We, therefore, believe that by using utilizing the CT-BERT, we can achieve improvement in performance. Furthermore, we did not experiment with other models because the One-for-One is more computationally expensive.

5.6 Results and Discussion

In the present chapter, we introduced Transformer-based models utilized for conspiracy detection. Two distinct strategies were employed to implement these models: the One-for-All and One-for-One approaches. The former entails constructing a single model capable of predicting all nine con-

[3]The overview of all optimizers can be found here: https://pytorch.org/docs/stable/optim.html

Model	Evaluation MCC	Holdout MCC
BERT-Base-Uncased	0.654	0.660
BERTweet-Covid19-Base-Uncased	0.682	0.653
RoBERTa-Large	0.658	0.680
Twitter-RoBERTa-Large	0.723	0.713
BERTweet-Large	0.733	0.727
CT-BERT	**0.753**	**0.765**

Table 5.2: The evaluation and holdout MCC score achieved by all six models with One-for-All.

spiracy categories, while the latter involves developing a separate model for each category. These methodologies were executed using 9-fold cross-validation sets, with a single model trained for each of the nine folds. Additionally, a separate test set, excluded from the nine folds, was utilized to assess each of the nine models. The model exhibiting the highest validation Matthews Correlation Coefficient (MCC) score was subsequently deemed the best among the nine folds and tested on the holdout set.

5.6.1 Comparison of the One-for-All approaches

Six pre-trained BERT-based models were implemented, as delineated in Section 5.3. A comparison of these models, employing the One-for-All approach, is presented in Table 5.4. Notably, these models enhanced the baseline models from the preceding chapter, achieving the highest MCC score of **0.765** on the holdout set using CT-BERT. Moreover, domain-specific models outperformed general models, with Roberta-Large and BERTweet-Large registering MCC scores of **0.680** and **0.727**, respectively. However, the BERTweet-Covid19-Base model did not surpass the performance of the BERT-Base model, an unexpected outcome considering BERTweet-Covid19-Base is derived from the RoBERTa base model and additionally trained on 23 million tweets [71].

Twitter-RoBERTa-Large and BERTweet-Large are both predicated on the RoBERTa-Large model. The former is trained on 154 million tweets acquired by filtering 220 million tweets through the Twitter Academic API. The creators of Twitter-RoBERTa-Large assert that the training tweets were collected between January 2018 and December 2022. BERTweet-Large was trained on a substantial corpus of tweets (850 million), with five million tweets about the COVID-19 pandemic. Despite BERTweet-Large being trained on 5.5 times more data than Twitter-RoBERTa-Large, it only achieved a marginally superior MCC score of 0.014. This can be attributed to approximately 99.4% (845 of 850 million tweets) of the training data originating between January 2012 and August 2019 and is unrelated to the COVID-19 pandemic. As anticipated, the highest MCC score was obtained using CT-BERT, which has previously demonstrated exceptional performance on other COVID-19 NLP tasks [1, 59, 97]. Although the total num-

ber of tweets employed for CT-BERT training (22.5 million) is considerably lower than that of Twitter-RoBERTa-Large and BERTweet-Large, the training data was amassed during the peak of the COVID-19 pandemic (January 2020 to July 2020) and exclusively comprised pandemic-related tweets.

Furthermore, domain-specific models were expected to surpass general models, as they are already acquainted with COVID-19 terminologies and phrases, thus facilitating accurate categorization. In addition, they are specifically tailored to the nuances and context of the domain of COVID-19 conspiracy categories. By being trained on a large corpus of text related to a specific subject area, domain-specific models gain a deeper understanding of the language patterns, common phrases, and unique expressions prevalent within that domain. In contrast, general pre-trained models are designed to capture a broader understanding of language patterns across multiple domains. While they can provide a solid foundation for various tasks, they might not possess the specialized knowledge required to excel in a specific domain. More specifically, the domain-specific models are trained on a corpus of text that contains a higher frequency of domain-specific terms, enabling them to understand better and process that domain's unique vocabulary. In addition, these models better understand the context in which specific terms and phrases are used. They are often fine-tuned on a smaller dataset from the target domain, enabling them to adapt to the target data's specific language patterns and nuances, resulting in improved performance.

5.6.2 Comparison of the categories

To further examine the MCC score of the One-for-All approaches, we refer to Figure 5.4, which displays the MCC score obtained for each of the nine categories. Note that the mean of each row from this figure corresponds to the MCC scores in Table 5.4. We can observe that the categories such as **Population Reduction** and **New World Order** are relatively easier to classify since these columns are relatively darker than the other columns, and the smaller models, such as the BERT-Base and BERTweet-Covid19-Base, are doing quite well. Conversely, CT-BERT performs extremely well on columns such as the **Suppressed cures** and **Satanism**, thanks to its large domain-specific ability and outperforms other models. The CT-BERT obtains an MCC score on **0.945** and **0.716**, respectively, for these categories. Other Categories include the **Behaviour Mind control**, **Antivax**, and **Harmful radiation** are comparatively also easier to predict as the performance gets better with bigger models.

The Heatmap in Figure 5.4 has the lightest color for the **Fake virus** and **Intentional pandemic** categories, meaning that the models struggle most with these two categories and brings down the overall MCC score. The best MCC score for these categories was obtained at **0.615** and **0.585**, respectively, which are relatively lower than the mean score in Table 5.4. There could be several reasons why we obtain poor results in these

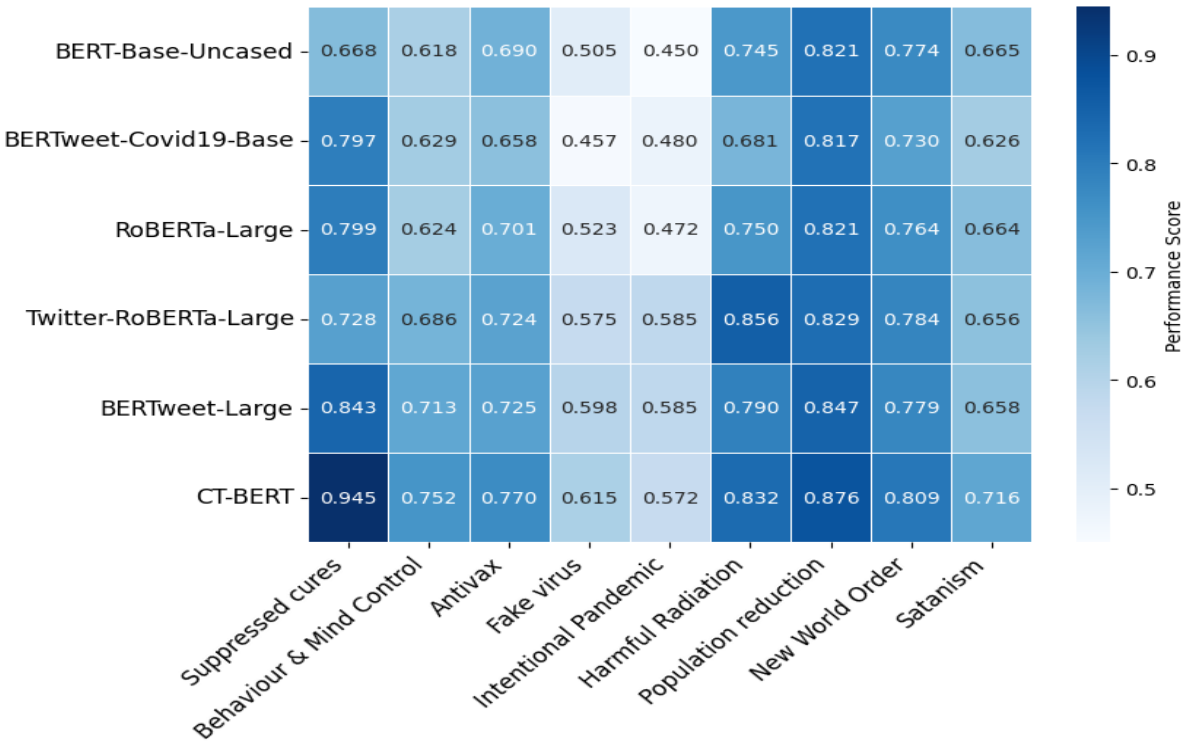


Figure 5.4: Heatmap displaying the performance of the six models evaluating all nine categories. The scores are given in MCC scores.

categories. These categories are typically more comprehensive than the others; one can express their support or discuss them in various ways. In addition, we also have a limited amount of manually labeled data samples for these categories. As shown in Figure 3.1, the 85.3% and 84.0%, respectively, percentages of the data contains the *Non-conspiracy* subcategories. A final reason could be that the data is noisy or has been manually labeled in a way that consists of inconsistent labels.

5.6.3 Error analysis

To investigate some of the claims in the previous section, we refer to the confusion matrix in Figure 5.5, which is the average of the normalized confusion matrix of the CT-BERT with One-for-All approach. First, the method can correctly classify the majority (97.3%) of the *Non-Conspiracy* subcategory. Still, however, the method struggles with the *Discusses* and *Promotes/Supports* subcategories. There is an overlap between the *Discusses* subcategory and the other two subcategories, where 17.9% of it has been predicted as the *Non-Conspiracy* and 6.3% has been predicted as *Promote/Support*. This could be because there is an overlap between the labels in these subcategories. On the other hand, 20.4% of the *Promotes/Supports* samples has been predicted as the *Non-Conspiracy*, while 6.80% of it has been predicted as **Discusses**.

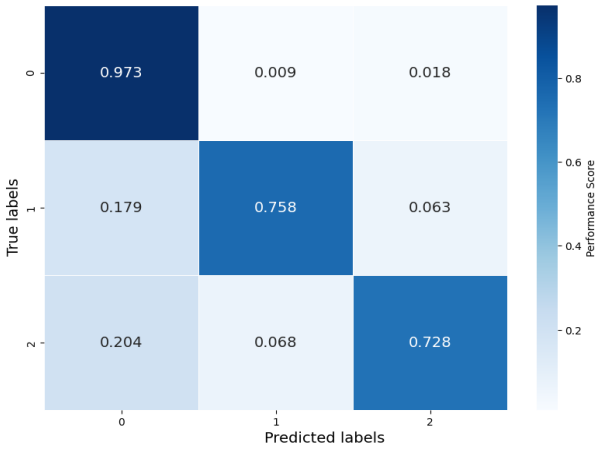


Figure 5.5: The average of normalized confusion matrix of One-for-All from Figure 5.6.

Optimizer	Evaluation MCC	Holdout MCC
SGD	0.748	0.716
Adagrad	0.598	0.301
Adadelta	0.705	0.677
RMSProp	0.492	0.294
AdamW	**0.753**	**0.765**

Table 5.3: Effect of optimizers on CT-BERT with One-for-All, and compared to AdamW.

The heatmap in Figure 5.6 displays the normalized confusion matrix for each category. In general, the model struggles with predicting on *Discusses* and *Promotes/Supports* subcategories, where six of the nine have a comparatively low score on the *Promotes/Supports*. Furthermore, the model performs best on the **Suppressed cures**, obtaining almost 100% for each subcategory. Furthermore, categories such as **Population reduction** and **New world order** obtain quite well scores. On the other hand, the **Fake virus** and **Intentional Pandemic** scores a high level of confusion, where almost a third of *Discusses* and *Promotes/Supports* have been classified as *Non-Conspiracy*.

5.6.4 CT-BERT with different parameters

To continue our investigation of CT-BERT, we carried out additional experiments utilizing various optimizers and learning rates. The result of these experiments is shown in Table 5.3, and it was found that the AdamW optimizer worked best for our case. AdamW incorporates weight decay, a regularization technique, which improves generalization performance in deep learning models and is well-suited for problems with sparse gradients where only a small subset of parameters is updated in each iteration.

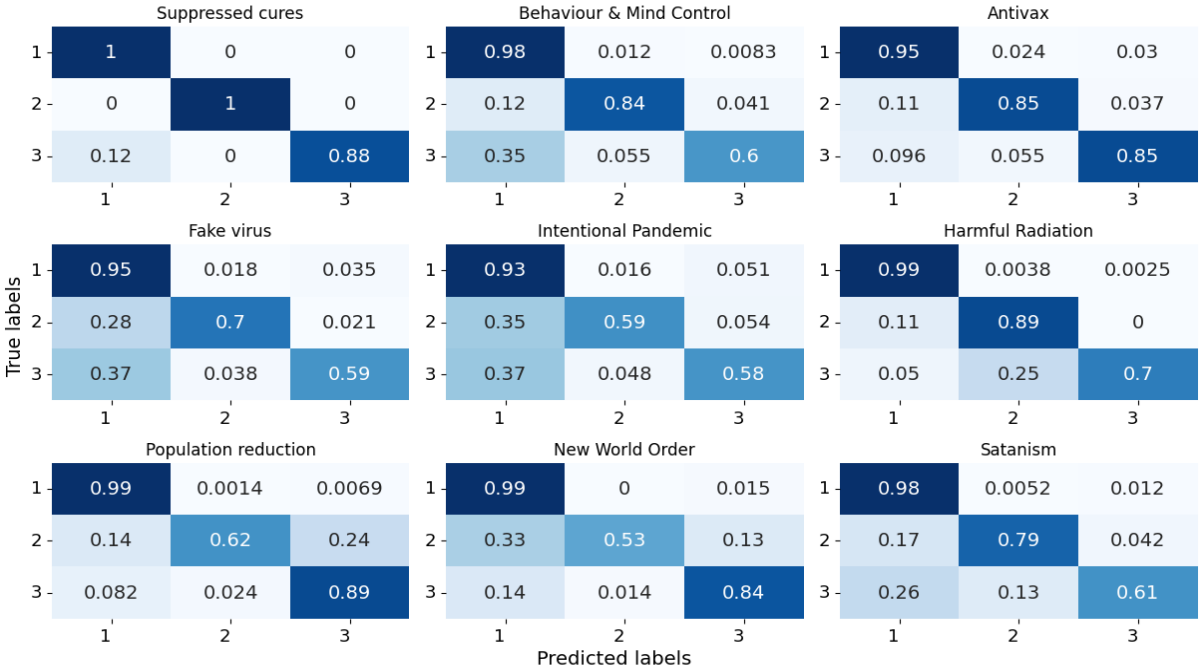


Figure 5.6: The normalized confusion matrix of One-for-All with CT-BERT for all categories. Note that 1, 2 and 3 relates to three subcategories, **Non-Conspiracy**, **Discusses Conspiracy** and **Promotes/Supports Conspiracy**.

5.6.5 One-for-One vs One-for-All

The comparison of the One-for-One and One-for-All is shown in Table 5.4, where the One-for-All completely surpasses the One-for-One. However, the One-for-One performs better for some single categories and cannot compete with One-for-All in the categories such as **Suppressed cures** and **Satanism**, where the differences are pretty significant. This result shows that training One-for-All with a weighted loss function and a multi-mask manner works better. The One-for-All approach obtains good results for some categories because there is a statistical relationship between the different categories, which is not discovered during the One-for-One approach.

5.7 Conclusion

This chapter presented the Transformer-based approach for tackling COVID-19 conspiracy detection. We first introduced the tools used to conduct the experiments, such as the hardware, software, and how the seeding was executed. The experimentation was executed in 9-fold cross-validation sets with stratification. Furthermore, we introduced two different methods; One-for-All and One-for-One. The One-for-All, implemented in a multi-task manner with a weighted loss function, was deployed with general and domain-specific pre-trained BERT- and RoBERTa-based models. Each

Category	One-for-All	One-for-One	TF-IDF
Suppressed cures	**0.945**	0.888	0.664
Behaviour and Mind Control	**0.752**	0.732	0.526
Antivax	**0.770**	0.728	0.557
Fake Virus	**0.615**	0.595	0.378
Intentional Pandemic	0.572	**0.575**	0.378
Harmful Radiation Influence	**0.832**	0.814	0.731
Population reduction Control	0.876	**0.881**	0.774
New World Order	0.809	**0.812**	0.767
Satanism	**0.716**	0.597	0.487
Average	**0.765**	0.736	0.585

Table 5.4: Comparison of the MCC scores achieved by One-for-All, One-for-One approach and TF-IDF.

model was deployed for every nine folds, which resulted in nine in total 54 models (six models times nine folds), and the best model was selected based on the validation set. As expected, the One-for-All approach with the CT-BERT was founded to be the best model, thanks to its large domain-specific training data, which is acquainted with COVID-19 terminologies and phrases, thus facilitating accurate categorization.

The best One-for-All approach, which achieved an MCC score of **0.765**, was obtained with CT-BERT and was further studied. We found that the categories of the **Fake virus** and **Intentional pandemic** were challenging to predict, so we achieved a low overall MCC score for each model. Furthermore, by examining the confusion matrix of every category, we concluded that one-third of these two categories for the second and third subcategories were confused with the subcategories. To further inquire about the CT-BERT approach, we examined the different optimizers and learning rates, which resulted in no improvement. Therefore, we concluded that the AdamW worked best for our system. Finally, we compared the One-for-All and One-for-One approaches. The One-for-One approach, built on developing nine separate models for each of the nine conspiracy categories, was implemented with no weighted loss function and conducted with fewer epochs. Based on our experimentation, we found that the One-for-All approach with CT-BERT and weighted loss function worked best for the task of conspiracy detection.

The results presented in Table 5.4 compare the best TF-IDF approach from Chapter 4 with the One-for-All and One-for-One approaches. While the TF-IDF scores exhibit lower values than the One-for-All approach, they still demonstrate the effectiveness of the model in capturing specific patterns and correlations within the dataset. However, the TF-IDF approach falls short of achieving comparable performance to the One-for-All approach. It is worth noting that the performance gap between the TF-IDF approach and the transformer-based approaches narrows for specific categories,

namely **Harmful Radiation Influence**, **Population reduction Control**, and **New World Order**. Nevertheless, the One-for-All approach outperforms both the One-for-One and TF-IDF methods regarding average MCC scores, making it the most successful approach.

Chapter 6

Experiments with advanced learning techniques

Building upon the insights gained in the previous chapter, we established that the One-for-All approach utilizing CT-BERT outperformed all other methods and pre-trained models. Therefore, this chapter delves deeper into the One-for-All approach by exploring ensemble and multi-task learning techniques and experimenting with related datasets and data augmentation methods. Our goal is to ascertain whether these additional experiments can enhance the performance of the One-for-All approach. To this end, we seek to address the following questions:

- Will advanced techniques such as ensemble learning and multi-task learning contribute to improving our model?
- Is ChatGPT a viable option for Text Data Augmentation in our specific case?
- Can incorporating other language resources strengthen the ability of One-for-All to detect COVID-19-related conspiracy theories?

We begin by outlining the experimental setup, followed by a comprehensive analysis of the various methods employed. Ultimately, we compare and discuss the results in the chapter's concluding section.

6.1 Deep ensemble methods

Ensemble learning methods have significantly enhanced various NLP tasks, including spam detection, fake news detection, and offensive language detection [3, 27, 31]. Combining multiple models, ensemble methods leverage their strengths and mitigate weaknesses, resulting in improved performance. While ensemble methods offer benefits such as reduced overfitting, they can be computationally demanding and require additional memory and storage resources.

In Chapter 5, we trained multiple One-for-All approaches using different pre-trained models and cross-validation folds. Based on the highest validation MCC score, we selected the best model from the nine cross-validation folds. This section delves deeper into these models and explores their combined predictions to enhance the performance of the CT-BERT-based One-for-All approach. Initially, we employed a primary ensembling method outlined in Sub-section 5.5.1, where nine models were trained for each pre-trained model using the One-for-All approach. The best model was chosen based on the highest validation MCC score. However, this approach may not yield the best result on the hold-out set. Therefore, more advanced techniques that consider the collective knowledge of all models are required.

6.1.1 Averaging ensemble method

The initial ensembling method employed was the *averaging ensemble method*, which combines the outputs of multiple models to enhance prediction scores and improve generalization. By leveraging the collective wisdom of the models, this method mitigates overfitting and achieves superior performance on unseen data.

Given a set of M base models $m_1, m_2, \ldots, m_M$, each trained on a dataset D of N samples $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, where x_i represents the text data and y_i represents the corresponding conspiracy categories, the goal of the averaging ensemble method is to create a single ensemble model E that combines the outputs of the base models in a manner that improves overall predictive performance. The averaging ensemble method computes the output of E as the mean of the outputs of the individual base models. Formally, for a given input x, the ensemble model E produces the output $E(x)$ as follows:

$$E(x) = \frac{1}{M} \sum_{j=1}^{M} m_j(x) \tag{6.1}$$

where $m_j(x)$ denotes the output of the base model m_j for the input x, where M represents the number of base models in the ensemble. The method calculates the mean of the class probabilities predicted by the base models, and the final class is assigned based on the highest mean probability. It assumes that the base models are diverse and possess complementary strengths, enabling the ensemble to leverage their combined knowledge for more accurate predictions. This approach is particularly effective when the base models exhibit low error correlation, as the ensemble can mitigate individual model errors and decrease overall variance.

6.1.2 Max voting

Max voting, or majority voting, is a model aggregation technique that harnesses the collective knowledge of multiple models to enhance perfor-

mance and mitigate overfitting on unseen data. Like the averaging ensemble method, the Max voting method aims to create a single ensemble model E by combining the class predictions of a set of M base models $m_1, m_2, \ldots, m_M$. These base models are trained on a dataset D comprising N samples $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$, where x_i represents the input features and y_i represents the corresponding class labels. The Max voting method computes the output of E as the class with the highest number of votes among the individual base models. Formally, for a given input x, the ensemble model E produces the output $E(x)$ as follows:

$$E(x) = \text{argmax}\, c \left(\sum j = 1^M I(m_j(x) = c) \right) \tag{6.2}$$

Where $m_j(x)$ denotes the class predicted by the base model m_j for the input x, $I(m_j(x) = c)$ is an indicator function that equals one if the predicted class is c and 0 otherwise, and M represents the number of base models in the ensemble. The Max voting method assumes that the base models are diverse and have complementary strengths, which allows the ensemble to exploit their combined knowledge to make more accurate predictions. This method is particularly effective when the base models exhibit low correlation in their errors, as the ensemble can cancel out individual model errors and reduce overall variance.

6.1.3 Execution of the experiments

The experiments were executed with the same setup as in Chapter 5, and all experiments were done with the One-for-All approach. Furthermore, we applied the following models to achieve the ensemble methods:

1) **CV-Based ensembles**: here, we applied every nine models, which were achieved from the nine cross-validation folds, for each of the six chosen pre-trained models.

2) **Optimal Model Ensembles**: here we applied ensembling of the six best models from 5.6.

3) **Top Two Model Ensembles**: ensembling of BERTweet-Large and CT-BERT.

4) **Top Three Model Ensembles**: ensembling of BERTweet-Large, Twitter-RoBERTa-Large and CT-BERT.

5) **Top Four Model Ensembles**: ensembling of RoBERTa-Large, BERTweet-Large, Twitter-RoBERTa-Large and CT-BERT.

6.2 Data Augmentation techniques

Figure 5.4 shows that the **Fake virus** and **Intentional Pandemic** categories are among the lightest colors, meaning that these categories are the most difficult to classify. There could be various reasons for that. One of the

Model	Holdout MCC	Max Voting	Averaging
BERT-Base-Uncased	0.660	0.655	0.659
BERTweet-Covid19-Base-Uncased	0.653	0.664	0.674
RoBERTa-Large	0.680	0.712	0.718
Twitter-RoBERTa-Large	0.713	0.734	0.737
BERTweet-Large	0.727	0.728	0.734
CT-BERT	0.765	**0.774**	0.772
Optimal Model Ensembles	X	0.735	0.746
Top Two Model Ensembles	X	0.737	0.761
Top Three Model Ensembles	X	0.751	0.758
Top Four Model Ensembles	X	0.733	0.752

Table 6.1: The performance of the CV-based ensembles on holdout set. In addition, the table also contains the result of the ensemble of the best models.

reasons could be that the data has not been annotated so that there is a clear difference between the subcategories in these categories, meaning that there could be overlap among all subcategories. To investigate this claim, we will apply several augmentation techniques to enrich these two categories with more samples and then train several models on these sets.

6.2.1 Easy Data Augmentation (EDA)

The first augmentation technique is based on EDA[1] [28], which is an easy data augmentation technique initially constructed for boosting performance on text classification tasks. According to the creators of EDA, using EDA has shown improvements in five NLP classification tasks. EDA consists of the following components:

- **Synonym Replacement**: Randomly choose words from the text sample and replace each of these words with one of its synonyms chosen at random.
- **Random Insertion**: Find a random synonym of a random word in the text sample and insert that synonym into a random position in the sentence.
- **Random Swap**: Randomly choose two words in the sentence and swap their positions.
- **Random Deletion**: For each word in the sentence, randomly remove it with a probability.

All these components, except the first, are repeated N times.

[1]The implementation is available here: https://github.com/jasonwei20/eda_nlp

6.2.2 Augmentations with ChatGPT

Our second augmentation technique uses GPT-4 by Open-AI's ChatGPT services[2]. ChatGPT is a highly advanced language model developed by Open-AI based on the GPT-4 architecture. It analyzes vast data, enabling it to understand the context and respond to user inputs with coherent and relevant information. The primary purpose is to assist users in various tasks, such as answering questions, providing recommendations, generating text, and facilitating engaging conversations. During the release of ChatGPT, it has been used for several tasks and has been researched a lot. Due to its text-generating capability has also been used for data augmentation and has improved text classification for various problems [18].

In our case, we provided ChatGPT the description of the **Fake virus** and **Intentional Pandemic** categories from Section 3.1 with the description of the subcategories. Furthermore, we asked the chatbot to provide examples of the different subcategories for each category. More specifically, we asked ChatGPT the following queries after providing it with information on the categories:

- Can you give more examples of the Promotes/Supports Conspiracy?
- Can you give more examples of the Promotes/Supports Conspiracy? Please use more fantasy and dramatization.
- Can you give more examples of the Promotes/Supports Conspiracy? Please use words such as `[MOST OCCURRED WORDS]`

This process was repeated with the **Discusses Category**, but we did not use ChatGPT to make samples for the **Non-Conspiracy**. In the last question, we asked ChatGPT to make samples based on the most occurred words. Here we provided the chatbot with the most occurred words that were found in each of the subcategories. Instead, we performed this process independently and twice for both categories. We achieved roughly 40 samples for each subcategory and 160 extra samples in total, and these samples were added to the training dataset, which was used to train new models. We first a model based on the samples for the **Fake virus** category, then we trained a model based on samples for the **Intentional Pandemic**. Lastly, we trained a model with a combination of both augmented samples.

6.3 Experiments with the fine-grained version of COCO dataset

This section presents experiments conducted on the fine-grained version of the dataset, as described in Section 3.3. The fine-grained version comprises two modified versions of the COCO dataset: the *misinformation detection* set and the *conspiracy recognition* set. The goal of experimenting with these

[2] https://chat.openai.com/

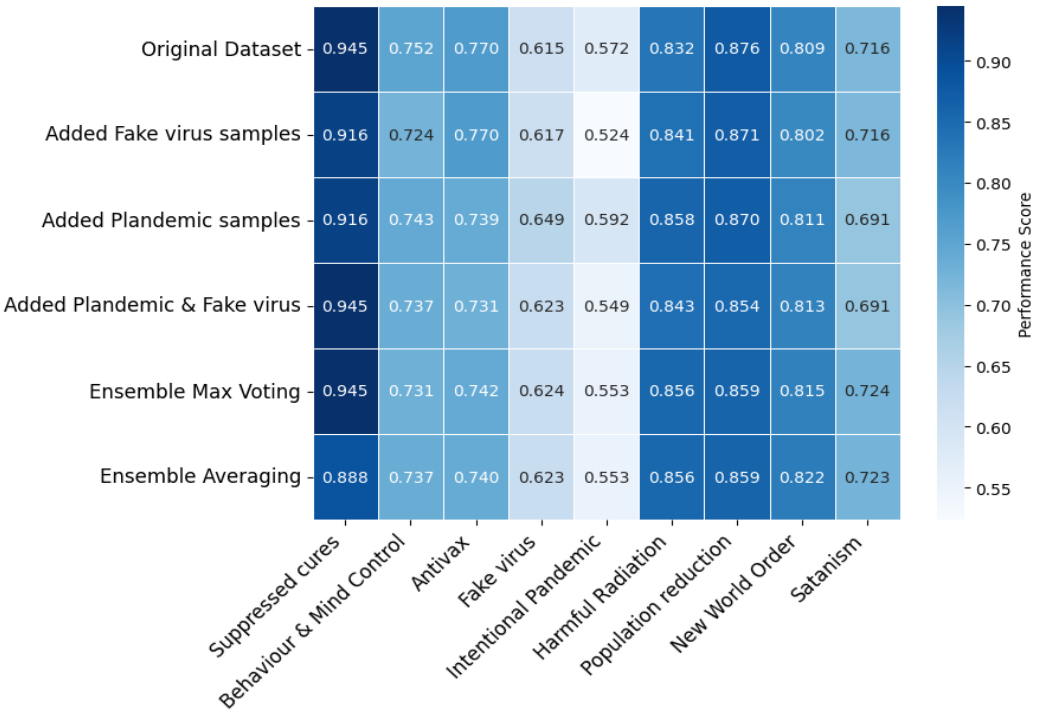


Figure 6.1: Heatmap displaying the performance of models trained with augmented samples compared to the original dataset.

versions, rather than the original dataset, is to investigate their potential for improving CT-BERT performance. This is achieved by training two models—one for misinformation detection and another for conspiracy recognition—and combining their outputs to predict the COCO dataset. The conspiracy recognition set is employed to identify whether a tweet mentions any of the nine conspiracies. The misinformation detection set determines whether a tweet promotes/supports, discusses, or is unrelated to a conspiracy. It is important to note that the misinformation detection set does not consider any specific conspiracy; it will be detected as long as a tweet discusses a conspiracy. By combining the predictions from these two sets, we obtain predictions for the COCO dataset since we first recognize the conspiracies and then detect the type of misinformation associated with that conspiracy.

6.3.1 A simple example

To clarify this process, consider the following fictitious tweet that supports the **Suppressed cures** conspiracy:

```
X = "I can't believe the government is hiding the fact that there
are effective treatments for COVID-19.  They're just trying to make
money off vaccines".
```

Applying the conspiracy recognition model to this sample would obtain the first equation of Equation 6.3. This prediction indicates that the tweet mentions the **Suppressed cures** conspiracy, but it does not reveal the type of misinformation being spread. To ascertain this information, we apply the misinformation detection model. By combining these outputs, we obtain the final prediction for the COCO dataset:

$$\begin{aligned} Model_{\text{Conspiracy-Recognition}}(X) &= \texttt{[1,0,0,0,0,0,0,0,0]} \\ Model_{\text{Misinformation-Detection}}(X) &= \texttt{3} \\ Model_{\text{COCO}}(X) &= \texttt{[3,0,0,0,0,0,0,0,0]} \end{aligned} \tag{6.3}$$

6.3.2 Setup and training

The configuration and training process for the misinformation detection model is identical to those of the One-for-One approach, in which the model is trained for text classification with three categories. The training of the conspiracy recognition model is conducted similarly to the One-for-All approach but with 18 output nodes instead of 27. Moreover, the training employs the `AdamW` optimizer and the `ReduceLROnPlateau` scheduler, with parameters consistent with those presented in Table 5.1. However, because the conspiracy recognition set is binary, we deployed sigmoid as a loss function.

6.3.3 Enriching our dataset with other datasets

Section 3.4 introduced the open-source COVID-19-related datasets developed to tackle fake news detection. This section will combine these datasets to enrich the COCO with more data samples and aim to improve the overall best model. Among the open-source COVID-19-related datasets, we have the COVID-19 Fake News Dataset [74], a large dataset of 10700 social media posts and articles of real and fake news on COVID-19. In addition, we have the COVID-19 category dataset developed by the authors of the CT-BERT [69] to evaluate and compare their model on it. The labels in this dataset are either personal narratives or news.

The challenge when combining these datasets with the COCO dataset is that the labels differ. As mentioned above, the label of the COVID-19 Fake News Dataset is **Real news** and **Fake news**, and the labels of the COVID-19 category dataset is **Personal narrative** and **News**. We have decided to combine these new labels with the misinformation dataset, the fine-grained version of the COCO. In the COVID-19 category dataset, the **Fake News** is quite similar to the **Promoting/Supporting Conspiracy** category, and the **Real News** is close to the **Non-Conspiracy**. Furthermore, we will experiment with the following extension of the dataset:

1) The first experiment consisted of adding the COVID-19 dataset with the following instructions:

I) **Fake news** $\rightarrow$ **Promoting/Supporting Conspiracy**

II) **Real news** $\rightarrow$ **Non-Conspiracy**

2) The first experiment consisted of adding the COVID-19 category dataset with the following instructions:

I) **Personal narrative** $\rightarrow$ **Promoting/Supporting Conspiracy**

II) **News** $\rightarrow$ **Non-Conspiracy**

Based on these new extended datasets, the misinformation detection model was trained and used with the conspiracy recognition model to predict on COCO, as described in the sections above.

6.4 Advanced Multi-Task Learning and Diverse Dataset Integration

This section will continue experimenting with our One-for-All approach with CT-BERT and the fine-grained version of the dataset. We aim to improve the performance of the best model by enriching it with these datasets and relying on them to share a typical statistical relationship. To achieve this, we will build a new multi-task learning approach. This approach is inspired by Dai et al. [19] where they proposed BERT-based multi-task learning for offensive language detection at SemEval-2020[3] workshop. Their proposed approach consisted of training one primary BERT model, which was then used to predict three tasks related to offensive language detection. The benefit of such an approach is that it can help the model generalize better because multiple tasks introduce more noise and prevent the model from over-fitting. Additionally, there are situations where learning features through one task may be challenging but comparatively easier through another task. Ultimately, MTL offers a valuable advantage by providing supplementary guidance for one task while also allowing for the acquisition of additional information by eavesdropping on other tasks.

6.4.1 The structure of the multi-task learning model

We attempted to adopt the Multi-Task Learning (MTL) method proposed in [19] and adapted it to our specific requirements. The original approach involved a shared BERT model for three sub-tasks, each having a separate module comprising a Recurrent Neural Network (RNN) with Long-Short Term Memory (LSTM) cells. The input $\mathbf{X}$ was first processed by the shared BERT, followed by each sub-task module utilizing the contextualized embeddings generated by BERT to produce a probability distribution for its target labels.

[3] https://alt.qcri.org/semeval2020/index.php?id=tasks

Model	Parameter	Parameter values
LSTM	dropout rate	0.1
LSTM	number of layers	1
LSTM	hidden size	100
LSTM	Combining of hidden layers	concat
FFNN	Number of layers	2
FFNN	Size of layers	1024, 512, 300

Table 6.2: The chosen parameters for LSTM and FFNN models in the MTL approach.

In our scenario, the COCO is a multi-label multi-class dataset, necessitating a different approach. Consequently, we developed the MTL model as follows:

- **A shared CT-BERT backbone** is common among N_{Tasks} sub-tasks. We will experiment with the COCO dataset and its two alternative variants; conspiracy recognition and misinformation detection datasets. In other words, N_{Tasks} is three.
- **Middle modules** consist of either LSTM cells or a simple FFNN model. These modules are built atop the shared CT-BERT model but are specific to each task or dataset.
- **Final modules** are tailored to individual tasks.

The experiments were conducted with the same parameters as displayed in Table 5.1, but we used additional parameters for the LSTM and the FFNN models, as shown in Table 6.2.

6.5 Discussion & Results

In this chapter, we carried out several experimentation rounds to improve the results from previous chapters. Among the chosen methods, we experimented with ensemble methods, data augmentation techniques, using the fine-grained version of the dataset, and other advanced methods to enhance the performance of the previous models.

6.5.1 Deep ensemble methods

We utilized two sophisticated ensemble techniques, namely the averaging ensemble and max voting methods, to harness the collective wisdom of the "crowd" for predicting unseen data. These ensembles were executed on nine training folds instead of the previous model selection approach based on the potentially biased validation Matthews Correlation Coefficient (MCC) score. By aggregating the predictions of all models, ensemble methods offer improved generalization. We applied these ensembles

to the nine folds of each of the six chosen models, considering the optimal, top two, top three, and top four best models. The results of these runs are presented in Table 6.1, which shows that the deep ensemble of CT-BERT, based on cross-validation folds, outperformed the others. The highest MCC score achieved was **0.774** using the max voting method, and except BERT-Base, the performance of all models improved when employing the cross-validation-based ensemble methods. However, combining the top models did not prove advantageous and resulted in a lower MCC score than the best individual model.

Ensemble learning, although powerful and effective in many scenarios, and our case, does present certain drawbacks. The computational cost, which encompasses the increased processing power, time, and memory resources needed to train, validate, and make predictions using an ensemble of models instead of a single model, is a crucial consideration. In our case, we trained nine different models to attain an MCC score improvement of 0.774, compared to a single model that yielded an MCC score of 0.765. This marginal improvement of only 0.009 raises questions about the trade-off between the benefits of ensemble learning and the associated computational expenses.

In addition to the computational cost, there are other factors to consider when evaluating the trade-off between ensemble learning and using a single model. In our case, the marginal improvement in the MCC score of 0.009 may not be substantial enough to justify the increased complexity, reduced interpretability, and potential scalability issues associated with ensemble learning. The increased complexity of ensemble learning can make it more challenging to understand, implement, and maintain the overall system. This, in turn, could lead to a higher likelihood of errors and increased difficulty in troubleshooting and fine-tuning the models. Scalability can also become an issue when dealing with large datasets. Given these additional considerations, it is crucial to carefully assess the benefits and drawbacks of ensemble learning about the specific problem at hand.

6.5.2 Data augmentation

We are confronted with two primary obstacles. The initial challenge stems from the uneven distribution of the COCO dataset, which complicates the development of effective classification models (refer to Figure 3.1). The second issue arises from the inextricable nature of certain categories, such as **Fake virus** and **Intentional pandemic**, from their corresponding subcategories. Observing the heatmap in Figure 5.4, it becomes evident that CT-BERT struggles to differentiate these subcategories from one another in contrast to other categories, subsequently leading to a diminished MCC score.

Using two distinct methods, we have enriched the dataset with augmented

	Model	Holdout MCC
Original	Misinformation detection	0.719
	Conspiracy recognition	0.783
	Combination of both	0.738
COVID-19 dataset	Misinformation detection	0.732
	Conspiracy recognition	0.783
	Combination of both	**0.747**
COVID-19 Category	Misinformation detection	0.729
	Conspiracy recognition	0.783
	Combination of both	0.742

Table 6.3: The result of the fine-grained version of the COCO dataset.

samples to address both challenges. The first employs Easy Data Augmentation (EDA) [28], a straightforward tool for text classification data augmentation. The second approach leverages ChatGPT based on the GPT-4 model, wherein we requested examples for specific categories. The heatmap depicted in Figure 6.1 reveals the One-for-All strategy's performance when incorporating samples from ChatGPT. The model demonstrated a slight improvement in predicting the **Fake virus** and **Intentional pandemic** categories when an additional **Intentional pandemic** sample was provided. Nevertheless, the overall Matthews Correlation Coefficient (MCC) score for the model decreased compared to the original model. This may be attributed to the close relationship among the nine conspiracy categories. In comparison, the ChatGPT samples might have enhanced the two selected categories, but they potentially had a detrimental effect on the others.

6.5.3 Fine-grained version of dataset

The experiment of the fine-grained dataset version was conducted with the motivation that perhaps the CT-BERT generalizes better with feature-engineered categories. Table 6.3 shows the MCC scores for the holdout dataset for each of the fine-grained versions of the dataset and their combination MCC on the COCO holdout set. With an MCC score of 0.738, it is clear that this type of future engineering of the categories in COCO did not help the model to generalize, but it worsened the performance. However, it is interesting to note that the model performs quite well on detecting the type of conspiracy, i.e., conspiracy recognition task, where it achieved an MCC of 0.783. Based on this, we can conclude that the reason for the One-for-All approach struggle is that it blunders to distinguish between **Discusses Conspiracy** and **Promoting/Supporting conspiracy** subcategories. Nonetheless, if we merge these two categories, we achieve a higher MCC score, as shown in Table 6.4.

We also conducted experiments incorporating additional datasets into the misinformation detection version of the COCO dataset and trained

Category	Conspiracy recognition	Combination
Suppressed cures	0.950	0.799
Behaviour and Mind Control	0.733	0.702
Antivax	0.756	0.758
Fake Virus	0.612	0.590
Intentional Pandemic	0.586	0.558
Harmful Radiation/ Influence	0.884	0.856
Population reduction	0.907	0.869
New World Order	0.813	0.783
Satanism	0.805	0.723
Average	0.783	0.738

Table 6.4: Comparison of MCC score of categories from the conspiracy recognition task and combination of it with misinformation detection task to obtain predictions on COCO dataset.

the CT-BERT models using these new datasets. Table 6.3 presents the MCC scores obtained with these new datasets. Integrating other datasets contributes to better model generalization. Specifically, including both the COVID-19 dataset and the COVID Category dataset, we have improved the performance of the misinformation detection task. However, the difference in MCC scores is not substantial enough. The combined performance of misinformation detection and conspiracy recognition tasks yields an MCC score of 0.747, which is still lower than the MCC score in the previous chapter.

6.5.4 The result of advanced multi-task learning

The proposed multi-task learning model, incorporating a shared CT-BERT backbone and task-specific middle and final modules, was evaluated on the COCO dataset. The MTL model achieved an MCC score of 0.7047 when utilizing a linear classifier, and when employing LSTM as the classifier, the MTL model achieved an MCC score of 0.6123.

6.6 Conclusion

In this chapter, we have conducted several experiments to enhance the performance of the One-for-All approach from the previous. The first method was based on deep ensemble methods, specifically averaging ensemble and max voting methods. By aggregating predictions from multiple models, ensemble methods offered better generalization. In the study, the ensemble of CT-BERT outperformed other models, achieving the highest MCC score of 0.774. However, the improvement was marginal, raising questions about the trade-off between ensemble learning benefits and computational costs. Additionally, ensemble learning can increase complexity, reduce interpretability, and present scalability issues. Thus, it is essential to carefully assess the benefits and drawbacks of ensemble learning for a specific prob-

lem.

In addition, we experimented with data augmentation techniques with EDA and ChatGPT to tackle the challenges of uneven dataset distribution and the difficulty in distinguishing between specific categories in the COCO dataset. The One-for-All strategy's performance improved slightly in predicting **Fake virus** and **Intentional pandemic** categories when additional samples were provided using ChatGPT. However, the overall MCC score decreased, potentially due to the close relationship among the conspiracy categories, indicating that augmentation may have had a detrimental effect on other categories. Furthermore, a fine-grained version of the dataset was used in an experiment to improve CT-BERT's generalization, but it resulted in a worsened performance. However, the model performed well in conspiracy recognition tasks. Additional datasets were incorporated for misinformation detection, improving model generalization slightly but not substantially. The combined performance of misinformation detection and conspiracy recognition tasks remained lower than the previously achieved score.

Chapter 7

Experiments on intelligence processing units

We have analyzed NLP-based models and evaluated their performance on the COCO dataset, identifying that the CT-BERT with the One-for-All approach yielded the most promising results. These experiments used a single GPU on an Nvidia Volta A100 within the eX3 computing cluster. As machine learning has advanced in recent years, new processing units have been designed to accelerate the training of large neural-based models. One example is the Intelligence Processing Unit (IPU) developed by Graphcore, a state-of-the-art High-Performance Computing processor tailored for AI acceleration. In this chapter, we will investigate the performance of large transformer-based pre-trained models in the context of conspiracy detection tasks, emphasizing execution speed. By comparing the outcomes of these models, we will examine the trade-offs between accuracy and computational efficiency and assess the potential advantages of employing sophisticated hardware like IPUs to boost the performance of conspiracy detection models.

7.1 Intelligence Processing Unit

Deep learning has been developed and progressed rapidly in the last few years. As a result, the models are getting smarter, bigger, and more complex. Training such models is becoming more difficult and computationally demanding, necessitating the development of specialized hardware and optimizations to process and manage vast data efficiently. In response to this challenge, companies like Graphcore have emerged, designing innovative solutions such as Intelligence Processing Units (IPUs) to accelerate and optimize the training and inference of deep learning models.

An Intelligence Processing Unit (IPU) is a specialized hardware accelerator that accelerates machine learning and artificial intelligence workloads. It is optimized for performing complex mathematical calculations required for deep learning algorithms and is highly parallel and efficient in performing matrix multiplication operations. Historically, GPUs have

accelerated training times compared to CPUs but are limited by their design for 2D matrices and dependency on large data batches, which can impact model quality. With IPUs, one can overcome these constraints, dramatically speeding up training and enabling the exploration of a more comprehensive array of machine learning models and algorithms for more flexible and powerful parallel computing.

7.1.1 The hardware

The Graphcore IPU comprises units known as *tiles* that process data simultaneously. Each tile consists of a core and a small amount of SRAM memory, a fast, reliable memory type. Each core operates six threads simultaneously using a technique known as *temporal multithreading*, enhancing performance and resource use. This process differs from the simultaneous multithreading common in CPUs and GPUs. The IPU's unique thread arrangement characterizes it as a barrel processor. IPU instructions, including data loading and storage, take six cycles, eliminating delays for individual threads.

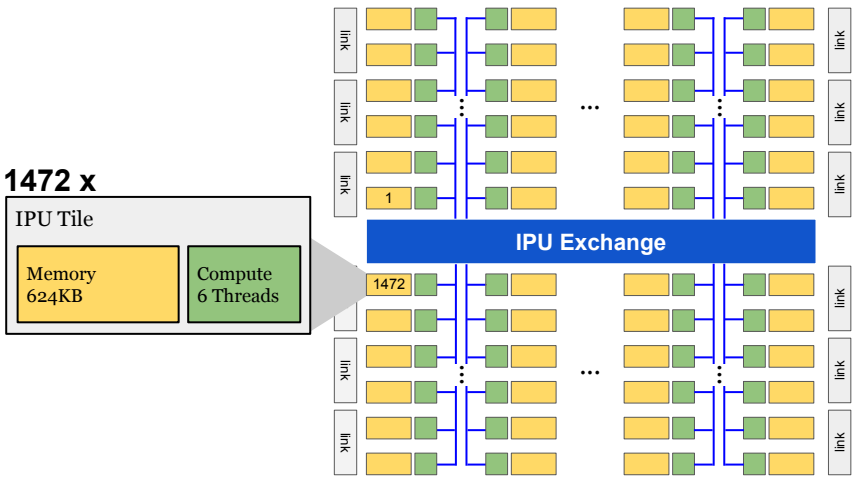


Figure 7.1: Tile layout on the GC200 IPU processor.

The IPU comprises tiles grouped into *islands* and *columns*, forming the IPU structure as shown in Figure 7.1. The number of cores varies with the IPU model, and key features of the GC2 and GC200 IPUs are detailed in Table 7.1. The IPU's memory bandwidth is superior to CPUs or GPUs, as all cores can access memory simultaneously. Still, non-local data transfer necessitates moving it between tiles. A tile can transfer 4 bytes per cycle, amounting to $5.3GB/s$ or $7.83TB/s$ for all 1472 cores of the GC200. The IPU exchange network links the cores within the IPU, and the GC200 IPU can reach DRAM memory at around $20GB/s$.

Chip	GC2	GC200
Number of tiles	1216	1472
Number of threads	7296	8832
Memory per tile	256 KB	624 KB
Total SRAM memory	311 MB	918 MB
Memory bandwidth	46.6 TB/s	46.9 TB/s
Aggregate tile-to-tile bandwidth	7.78 TB/s	7.83 TB/s
Total chip-to-chip bandwidth	320 GB/s	320 GB/s
Clock frequency	1.6 GHz	1.33 GHz
FP32 compute	31.1 TFLOPS/s	62.5 TFLOPS/s

Table 7.1: Key architectural features of GC2 and GC200 IPU.

The IPU-Link manages data transfer between IPUs, similar to PCIe and Infiniband in CPU/GPU systems. Each IPU has 10 links, delivering a total bandwidth of 320 GB/s. IPU pairs are connected with 12 links, providing 192GB/s bandwidth. The remaining links connect to other IPUs at 64GB/s. Up to 32 pairs can link in a ladder setup, offering 128 GB/s bandwidth. Forming a torus from the ladder doubles this bandwidth. Multiple groups of 64 IPUs, known as PODs, can be linked via a Gateway Link. An individual IPU has a 150 W thermal design power (TDP), around half that of a comparable GPU. So, in power terms, an IPU pair matches a high-end GPU like the NVIDIA V100 or A100.

7.2 Deep learning with IPUs

There are multiple ways to work with the IPUs. Deep learning on IPUs is built on the *Poplar* framework, which follows a dataflow model. Programs are structured using layered graphs, with vertices in each layer alternating between representing states stored in multidimensional arrays called tensors and subroutines (called codelets) that transition from one state to the next. Each vertex in a computation layer has a codelet, and all codelets within a layer can be executed in parallel without race conditions. Data is distributed using the bulk-synchronous parallel (BSP) method, which emphasizes coordinated processing through distinct computational phases. In BSP, multiple processors execute tasks concurrently, with synchronization barriers ensuring the completion and consistency of results before moving to the next phase. This approach effectively balances computation and communication while minimizing conflicts, providing a structured approach to parallel processing. Tensor sizes and communication in each step are determined at compile time.

As of this date, various Python frameworks have been developed to facilitate deep learning on IPUs. One notable example is PopTorch[1], a software

[1] https://docs.graphcore.ai/projects/poptorch-user-guide/en/latest/

library created by Graphcore. This library serves as an interface between the widely-used deep learning framework PyTorch and IPUs. PopTorch offers a collection of PyTorch extensions that enable seamless IPU integration within existing PyTorch-based workflows. By harnessing the IPUs' massively parallel architecture, one can potentially enhance the speed and efficiency of training and inference in deep learning models. Furthermore, PopTorch compiles PyTorch models into Poplar executables while providing IPU-specific functions.

In addition, Hugging Face introduced the *Optimum* [2], which is an open-source library that supports deploying pre-trained transformers models on a variety of hardware platforms, especially on IPUs. Optimum offers tools that facilitate model parallelization and loading on IPUs, supporting training and fine-tuning across tasks already compatible with Transformers. Additionally, it seamlessly integrates with the Hugging Face Hub, providing compatibility with all available models right out of the box. Optimum is very similar to the built-in classes of Transformers, e.g., `Trainer` and `TrainingArguments`. However, the main difference is that to train and predict a transformers model; one must use `IPUTrainer`, which compiles the model to run on IPUs and performs training and evaluation. In addition, one must also define a `IPUConfig` class that specifies attributes and configuration parameters to compile and put the model on the device. The complete list of IPU configurations is available at Hugging Face website [3]. We deployed the CT-BERT, a BERT-Large, using the `Graphcore/bert-large-ipu` model. Finally, one must decide the PODs type to run the IPUTrainer on, which can be *POD4*, *POD8*, *POD16*, etc.

Lastly, PyTorch Lightning is an open-source, high-level framework built upon PyTorch that enables the deployment of deep learning models on IPUs, CPUs, and GPUs. The framework is designed to streamline and expedite deep learning models' training, development, and deployment processes. PyTorch Lightning enhances code readability and maintainability by providing a structured, modular approach to code writing. A significant advantage of this package is its ability to allow users to explicitly designate the number of IPUs they wish their models to run on. This feature sets it apart from alternatives like Optimum, which only permits the selection of the type of POD without the ability to control the number of IPUs utilized.

7.3 Experiments setup

Within this section, we shall provide an overview of the construction process of our code utilizing the Optimum package and a comparative analysis of prediction time.

[2]Optimum source code: https://github.com/huggingface/optimum
[3]IPU configurations: https://huggingface.co/Graphcore.

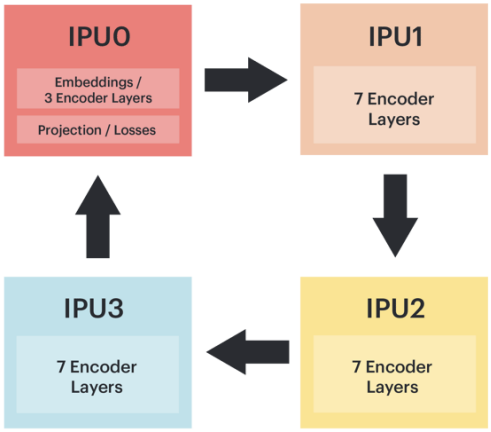


Figure 7.2: Model parallelism of BERT-Large model on IPU's POD4.

7.3.1 Building an NLP-model for conspiracy detection

To build an NLP model for conspiracy detection, a multi-class multi-label CT-BERT classification model, we load the entire model's parameters onto the IPUs using the Optimum library. In this way, CT-BERT runs an efficient pod by splitting the model across four IPUs and executing the model as a pipeline during training. CT-BERT, a large BERT model, is shared between all four IPUs in a POD4. IPU-0 contains the embedding layer, projection/loss layers, and three encoder layers, and the remaining 21 layers are evenly distributed over the other three IPUs, as shown in 7.2. To minimize on-chip memory usage, the CT-BERT model is employed with recomputation [16], eliminating the need to store intermediate layer activations for backward pass calculations. This technique is beneficial when training models. With multiple batches concurrently progressing through the pipeline, the volume of stored activations can be considerable unless recomputation is utilized. The optimizer state for the pre-training system is stored in Streaming Memory and loaded on demand during the optimizer step.

More practically, the CT-BERT model was deployed with the `IPUTrainer` with 27 number of outputs, and `Graphcore/bert-large-ipu` as the IPU configuration. Since the model runs on the entire IPU-POD4, we must declare parameters specific to distributed training. We chose the following `per_device_train_batch_size=40` and `per_device_eval_batch_size=40`; these parameters refer to the number of batch sizes that are processed on each IPU devices. In addition, we also the `gradient_accumulation_steps`, refers to the number of times gradients are accumulated before updating the weights of the model, to be at 8. Despite these changes, they were built similarly to NLP modeling Face's *Transformers* for multi-label classification.

To compare the IPU's training and evaluation time with GPU's upgraded

Hardware	Training time
IPU-POD4	6317 seconds
Single GPU	1684 seconds
Dual GPUs	1121 seconds

Table 7.2: The training time for training the One-for-All approach on IPUs and GPUs.

the code of the One-for-All approach from previous chapters to work with multiple GPUs. We implemented data parallel executing distributed data parallelism (DDP) [56], a technique to enable efficient training and evaluation of large deep learning models across multiple GPU devices. In a distributed data-parallel setup, the data is split into multiple subsets, and each subset is processed simultaneously by different GPUs. GPUs are particularly well-suited for this type of parallel processing because they have many processing cores simultaneously executing many small, parallel computations. Each GPU is assigned a subset of the training data during training, and the model is replicated across each GPU. Each GPU then computes the gradients for its subset of the data and sends those gradients to a central node, which aggregates the gradients and updates the model parameters. This process continues until the model converges to a satisfactory solution.

7.3.2 Comparative Analysis of Time Performance

We designed a two-part experiment to evaluate the time performance of IPUs and GPUs in the context of the large BERT NLP model. The initial phase involves training the One-for-All approach using both IPUs and GPUs, during which we record their respective time performances. Subsequently, the second phase entails running inference on the trained model using various data sizes. Specifically, we selected multiple tweet counts from the extensive dataset discussed in Chapter 8 to perform these experiments. The chosen tweet quantities for the inference tests are as follows:

- **Thousands of tweets**: 1K, 5K, 10K, 50K
- **Millions of tweets**: 0.1 M, 0.5M, 1M

Each tweet was randomly selected from the more extensive dataset and saved as a separate file.

7.4 Results & Discussion

In this section, we assessed the One-for-All approach on IPU-POD4, a single A100 GPU, and a pair of A100 GPUs to evaluate their performance concerning training and inference in NLP-based models. As illustrated in Figure 7.3, the IPU-POD, comprising four IPUs, significantly outperforms the

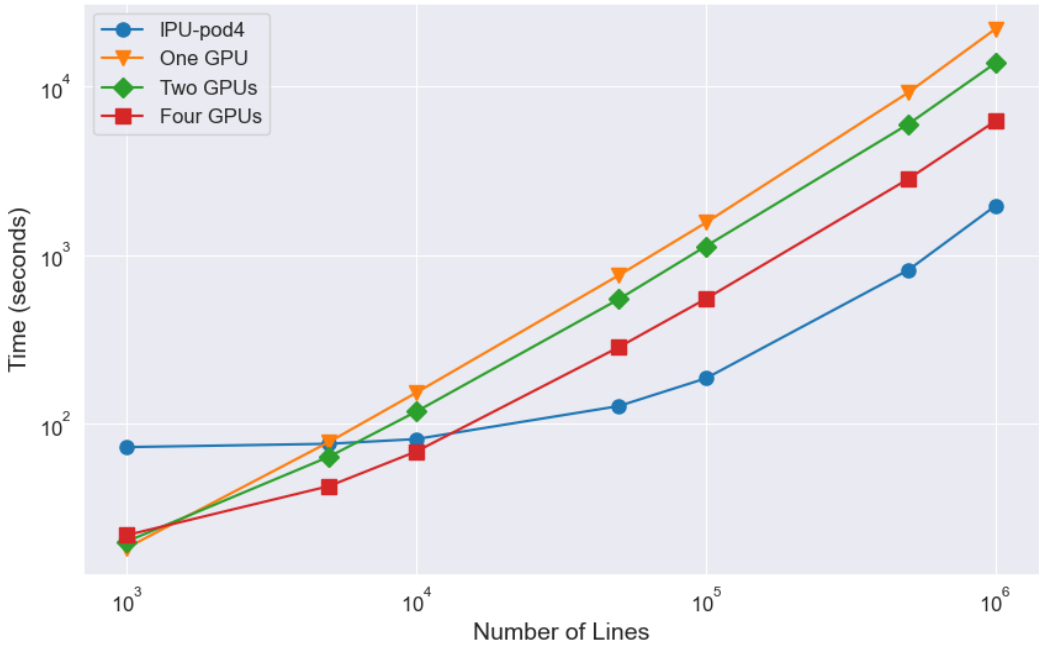


Figure 7.3: Inference time comparison of IPU-POD4 with one, two and four NVIDIA A100 GPUs on various amounts of dataset.

GPUs regarding inference time. As mentioned in Section 7.1.1, an individual IPU possesses a 150W TDP, roughly half the TDP of a competitive GPU like the NVIDIA A100. Consequently, a duo of IPUs is comparable to a single high-performance GPU in terms of power consumption. Nevertheless, our results demonstrate that IPUs are more efficient and faster for inferencing on larger datasets. According to the same figure, the inference time for GPUs increases linearly as the dataset size expands, while IPUs exhibit a parabolic growth pattern with a slower rate of increase.

Based on Table 7.3, the IPU-POD4 showcases impressive performance compared to one and two NVIDIA A100 GPUs across various dataset sizes. As the number of tweets in the dataset increases, the IPU-POD4 maintains a relatively low inference time while the speedup over single and dual GPUs becomes more pronounced. The speedup is minimal for smaller dataset sizes, such as 1,000 lines (0.25 and 0.27 for one and dual GPUs, respectively). However, as the dataset grows, the IPU-POD4's advantage becomes more evident. At 50,000 lines, the IPU-POD4 achieves a speedup of 5.98 and 4.31 over one and dual GPUs, respectively. The performance gap further widens with larger datasets. For instance, at 1,000,000 lines, the IPU-POD4 demonstrates a remarkable speedup of 11.23 and 7.04 over one and two NVIDIA A100 GPUs, respectively. These results indicate the superior efficiency and scalability of the IPU-POD4 in handling larger datasets for inference tasks compared to single and dual NVIDIA A100 GPUs.

Despite the impressive, efficient inference of the IPUs on large datasets, it

Number of lines	IPU-POD4	Single GPU	Dual GPU	Four GPUs
1000	72.32 s	0.25	0.27	0.30
5000	75.67 s	1.03	0.84	0.56
10000	80.74 s	1.89	1.46	0.84
50000	126.72 s	5.98	4.31	2.24
100000	185.19 s	8.41	6.10	2.97
500000	814.26 s	11.33	7.36	3.48
1000000	1961.51 s	11.23	7.04	3.18

Table 7.3: The original IPU-POD4 inference time and speedup comparison with one NVIDIA A100 GPU and two NVIDIA A100 GPUs.

struggles with the training time. As Table 7.2 displays, the training time of the IPUs is much higher than for the GPUs. The IPU-POD4 requires 6,317 seconds for training, while the single GPU takes 1,684 seconds. In this situation, the single GPU performs better than the IPU-POD4, achieving a notably shorter training time and a relative speedup of 3.7 times. While the IPU-POD4 exhibits superior performance in inference tasks, as seen in the previous table, the single GPU appears to be more efficient when training the One-for-All approach based on the provided data. Table 7.3 compares the inference time between IPU-POD4 and four NVIDIA A100 GPUs for various numbers of lines. For 1000 lines, IPU-POD4 takes 72.32 seconds, whereas four GPUs require only 0.30 times that duration. In this case, the GPUs significantly outperform IPU-POD4. As the number of tweets increases, the performance gap between IPU-POD4 and the four GPUs narrows. For instance, with 100,000 lines, IPU-POD4 takes 185.19 seconds, while four GPUs need 2.97 times less time. The GPUs consistently offer faster inference times than IPU-POD4, with four GPUs providing the best performance for files up to 50,000 lines in size. However, at 1,000,000 lines, IPU-POD4 outperforms the four GPUs, taking 3.18 times less time for the task.

Graph compilation is the primary factor contributing to the extended training time on IPUs. This process optimizes and translates high-level machine-learning models into a format that can be efficiently executed on IPU hardware. Graph compilation ensures that IPUs provide high-performance computing and energy efficiency when running machine learning workloads. However, frequently compiling and swapping graphs between training and validation phases can result in longer training times. To mitigate this issue, several strategies can be employed. One solution is to pre-compile the graphs before executing the program and store the corresponding executables on persistent storage, allowing subsequent executions without impacting throughput. Storing relevant weights in checkpoint files also enables separate validation after a run, minimizing executable swapping. Alternatively, with some extra engineering effort, training and validation can be performed using the same graph by

suppressing the training update, further optimizing the process.

7.5 Conclusion

In this chapter, we explored Intelligence Processing Units (IPUs) and performed various experiments using them. We developed the One-for-All model, a multi-class multi-label classification approach for conspiracy detection, and trained it on IPU-POD4, a system comprising four interconnected Graphcore IPUs. This model achieved an MCC score of 0.756. Additionally, we conducted experiments comparing the training and inference speed of IPUs and GPUs, where we found that IPUs are much slower when fine-tuning large language models, as shown in Table 7.2, but they outperform GPUs in terms of efficiency, mainly when processing large datasets. This investigation highlights the potential advantages of leveraging IPUs for deep learning tasks in terms of performance and scalability. Building on these findings, future research can explore the optimization and adaptation of other deep learning models and architectures for IPUs. Moreover, it is essential to investigate techniques that minimize the impact of graph compilation on training time, enabling more efficient utilization of IPUs for training and inference tasks.

Chapter 8

Inference on big data

In the preceding chapter, we discovered that executing inference on IPUs using an NLP-based model yields superior performance compared to state-of-the-art GPU accelerators. This outcome is highly beneficial, enabling efficient inference on large-scale datasets and further enhancing our capabilities in processing and analyzing big data. In this chapter, we will present the details of the big data and outline the inference process. This inference aims to determine the most widely spread conspiracies on Twitter and identify those that generated the most discussion.

8.1 Motivation

As described in the introduction of the thesis, combating misinformation and fake news is a critical challenge in this digital age, as spreading false information can impact individuals and societies. Addressing misinformation involves promoting digital literacy, fact-checking, improving content moderation, and engaging in public awareness campaigns. These efforts enable informed decision-making and promote a fact-based society where rational discourse thrives. Therefore, performing a large-scale NLP-based inference for detecting COVID-19 conspiracy theories is essential. We can effectively identify, analyze, and mitigate the spread of such conspiracies by leveraging advanced NLP techniques and machine learning algorithms.

The end result of the inference is a large dataset of around 2.5 tweets with nine COVID-19-related conspiracy theories. This dataset can help facilitate research and more understanding of the COVID-19 conspiracies and provide valuable insights into their origins, evolution, and societal effects. By analyzing this dataset, we can identify common themes, patterns, and factors contributing to the spread of COVID-19 conspiracy theories. Furthermore, the dataset can be a valuable resource for advancing natural language processing and machine learning techniques in the ongoing research against misinformation and conspiracy theories.

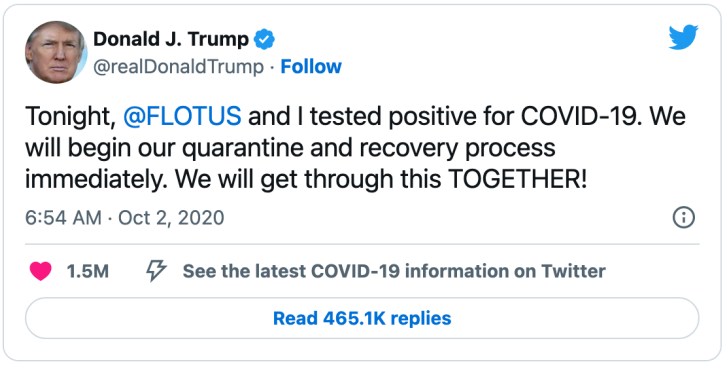


Figure 8.1: Former US President Donald Trump tweeting that he tested positive for COVID-19, which led to a massive discussion on Twitter regarding the virus, as shown in Figure 8.5.

8.2 Big data

The big data collection process comprised multiple stages, including accumulating tweets related to the COVID-19 pandemic between January 17, 2020, and November 30, 2021. A custom-distributed Twitter scraping framework known as FACT (cited as [86]) was employed with the Twitter search API, targeting specific COVID-19-associated keywords. Acknowledging the rarity of conspiracy-related tweets, a keyword list linked to conspiracy theories was used to perform a focused text search. In total, 6,286,886,977 tweets were gathered. However, these tweets included duplicates, retweets, replies, and quotes. Furthermore, they contained identical content regarding tweets, retweets, replies, and quotes, albeit with different IDs and sources. Finally, the collection also encompassed general retweets, such as when users retweeted pre-COVID-19 era tweets to convey a particular message.

The first stage of preprocessing of this massive dataset was done by removing the duplicates, which ended up with 2,570,581,178 in tweets. Among these tweets, we had retweets in general, replies, and quotes that contained the same tweets with other tweet IDs. The final preprocessing stage removed the retweets that resulted in 381,136,088 tweets. To perform the inference, we worked with the 381 million datasets, not the 2.5 billion, but we can track each of these back to their original form.

8.3 Running the inference

We ran the experiments by splitting the whole data into smaller chunks. Each of these chunks, or files, contained around one million tweets, which resulted in 382 separate files. The tweets were in their original format, meaning that they contained URL links and user mentions, and due to the scraping of the data, they also contained some unexpected HTML tags,

Category	Non-conspiracy	Discusses conspiracy	Promoting or Supporting
Suppressed cures	99.034 %	0.034 %	0.932 %
Behaviour & Mind Control	99.906 %	0.047 %	0.047 %
Antivax	97.943 %	1.592 %	0.464 %
Fake Virus	98.776 %	0.623 %	0.602 %
Intentional Pandemic	99.517 %	0.135 %	0.348 %
Harmful Radiation	99.883 %	0.08 %	0.037 %
Population reduction	99.925 %	0.017 %	0.058 %
New World Order	99.928 %	0.006 %	0.067 %
Satanism	99.933 %	0.032 %	0.035 %

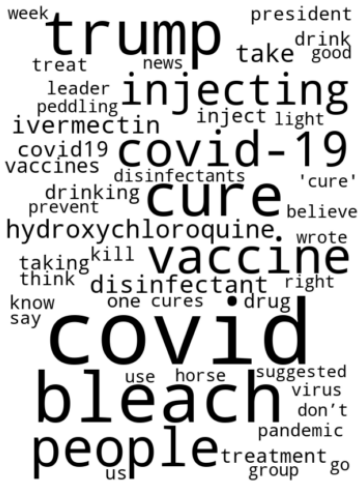
Table 8.1: The result of inference; the percentage of the total tweets in the big data for the three subcategories for each of the nine conspiracy categories.

e.g., `xa0` and `<200d>`. Therefore, the tweets were driven by a preprocessing function before feeding to the trained model. All the usernames were replaced with a common username token `@USERNAME`, and all the URLs were replaced with a common URL token `URL`. The preprocessing of each file took around two minutes. The inference was initialized with the IPU-POD4, resulting in around 35 minutes of prediction time for each file. The whole time for inference of the big data took around nine days and nights; however, we had to restart the prediction due to some minor errors caused by the data.

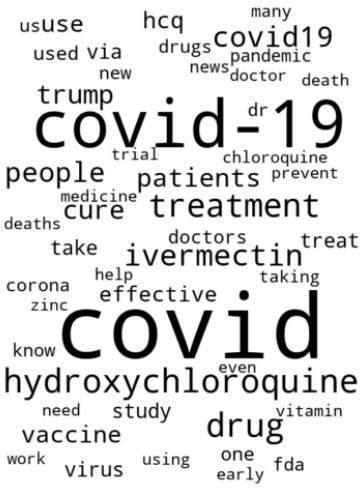
8.4 Results & Discussion

The big data contained around 381 million unique tweets, and we can trace all these tweets back to the 2.5 billion tweets, i.e., the inference covers much larger tweets. The inference of the big data took around 223 hours. Among the 381 million tweets, around 18,627,660 (4.89%) tweets were assigned with at least one *Discusses Conspiracy* or *Promoting/Supporting Conspiracy* subcategory.

Table 8.1 presents the analysis outcomes for each conspiracy category. The most prevalent conspiracy topic on Twitter is **Antivax**, accounting for 1.592% (6,068,439) of categorized tweets. It is important to note that this category excludes concerns regarding vaccine safety, effectiveness, or trustworthiness, as these are not conspiracy theories. Instead, the **Anti-vax** category pertains to beliefs that COVID-19 vaccines serve a concealed, malevolent purpose or can cause fatalities to manipulate population numbers. Numerous anti-vaccination demonstrations have occurred throughout the pandemic for various reasons, such as the belief that vaccines are unnecessary, ineffective, dangerous, or contain harmful substances like mercury and aluminum. Despite the range of motivations behind the anti-vaccination protests, the topic has become widely discussed on Twitter,

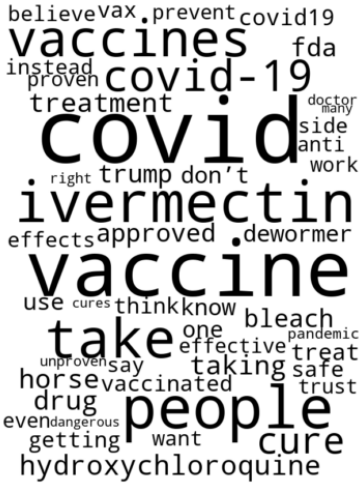


((a)) Discusses conspiracy

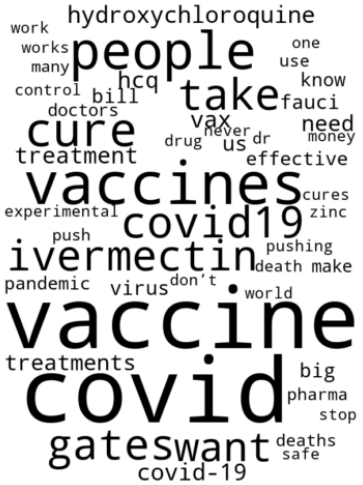


((b)) Promoting/Supporting conspiracy

Figure 8.2: The word cloud of subcategories *Discusses conspiracy* and *Promoting/Supporting conspiracy* for the **Suppressed cures**. These word clouds have been generated from one million tweets.



((a)) Discusses conspiracy



((b)) Promoting/Supporting conspiracy

Figure 8.3: The word cloud of subcategories *Discusses conspiracy* and *Promoting/Supporting conspiracy* for the **Antivax**.

Most occurred words	30-th Septer	1st October	2nd October	3rd October
'hoax'	49.91%	46.37%	79.52%	80.2%
'trump'	33.02%	24.6%	57.88%	54.52%
'china'	2.33%	1.38%	1.76%	1.93%
'tested'	0.43%	0.89%	4.18%	1.66%
'positive'	0.71%	0.86%	7.42%	3.35%
'fake'	12.07%	9.49%	13.49%	11.24%
'democratic'	2.53%	1.67%	3.8%	3.92%
'republican'	2.3%	2.46%	2.02%	4.34%
'quarantine'	0.45%	0.3%	4.77%	0.79%
'flu'	6.25%	5.21%	3.7%	4.8%
Number of tweets	3522	2687	65110	19699

Table 8.2: The most occurred words during the start of October 2020 when former US president Donald Trump tested positive for COVID-19. These words are from the discussing subcategory of **Antivax**.

which is further supported by the results of our analysis.

Furthermore, the conspiracy category that achieved the highest percentage of *Promoting/Supporting* subcategory is the **Suppressed cures**, where it got a 0.932%, which adds to a total amount of $3,551,862$ tweets. This category contains tweets based on the narrative that effective medications for COVID-19 were available but whose existence or effectiveness has been denied by authorities, either for financial gain by the vaccine producers or some other harmful harm intent. Figure 8.2 shows the top 50 words for this category, where terms like hydroxychloroquine, zinc, and chloroquine appear. Hydroxychloroquine and chloroquine are medications that have been used for a long time, especially for treating malaria [90]. These chemicals were explored as a possible treatment for COVID-19 early in the pandemic. However, numerous clinical trials and systematic reviews have been conducted to evaluate the effectiveness of hydroxychloroquine for COVID-19. The results from these studies have mainly been inconsistent, and overall, the evidence did not support its routine use for COVID-19 treatment [22]. Zinc is another medication evaluated as a potential cure for COVID-19 in the early stages of COVID-19[91], and is still ongoing research and debate. Still, hydroxychloroquine and zinc were believed to be a cure for COVID-19 by a group of conspiracists [95]. In addition, words like ivermectin have also been observed in the word clouds of Figure 8.2 and 8.3. Ivermectin is an antiparasitic drug that has also been investigated for its potential use in treating COVID-19. Although the role of ivermectin in treating COVID-19 is still under debate, the drug has been widely used in some parts of the world[64]. The interesting founding from the word clouds is that the term "ivermectin" is not present in the training set of COCO dataset, and the model was not trained with examples of this word. But still, the model was able to understand the context of sentences and the meaning of ivermectin due to the similar language in these tweets and could draw a similar

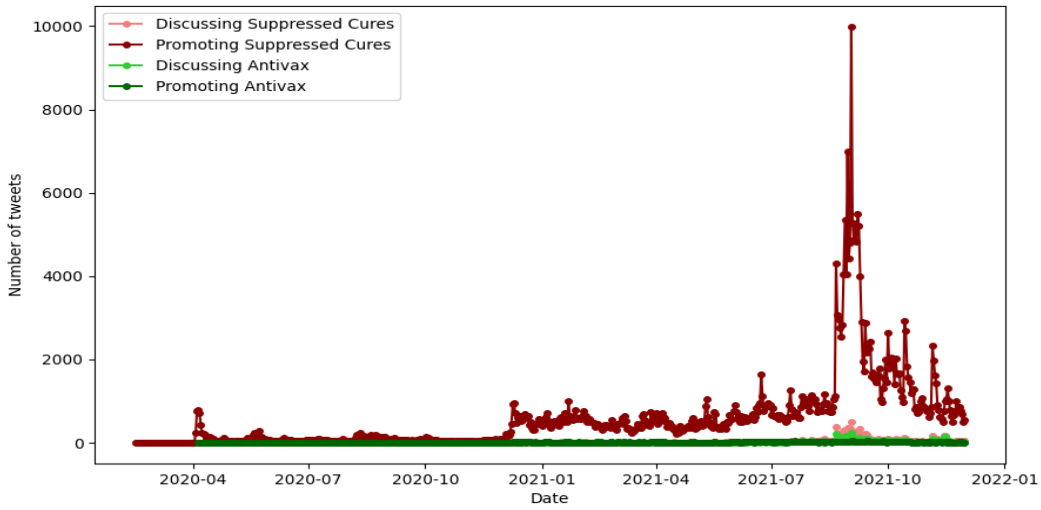


Figure 8.4: The number of tweets in various categories mentioning ivermectin.

understanding as for zinc and hydroxychloroquine. Although hydroxychloroquine became popular among the spreaders during the early stages of the pandemic, ivermectin became famous at the end of 2021, as shown in Figure 8.4. However, this term has been observed in tweets related to **Antivax**; most occurrences were found in the promoting category of **Suppressed cures**. The most occurrence was founded around September 2021, with around 10000.

Over time, there has been a general trend in the number of conspiracy theories, as shown in Figure 8.5. This figure shows the distribution of the nine conspiracy theories from January 2020 to November 2021 for both the discussing and the promoting/supporting subcategories. Based on this figure, the most discussed conspiracy category appears to be the **Fake virus** in the first third of the COVID-19 pandemic. The number of discussing this category exploded around the 2nd of October, marked with a star in Figure 8.5. This is the day when the former President of the USA, Donald Trump, tested positive for COVID-19 (Figure 8.1), which led to a massive discussion regarding the virus on Twitter. The interesting founding from this period, early October 2020, is the number of curtain words used. Table 8.2 displays some of the most used words from tweets classified as discussing **Antivax**, where the word hoax has been used around 80% of the time 2nd October and 3rd October. This word had been used at least **51775** tweets on the 2nd of October.

Pfizer announced that their vaccine candidate against COVID-19 succeeded on the ninth of November 2020[1] (marked as a dot on Figure 8.5), which was the start of an almost continuous discussion regarding the **An-**

[1] https://www.pfizer.com/news/press-release/press-release-detail/pfizer-and-biontech-announce-vaccine-candidate-against

tivax category on Twitter. After this date, the **Antivax** became the most discussed category on Twitter, in contrast to the other categories decreasing around this date. The distribution of the tweets promoting/supporting conspiracies differs from the discussing tweets. The conspiracy that has been most spread is the **Supperessed cures**, where it achieved its highest peak around April 2020. However, it remained low but increased again around September 2021, when ivermectin became popular, as shown in Figure 8.4. Other than **Supperessed cures**, **Intentional Pandemic** had a high distribution around April 2020, but it decreased around June 2020, and the **Fake virus** took over. Towards the end of 2021, **Antivax** became the second most supported conspiracy.

The timeline of discussing and supporting conspiracy categories are compared in Figure 8.6, 8.7 and 8.8. Based on these graphs, the categories of **Suppressed cures**, **Behaviour and Mind Control**, **Intentional Pandemic**, **Harmful Radiation Influence**, **Population reduction Control**, **New World Order** and **Satanism** more promoted. Other categories, such as **Harmful Radiation**, **Fake virus**, and **Antivax**, are among the categories that are likely to be discussed and less spread. However, the most extreme cases are with the categories of **Suppressed cures**, **Population reduction** and **New World Order** since there is a massive gap between the discussing and promoting graphs, which means that Twitter users accept, i.e., believe these categories.

8.4.1 Tweets from before 2020

Out of the 18 million tweets with at least one discussing or supporting conspiracy category, 1106 was founded to be tweets from before 2020, as shown in Table 8.4. These tweets were initially shared on Twitter before the onset of the pandemic. However, they gained visibility and were retweeted by users during the pandemic. To give an overview of the topics in these tweets, we have listed the most occurred bigrams and trigrams in Table 8.3. Like the COCO dataset, Bill Gates and New World Order are among the biggest bigrams and trigrams. Bill Gates is typically combined with the narrative that he is sponsoring some bioweapon to depopulate the world through some bioweapon or vaccination. As we know, this specific conspiracy theory later became that Bill Gates sponsored and made the COVID-19 conspiracy theory to depopulate the world [88]. Based on our analysis, these kinds of tweets date back to 2015.

Furthermore, within these tweets, there is a notable presence of mentions about Robert F. Kennedy Jr. and autism. Upon analyzing the content of these tweets, it becomes evident that the prevailing narrative revolves around the assertion that vaccination causes autism among children. According to these claims, vaccines allegedly contain heavy metals and toxins contributing to an "epidemic of autism" [21]. Robert F. Kennedy Jr., an American environmental activist, and author, has been vocal about his concerns regarding vaccines and their purported association with autism.

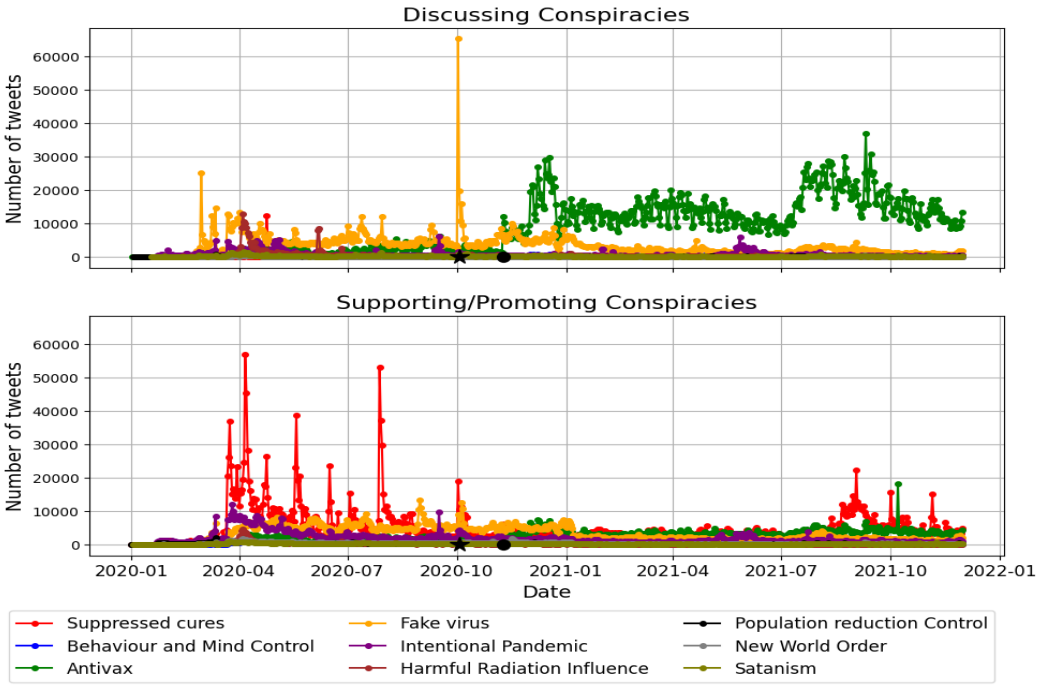


Figure 8.5: The distribution of various conspiracy theories on Twitter during the COVID-19 pandemic is depicted in the graphs, which show how these theories evolved over time. The plot was generated by analyzing one-day intervals.

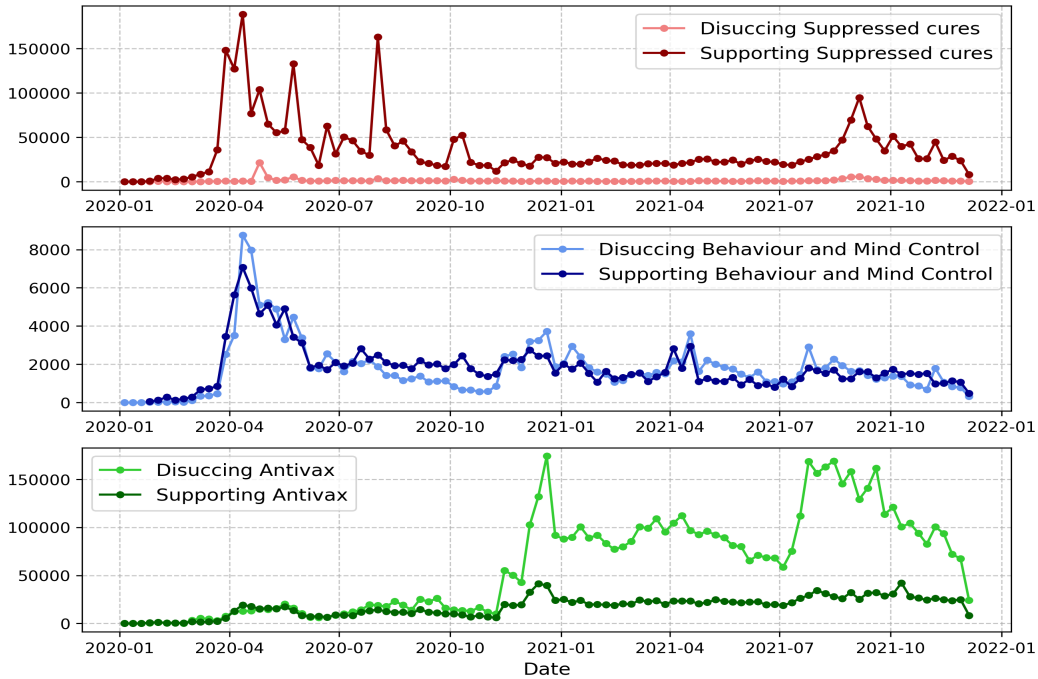


Figure 8.6: Comparison of *Discussing* and *Supporting/Promoting* subcategories of **Supperessed cures**, **Behaviour and Mind Control** and **Antivax** categories. These plots were made at one-week intervals.

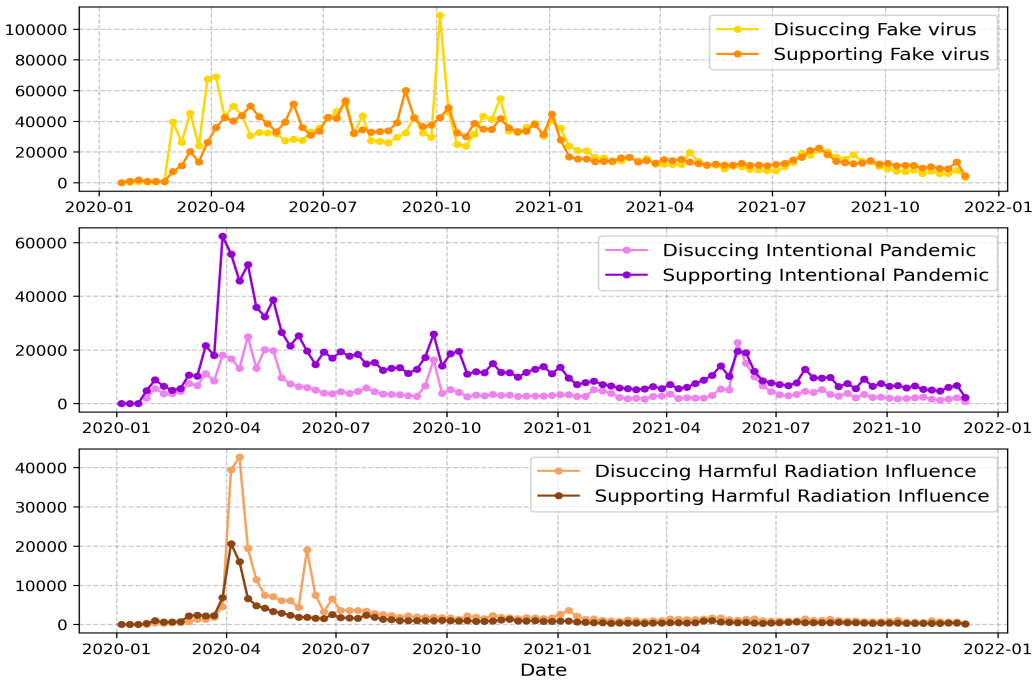


Figure 8.7: Comparison of *Discussing* and *Supporting/Promoting* subcategories of **Fake virus**, **Intentional Pandemic** and **Harmful Radiation Influence** categories. These plots were made at one-week intervals.

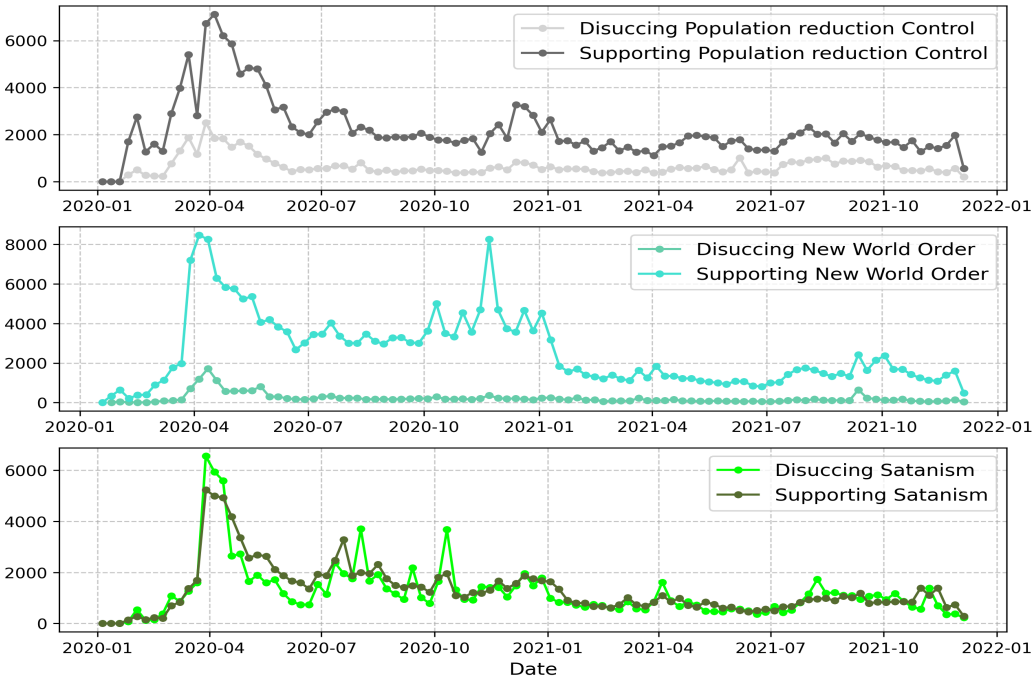


Figure 8.8: Comparison of *Discussing* and *Supporting/Promoting* subcategories of **Population reduction Control**, **New World Order** and **Satanism** categories. These plots were made at one-week intervals.

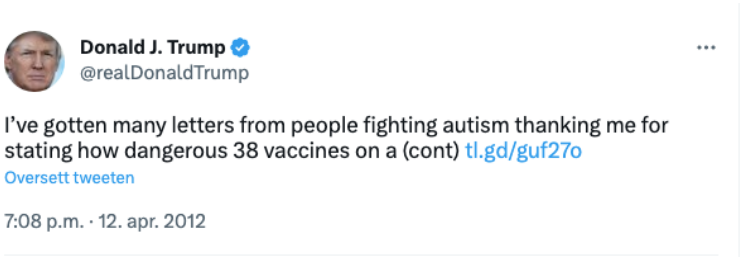


Figure 8.9: Donald Trump in 2012 tweeting about people thanking him for stating how dangerous multiple vaccines on children can be. This tweet was classified as discussing **Antivax** by our model.

Notably, the earliest tweet within this context can be traced back to 2012 and was authored by Donald Trump (see Figure 8.9). Most of these tweets fall under discussing or supporting the **Antivax** stance. Although these tweets are not directly linked to COVID-19 conspiracy theories, our model captures their relevance. In the COCO dataset, four samples establish a connection between autism and the **Antivax** category.

The tweets around 2009 mainly concern the 2009 swine flu pandemic[2]. In these tweets, COVID-19 has been swiped out with different synonyms of the swine flu and H1N1 virus. In addition, some tweets consider the swine flu as false, which falls under the **Fake virus** category. There are also some examples of the **Antivax** and **Suppressed cures** in these tweets. Some tweets attribute the same categories to ebola, stating that ebola is a fake virus.

((a))

Bi-grams	
bill gates	32
autism epidemic	27
new world	19
world order	19
population control	17
vaccine injury	14
climate change	14
child trafficking	13
sex trafficking	12
child sex	12

((b))

Tri-grams	
new world order	16
robert f kennedy	7
need new plague	7
end autism epidemic	7
vaccine injury epidemic	6
f kennedy jr	5
vaccines cause autism	5
climate change hoax	5
child sex trafficking	5
swine flu vaccine	4

Table 8.3: Top 10 most common bigrams and trigrams in the tweets before 2020. These sequences have been generated after the removal of stopwords.

Table 8.3 highlights terms such as "children," "sex," and "trafficking." Tweets containing these words are typically classified under the **Satanism** category. However, a closer examination of these tweets reveals a familiar

[2]https://www.who.int/emergencies/situations/influenza-a-(h1n1)-outbreak

Year	Number of tweets
2006	1
2009	7
2010	10
2011	3
2012	11
2013	9
2014	32
2015	49
2016	55
2017	75
2018	203
2019	651
Total	1106

Table 8.4: The number of tweets from years before the COVID-19 pandemic that was predicted as either discussing or promoting conspiracy theories.

narrative: children are trafficked, sold to pedophiles, and used in satanic sacrificial rituals. The earliest tweets of this nature can be traced back to 2017. Additionally, the climate change conspiracy is among this dataset's most frequently occurring words, suggesting the climate change crisis is a hoax and fabricated. Interestingly, most of these tweets are categorized under the **Fake virus** category, which implies that our model perceives climate change as COVID-19 and classifies it under this category due to the similar language used in these tweets.

8.5 Conclusion

In this chapter, we presented the big data, which consisted of 2.5 billion tweets. After preprocessing, cleaning, and removing the retweets from the big data, the number of unique tweets ended at around 381 million. We performed an inference on this dataset with Graphcore IPUs, which lasted around nine days. Around 18 million tweets were founded to discuss or promote the nine conspiracy categories. The most discussed category is the **Antivax**, and the most promoted or supported category were **Suppressed cures**. One of the exciting founding of the inference was the term *ivermectin*, a drug promoted to cure COVID-19. Although the model was not trained on samples with ivermectin, it could understand it. Finally, we also provided the timeline of each conspiracy category. We also saw how former US president Donald Trump testing positive for COVID-19 affected Twitter and the conspiracy categories.

Chapter 9

Conclusion

In this chapter, we outline the conclusions drawn from our primary findings and address the problem statement outlined in Section 1.2, which forms the basis for the critical contributions of this thesis. Additionally, we share our perspectives on potential future research directions that can build upon the work presented in this thesis.

9.1 Summary

In this thesis, we present several language-based models for COVID-19-related conspiracy detection. We started with evaluating text vectorization techniques to capture a baseline for COCO dataset, such as BOW and TF-IDF. Our study of these methods emphasizes the significance of choosing suitable N-gram ranges and vectorization techniques. We found that the SGD Classifier with TF-IDF outperformed other approaches, particularly when using expanded N-gram ranges. However, acknowledging individual conspiracy categories' distinctive challenges and characteristics is crucial for optimizing detection performance across various theories.

Upon further experimentation, we explored Transformers-based approaches, employing pre-trained BERT- and RoBERTa-based models with One-for-All and One-for-One methods. Our results indicated that the CT-BERT model, which has large domain-specific training data, performed best in the One-for-All approach, achieving an MCC score of **0.765**. However, some categories, such as **Fake virus** and **Intentional pandemic**, proved challenging to predict. Lastly, we explored various techniques to enhance the performance of One-for-All, including ensemble learning and data augmentation techniques. While the ensemble of CT-BERT models achieved a slightly higher MCC score of **0.774**, the improvement was marginal, and the method introduced computational costs and complexity. Data augmentation using EDA and ChatGPT yielded mixed results, with slight improvements in certain specifics but decreased overall MCC score. Experiments with fine-grained datasets, conspiracy recognition tasks, and misinformation detection did not yield substantial improvements. Careful evaluation of the benefits and drawbacks of these techniques is crucial for

optimizing performance in conspiracy detection tasks.

To automate the detection of conspiracy theories on the large dataset from Twitter, we introduced Graphcore's Intelligence Processing Units (IPUs). We gave an overview of how to run deep learning models on IPUs and conducted several experiments to recreate our results from the other experiments. Furthermore, we also carried out experiments regarding the speed of training and inference and compared IPU-POD4 with multiple NVIDIA A100 GPUs. IPU-POD4 is a system comprising four interconnected Graphcore IPUs. The experiments demonstrated that IPUs are slower at training such models but outperform GPUs, mainly when performing inference over large datasets. This investigation highlights the potential advantages of leveraging IPUs for deep learning tasks in terms of scalability. Note that a single IPU has 150 *W* thermal design power (TDP), approximately half of a competitive GPU. Thus, concerning power, each IPU pair is comparable to one powerful GPU, such as the NVIDIA A100. Despite that, the TDP, when we compared IPU-POD4 with four NVIDIA A100 GPUs, the IPU-POD4 was founded to be around three times faster than the GPUs (Table 7.3).

Building upon the results from experimentation with IPUs, we performed the inference over big data, which consisted of around 2.5 billion tweets. However, once the tweets were preprocessed and the duplicates and retweets were removed, the number of unique tweets was around 381 million. The inference was initialized with the IPU-POD4, resulting in around 35 minutes of prediction time for each file. The whole time inference of the big data took around 223 hours, i.e., nine days and nights. Among the 381 million tweets, around $18,627,660$ (4.89%) tweets were assigned with at least one *Discusses Conspiracy* or *Promoting/Supporting Conspiracy* subcategory.

9.2 Research questions

We revisit the research questions in Section 1.2 by repeating and then answering them.

RQ1 *Which machine-learning model is best suited for creating a multi-class multi-label conspiracy theory detection based on the COCO dataset?*

Chapter 5 mainly deals with this research question, where we introduced One-for-All and One-for-One approaches. Both of these approaches have a BERT-based model in their backbone. Based on our experiments, the One-for-All, a multi-class multi-label classifier, was the best performer on our dataset. We experimented with different pre-trained models, such as BERT-Base, RoBERTa-Large, BERTweet, and CT-BERT, where the last completely outperformed the others thanks to its large domain-specific training data, with an MCC score of **0.765**. Chapter 6 contains more experimentation on

this approach to enhance the performance. Most of it failed except the deep ensemble methods, which achieved an MCC score of **0.774**.

1a) *Some conspiracy categories are known to be more keyword-based than others in the COCO dataset. How big is the gap between the text vectorization methods like the TF-IDF approaches compared to the large pre-trained models for these categories?*

Table 3.1 shows the top 10 most common bigrams and trigrams in the training data of COCO dataset, where the most occurred bi-grams and tri-grams are mentioning terms that clearly describe **Population Reduction** and **News World Order** categories from COCO dataset. Chapter 4 mainly deals with this sub-question, where we performed BoW and TF-IDF approach together with various N-gram ranges. Our findings indicated that combining unigrams, bigrams, and trigrams yields the best results, as shown in Figure 4.2. Comparing Figure 4.3 and 5.4, it is clear that Transformer-based outperforms TF-IDF for almost all of the categories. Still, however, for the **New World Order** category, the MCC from both approaches come close, where the best TF-IDF approach gets 0.767. However, for the category of **Population Reduction**, the Transformer-based approaches outperform the TF-IDF approaches, despite this category having the most occurred bigrams and trigrams.

RQ2 *How does the performance of IPUs compare to other hardware solutions, such as GPUs, when conducting large-scale inference tasks for conspiracy theory detection using large pre-trained models?*

Chapter 7 mainly deals with this research question. This chapter describes the experiment by comparing the speed time when conducting large-scale inference tasks. We compared IPU-POD4, a system consisting of four IPUs, with one, two, and four NVIDIA A100 GPUs. Figure 7.3 displays the experiment result. The main finding was that single and dual GPUs are faster at smaller inferences, but for significant inferences, typically in millions of tweets, the IPUs turned out to be much fast. Our experiments show that the IPUs were around 11, 7, and 3 times faster for half to one million tweets than single GPU, dual GPUs, and four GPUs, respectively. The superior speed of the IPUs in handling inference tasks allowedtweetsto successfully execute the large-scale inference process, which required approximately 223 hours to complete. Comparatively, conducting the exact inference with a single GPU would have taken approximately 2453 hours. Employing two GPUs would have reduced the time to around 1561 hours, and utilizing four GPUs would have further decreased it to approximately 669 hours.

2a) *What is the distribution of conspiracy theories on big data? How do the different conspiracy categories evolve through time?*

Chapter 8 describes the result of the inference over big data. Among the 381 million unique tweets, at least 4.89 percent (18627660) discussed or

promoted/supported conspiracy theories through their tweets. The most discussed conspiracy topic on Twitter is **Antivax**, accounting for 1.592% (6,068,439) of categorized tweets. On the other hand, the most promoted or supported conspiracy theory was the **Suppressed Cures**, where it got a 0.932%, which adds to a total of 3,551,862 tweets.

9.3 Future work

In this section, we briefly present some directions into future work that can help the field of fake news and conspiracy detection.

9.3.1 Further pre-training of CT-BERT

In this thesis, we experimented with BERT-Base [23], RoBERTa [61], BERTweet [71], Twitter-RoBERTa-Large[7], and CT-BERT [69] on the COCO dataset. Thanks to its large domain-specific training data, the best-performing approach was the CT-BERT. A strong covariance exists between larger domain-specific pre-trained models and their performance on NLP tasks. The larger the models, the better their performance on NLP tasks. Therefore, we believe further pre-training of CT-BERT can help progress future work on COVID-19-related conspiracy theories and achieve better scores. CT-BERT was trained on a corpus of 160 million tweets about the coronavirus. However, in the end, all retweets, duplicates, and close duplicates were removed from the dataset, resulting in a final corpus of 22.5 million. The backbone of CT-BERT is BERT-Large [23], and the domain-specific dataset consists of 1/7th the size of what is used for training the main model. In our case, we worked with around 381 million unique tweets, which can be used to train a much larger domain-specific pre-trained model.

The total time of training CT-BERT was around 120 hours on a TPU v3-8. TPU v3-8 is a Tensor Processing Unit (TPU) accelerator on the Google Cloud Platform (GCP)[1]. It is a third-generation TPU chip with eight cores to accelerate machine learning workloads. The time and text were around 5.33 per hour per million. Based on this time rate, training on our data would take around 2000 hours.

9.3.2 Domain-specific pre-trained models for other languages

As the world's most widely spoken language, English has benefited from significant research and development in natural language processing. Many resources exist for English, such as large language models, sentiment analysis models, Twitter-based models, datasets, etc. We found CT-BERT very helpful regarding fake news and COVID-19 conspiracy detection. This model performed exceptionally well on the COCO dataset, despite the dataset being highly imbalanced. In contrast, there exist limited resources

[1] https://cloud.google.com/tpu/docs/regions-zones

in other languages. Other than English and Mandarin Chinese, languages like Spanish, French, and Hindi are amongst the most spoken languages in the world. According to the author's knowlegde, no Twitter or COVID-19-related pre-trained open-source Transformer exists for these languages. To combat misinformation online, especially on social media platforms, it is essential to have domain-specific models in their language. Therefore, we believe that one of the future works can be based on training new language models based on social media platforms, such as Twitter.

9.3.3 Augmentation and labeling with ChatGPT

In chapter 6, we used ChatGPT to construct samples for the **Fake Virus** and **Intentional Pandemimc** categories to enrich the dataset and enhance the performance of our model. As described in Section 6.2, we provided ChatGPT the description of the categories and asked it to provide samples in three different stages. This process improved our model slightly, as shown in Figure 6.1. However, our approach to making samples with ChatGPT can be seen as naive, but it also opens the doors toward ChatGPT-driven labeling. Therefore, one of the future works could include using ChatGPT to either make data augmentation or label tweets to improve the One-for-All approach for the COCO dataset. This section will present works leveraging ChatGPT services for data annotation and augmentation.

Since the release of GPT-4, several works have been done to use ChatGPT to label data. One of the examples is the paper of Törnberga et al. [96], where it was founded that ChatGPT-4 outperforms experts and crowd workers in annotating political Twitter messages with zero-shot learning. ChatGPT was compared to manual annotation by experts and crowd workers in this work. The annotation was done on tweets from US politicians during the 2020 election, providing a ground truth against which to measure accuracy. The paper finds that ChatGPT-4 has achieved higher accuracy, reliability, and equal or lower bias than human annotators. In addition, we also introduced AugGPT [18] in Section 6.2, where the authors of AugGPT had achieved promising results by leveraging ChatGPT services for the augmentation of text data. Zhu et al. [115] challenged ChatGPT to study whether it was able to produce human-generated labels. Their results highlight that ChatGPT does have the potential to handle data annotation tasks, where ChatGPT obtains an average accuracy of 0.609.

To conclude, based on these works, ChatGPT-4 has the potential to annotate or construct labels for NLP-based models. This capability could significantly enhance the efficiency and accuracy of natural language processing tasks and pave the way for more advanced AI applications in machine translation, sentiment analysis, and especially in the battle of misinformation.

9.3.4 Graph-Based Source Detection

The result of the inference dataset, a large-scale Twitter dataset with nine distinct conspiracy theories, can be used in various research areas. One of the areas could be to convert the dataset into graphs [77] by using the user IDs to detect the source of the conspiracy theories. The graph representation could be an undirected graph, where the vertices are users and the edges represent connections between them. Each vertex could have a set of attributes, including location, number of followers, etc. Such a classification problem would identify the misinformation spreader based on the user's network. The most significant advantage of such an approach would be that one can develop robust models since the dataset is enormous, which can help the battle against misinformation and potentially be used to identify the spreaders in other types of misinformation, e.g., the 2024 US presidential election. The 2016 US presidential election was imprinted with considerable misinformation on Twitter; using such graph-based conspiracy source detection, one could explore misinformation in the next US presidential election.

9.3.5 In-depth time and evolution analysis of the conspiracies

Accessing the tweets along with their corresponding conspiracy categories and posting timestamps makes it possible to examine the temporal patterns and progression of the associated conspiracy theories. Langguth et al. [53] studied the long-term observation of the COVID-19 and 5G digital wildfire. This work discusses the origin and spread of 5G-COVID misinformation in early 2020. Furthermore, it discusses the rapid growth of the topic from obscurity to widespread discussion and the connection between misinformation and real-world consequences. Our inference results can contribute to this type of study by providing an in-depth analysis of the time and evolution of conspiracy theories, which are often tied to real events and influenced by various factors that play a role in their spread.

In addition to examining the temporal patterns and progression of conspiracy theories, our approach also allows us to investigate the factors contributing to their proliferation. A future study could identify the key drivers and catalysts that drive these theories into mainstream discourse by analyzing the tweets and their associated conspiracy categories. To build upon the work of Langguth et al. [53], we can also explore the role of social media platforms and their algorithms in disseminating conspiracy theories. By understanding how these platforms prioritize and distribute content, we can shed light on the mechanisms that facilitate the rapid growth and spread of misinformation. This insight may, in turn, inform policy and interventions aimed at mitigating the impact of conspiracy theories on public discourse and decision-making.

Moreover, our inference results can be used to assess the influence of external events, such as political developments or public health crises,

on the emergence and persistence of conspiracy theories. By mapping the temporal patterns of these theories against the backdrop of real-world events, we can gain a deeper understanding of how and why certain conspiracy narratives take hold and persist in the public consciousness. Finally, our approach can contribute to studying the psychological and socio-cultural factors underpinning the appeal and resonance of conspiracy theories. By examining the content and structure of these narratives and the language and rhetoric employed by their supporters, we can gain insight into the cognitive and emotional drivers that make these theories compelling to specific individuals and communities.

9.3.6 IPUs for other artificial intelligence algorithms

In Chapter 7, we introduced the IPUs hardware and gave an overview of how it can be used for NLP-based models. We conducted experiments with these units and found them efficient for making large-scale inferences compared to the GPUs. Although IPUs are relatively newer processing units, more experimentation is needed to compare to the state-of-art processing units for deep learning models. With the release of large language models, such as ChatGPT, training sizeable deep learning models are more relevant than ever. IPUs can help the research and industry community train and deploy large models faster. Therefore, one possible future work could be to experiment further with these units and compare them with GPUs.

As mentioned in Section 9.3.4, converting the inference dataset into graphs is the one possible future direction for dealing with misinformation spread. A typical way to work with these graphs is *graph neural network* (GNNs)[113]. A combination of GNNs and IPUs can significantly improve the efficiency and performance of graph-based machine-learning tasks. Moe et al. [68] investigated the viability of the IPUs for efficient implementation of Spatio-Temporal graph convolutional networks, and their results showed that IPUs are well suited for this task.

Bibliography

[1] Rabab Alkhalifa et al. *QMUL-SDS at CheckThat! 2020: Determining COVID-19 Tweet Check-Worthiness Using an Enhanced CT-BERT with Numeric Expressions*. 2020. arXiv: 2008.13160 [cs.CL].

[2] Sarah Alqurashi, Ahmad Alhindi, and Eisa Alanazi. *Large Arabic Twitter Dataset on COVID-19*. 2020. arXiv: 2004.04315 [cs.SI].

[3] M. Anand et al. "Deep learning and natural language processing in computation for offensive language detection in online social networks by feature selection and ensemble classification techniques." In: *Theoretical Computer Science* 943 (2023), pp. 203–218. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2022.06.020. URL: https://www.sciencedirect.com/science/article/pii/S0304397522003887.

[4] Azzalini A. Scarpa B. *Data Analysis and Data Mining: An Introduction*. Oxford University Press; Illustrated edition (April 23, 2012), 2012.

[5] Nihel Fatima Baarir and Abdelhamid Djeffal. "Fake news detection using machine learning." In: *2020 2nd International Workshop on Human-Centric Smart Environments for Health and Well-being (IHSH)*. IEEE. 2021, pp. 125–130.

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. 2015. DOI: 10.48550/ARXIV.1511.00561. URL: https://arxiv.org/abs/1511.00561.

[7] Francesco Barbieri et al. "TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification." In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 1644–1650. DOI: 10.18653/v1/2020.findings-emnlp.148. URL: https://aclanthology.org/2020.findings-emnlp.148.

[8] Christos Baziotis, Nikos Pelekis, and Christos Doulkeridis. "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis." In: *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 747–754.

[9] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." In: *Journal of Machine Learning Research* 13.10 (2012), pp. 281–305. URL: http://jmlr.org/papers/v13/bergstra12a.html.

[10] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization." In: *Advances in Neural Information Processing Systems*. Ed. by J. Shawe-Taylor et al. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

[11] Jay L. Devore Kenneth N. Berk. *Modern Mathematical Statistics with Applications*. Spring Link, 2012.

[12] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers." In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[13] A. Bovet and H.A Makse. "Influence of fake news in Twitter during the 2016 US presidential election." In: *Nat Commun 10*. 2019. URL: https://doi.org/10.1038/s41467-018-07761-2.

[14] L. Breiman. "Bagging predictors." In: *Machine Learning* 24 (1994), pp. 123–140.

[15] Rich Caruana. "Multitask Learning: A Knowledge-Based Source of Inductive Bias." In: *International Conference on Machine Learning*. 1993.

[16] Tianqi Chen et al. *Training Deep Nets with Sublinear Memory Cost*. 2016. arXiv: 1604.06174 [cs.LG].

[17] D. Chicco and G. Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation." In: *BMC Genomics* 21.6 (2020). DOI: https://doi.org/10.1186/s12864-019-6413-7. URL: https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7#citeas.

[18] Haixing Dai et al. *AugGPT: Leveraging ChatGPT for Text Data Augmentation*. 2023. arXiv: 2302.13007 [cs.CL].

[19] Wenliang Dai et al. *Kungfupanda at SemEval-2020 Task 12: BERT-Based Multi-Task Learning for Offensive Language Detection*. 2020. arXiv: 2004.13432 [cs.CL].

[20] Sourya Dipta Das, Ayan Basak, and Saikat Dutta. "A heuristic-driven ensemble framework for COVID-19 fake news detection." In: *Combating Online Hostile Posts in Regional Languages during Emergency Situation: First International Workshop, CONSTRAINT 2021, Collocated with AAAI 2021, Virtual Event, February 8, 2021, Revised Selected Papers 1*. Springer. 2021, pp. 164–176.

[21] Michael Davidson. "Vaccination as a cause of autism—myths and controversies." In: *Dialogues in clinical neuroscience* (2022).

[22] Jiawen Deng et al. "Efficacy of chloroquine and hydroxychloroquine for the treatment of hospitalized COVID-19 patients: a meta-analysis." In: *Future virology* 17.2 (2022), pp. 95–118.

[23] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[24] Thomas G. Dietterich. "Ensemble Methods in Machine Learning." In: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15. ISBN: 978-3-540-45014-6.

[25] Mohamed K Elhadad, Kin Fun Li, and Fayez Gebali. "Detecting misleading information on COVID-19." In: *Ieee Access* 8 (2020), pp. 165201–165215.

[26] Birsen Eygi Erdogan, Süreyya Özöğür-Akyüz, and Pınar Karadayı Ataş. "A novel approach for panel data: An ensemble of weighted functional margin SVM models." In: *Information Sciences* 557 (2021), pp. 373–381. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2019.02.045. URL: https://www.sciencedirect.com/science/article/pii/S0020025519301549.

[27] Jaouhar Fattahi and Mohamed Mejri. "SpaML: a Bimodal Ensemble Learning Spam Detector based on NLP Techniques." In: *2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP)*. 2021, pp. 107–112. DOI: 10.1109/CSP51677.2021.9357595.

[28] Steven Y. Feng et al. *A Survey of Data Augmentation Approaches for NLP*. 2021. arXiv: 2105.03075 [cs.CL].

[29] G.D. Forney. "The viterbi algorithm." In: *Proceedings of the IEEE* 61.3 (1973), pp. 268–278. DOI: 10.1109/PROC.1973.9030.

[30] Yoav Freund and Robert E. Schapire. "Experiments with a New Boosting Algorithm." In: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. ICML'96. Bari, Italy: Morgan Kaufmann Publishers Inc., 1996, pp. 148–156. ISBN: 1558604197.

[31] Anna Glazkova, Maksim Glazkov, and Timofey Trifonov. "g2tmn at Constraint@AAAI2021: Exploiting CT-BERT and Ensembling Learning for COVID-19 Fake News Detection." In: *Combating Online Hostile Posts in Regional Languages during Emergency Situation*. Springer International Publishing, 2021, pp. 116–127. DOI: 10.1007/978-3-030-73696-5_12. URL: https://doi.org/10.1007%2F978-3-030-73696-5_12.

[32] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

[33] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[34] Yoav Goldberg. *Neural Network Methods for Natural Language Processing*. Vol. 37. Synthesis Lectures on Human Language Technologies. San Rafael, CA: Morgan & Claypool, 2017. ISBN: 978-1-62705-298-6. DOI: 10.2200/S00762ED1V01Y201703HLT037.

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[36] J. Gorodkin. "Comparing two K-category assignments by a K-category correlation coefficient." In: *Computational Biology and Chemistry* 28.5 (2004), pp. 367–374. ISSN: 1476-9271. DOI: https://doi.org/10.1016/j.compbiolchem.2004.09.006. URL: https://www.sciencedirect.com/science/article/pii/S1476927104000799.

[37] Nir Grinberg et al. "Fake news on Twitter during the 2016 US presidential election." In: *Science* 363.6425 (2019), pp. 374–378.

[38] L.K. Hansen and P. Salamon. "Neural network ensembles." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (1990), pp. 993–1001. DOI: 10.1109/34.58871.

[39] Fatima Haouari et al. *ArCOV-19: The First Arabic COVID-19 Twitter Dataset with Propagation Networks*. 2021. arXiv: 2004.05861 [cs.CL].

[40] Moritz Hardt, Benjamin Recht, and Yoram Singer. *Train faster, generalize better: Stability of stochastic gradient descent*. 2015. DOI: 10.48550/ARXIV.1509.01240. URL: https://arxiv.org/abs/1509.01240.

[41] Charles R. Harris et al. "Array programming with NumPy." In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[42] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[43] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[44] Charles H Hennekens et al. "Updates on Hydroxychloroquine in Prevention and Treatment of COVID-19." In: *The American Journal of Medicine* 135.1 (2022), pp. 7–9.

[45] Geoffrey E. Hinton et al. *Improving neural networks by preventing co-adaptation of feature detectors*. 2012. DOI: 10.48550/ARXIV.1207.0580. URL: https://arxiv.org/abs/1207.0580.

[46] Matthew Honnibal and Ines Montani. "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing." To appear. 2017.

[47] Robert A. Jacobs. "Increased rates of convergence through learning rate adaptation." In: *Neural Networks* 1.4 (1988), pp. 295–307. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90003-2. URL: https://www.sciencedirect.com/science/article/pii/0893608088900032.

[48] Dan Jurafsky and James H. Martin. *Speech and Language Processing (3rd ed. draft)*. DOI: https://web.stanford.edu/~jurafsky/slp3/.

[49] Debanjana Kar et al. *No Rumours Please! A Multi-Indic-Lingual Approach for COVID Fake-Tweet Detection*. 2020. arXiv: 2010.06906 [cs.CL].

[50] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[51] Robert Kleinberg, Yuanzhi Li, and Yang Yuan. *An Alternative View: When Does SGD Escape Local Minima?* 2018. DOI: 10.48550/ARXIV.1802.06175. URL: https://arxiv.org/abs/1802.06175.

[52] Johannes Langguth et al. "COCO: an annotated Twitter dataset of COVID-19 conspiracy theories." In: *Journal of Computational Social Science* (2023), pp. 1–42.

[53] Johannes Langguth et al. "COVID-19 and 5G conspiracy theories: long term observation of a digital wildfire." In: *International Journal of Data Science and Analytics* 15.3 (2023), pp. 329–346.

[54] Nayeon Lee et al. *On Unifying Misinformation Detection*. 2021. DOI: 10.48550/ARXIV.2104.05243. URL: https://arxiv.org/abs/2104.05243.

[55] Ranran Li et al. "Internet financial fraud detection based on graph learning." In: *Ieee Transactions on Computational Social Systems* (2022).

[56] Shen Li et al. *PyTorch Distributed: Experiences on Accelerating Data Parallel Training*. 2020. arXiv: 2006.15704 [cs.DC].

[57] Qing Liao et al. "An Integrated Multi-Task Model for Fake News Detection." In: *IEEE Transactions on Knowledge and Data Engineering* 34.11 (2022), pp. 5154–5165. DOI: 10.1109/TKDE.2021.3054993.

[58] Jindřich Libovický, Rudolf Rosa, and Alexander Fraser. *How Language-Neutral is Multilingual BERT?* 2019. arXiv: 1911.03310 [cs.CL].

[59] Hung Yeh Lin and Teng-Sheng Moh. "Sentiment Analysis on COVID Tweets Using COVID-Twitter-BERT with Auxiliary Sentence Approach." In: *Proceedings of the 2021 ACM Southeast Conference*. ACM SE '21. Virtual Event, USA: Association for Computing Machinery, 2021, pp. 234–238. ISBN: 9781450380683. DOI: 10.1145/3409334.3452074. URL: https://doi.org/10.1145/3409334.3452074.

[60] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2013. DOI: 10.48550/ARXIV.1312.4400. URL: https://arxiv.org/abs/1312.4400.

[61] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. DOI: 10.48550/ARXIV.1907.11692. URL: https://arxiv.org/abs/1907.11692.

[62] Amr Magdy and Nayer Wanas. "Web-Based Statistical Fact Checking of Textual Documents." In: *Proceedings of the 2nd International Workshop on Search and Mining User-Generated Contents*. SMUC '10. Toronto, ON, Canada: Association for Computing Machinery, 2010, pp. 103–110. ISBN: 9781450303866. DOI: 10.1145/1871985.1872002. URL: https://doi.org/10.1145/1871985.1872002.

[63] Xuting Mao et al. "Financial fraud detection using the related-party transaction knowledge graph." In: *Procedia Computer Science* 199 (2022), pp. 733–740.

[64] Milena Soriano Marcolino et al. "Systematic review and meta-analysis of ivermectin for treatment of COVID-19: evidence beyond the hype." In: *BMC Infectious Diseases* 22.1 (2022), pp. 1–25.

[65] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC, 2014. DOI: https://doi.org/10.1201/b17476.

[66] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[67] Wes McKinney. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[68] Johannes Moe et al. "Implementating spatio-temporal graph convolutional networks on Graphcore IPUs." In: *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE. 2022, pp. 45–54.

[69] Martin Müller, Marcel Salathé, and Per E Kummervold. *COVID-Twitter-BERT: A Natural Language Processing Model to Analyse COVID-19 Content on Twitter*. 2020. arXiv: 2005.07503 [cs.CL].

[70] Preslav Nakov et al. *SemEval-2016 Task 4: Sentiment Analysis in Twitter*. 2019. arXiv: 1912.01973 [cs.CL].

[71] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. "BERTweet: A pre-trained language model for English Tweets." In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 9–14. DOI: 10.18653/v1/2020.emnlp-demos.2. URL: https://aclanthology.org/2020.emnlp-demos.2.

[72] Linh The Nguyen and Dat Quoc Nguyen. *PhoNLP: A joint multi-task learning model for Vietnamese part-of-speech tagging, named entity recognition and dependency parsing*. 2021. DOI: 10.48550/ARXIV.2101.01476. URL: https://arxiv.org/abs/2101.01476.

[73] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[74] Parth Patwa et al. "Fighting an Infodemic: COVID-19 Fake News Dataset." In: *Combating Online Hostile Posts in Regional Languages during Emergency Situation*. Springer International Publishing, 2021, pp. 21–29. DOI: 10.1007/978-3-030-73696-5_3. URL: https://doi.org/10.1007%2F978-3-030-73696-5_3.

[75] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[76] Karishnu Poddar, KS Umadevi, et al. "Comparison of various machine learning models for accurate detection of fake news." In: *2019 Innovations in Power and Advanced Computing Technologies (i-PACT)*. Vol. 1. IEEE. 2019, pp. 1–5.

[77] Konstantin Pogorelov et al. "Combining tweets and connections graph for fakenews detection at mediaeval 2022." In: *Multimedia Benchmark Workshop*. 2022.

[78] Konstantin Pogorelov et al. "FakeNews: Corona Virus and 5G Conspiracy Task at MediaEval 2020." In: *MediaEval*. 2020.

[79] Konstantin Pogorelov et al. "Wico text: a labeled dataset of conspiracy theory and 5g-corona misinformation tweets." In: *Proceedings of the 2021 Workshop on Open Challenges in Online Social Networks*. 2021, pp. 21–25.

[80] B.T. Polyak. "Some methods of speeding up the convergence of iteration methods." In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(64)90137-5. URL: https://www.sciencedirect.com/science/article/pii/0041555364901375.

[81] P. Przybyla. "Capturing the Style of Fake News." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2020, pp. 490–497. URL: https://doi.org/10.1609/aaai.v34i01.5386.

[82] Alec Radford et al. *Improving language understanding by generative pre-training*. 2018. DOI: https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf.

[83] Mabrook S Al-Rakhami and Atif M Al-Amri. "Lies kill, facts save: Detecting COVID-19 misinformation in Twitter." In: *Ieee Access* 8 (2020), pp. 155961–155970.

[84] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. DOI: 10.48550/ARXIV.1609.04747. URL: https://arxiv.org/abs/1609.04747.

[85] Adam Satariano and Davey Alba. "Burning Cell Towers, Out of Baseless Fear They Spread the Virus." In: (2020). URL: https://www.nytimes.com/2020/04/10/technology/coronavirus-5g-uk.html.

[86] Daniel Thilo Schroeder, Konstantin Pogorelov, and Johannes Langguth. "FACT: a Framework for Analysis and Capture of Twitter Graphs." In: IEEE, 2019, pp. 134–141. DOI: 10.1109/SNAMS.2019.8931870.

[87] Gautam Kishore Shahi and Durgesh Nandini. *FakeCovid- A Multilingual Cross-domain Fact Check News Dataset for COVID-19*. ICWSM, June 2020. DOI: 10.36190/2020.14. URL: https://doi.org/10.36190/2020.14.

[88] Shadi Shahsavari et al. "Conspiracy in the time of corona: automatic detection of emerging COVID-19 conspiracy theories in social media and the news." In: *Journal of computational social science* 3.2 (2020), pp. 279–317.

[89] Alex Sherstinsky. "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network." In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306.

[90] Neeraj Sinha and Galit Balayla. "Hydroxychloroquine and COVID-19." In: *Postgraduate medical journal* 96.1139 (2020), pp. 550–555.

[91] Anatoly V Skalny et al. "Zinc and respiratory tract infections: Perspectives for COVID-19." In: *International journal of molecular medicine* 46.1 (2020), pp. 17–26.

[92] Richard Socher et al. "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank." In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. URL: https://aclanthology.org/D13-1170.

[93] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[94] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 0387772413.

[95] Fabio Silvio Taccone, Maya Hites, and Nicolas Dauby. "From hydroxychloroquine to ivermectin: how unproven "cures" can go viral." In: *Clinical Microbiology and Infection* 28.4 (2022), pp. 472–474.

[96] Petter Törnberg. *ChatGPT-4 Outperforms Experts and Crowd Workers in Annotating Political Twitter Messages with Zero-Shot Learning*. 2023. arXiv: 2304.06588 [cs.CL].

[97] Khiem Vinh Tran et al. *UIT-HSE at WNUT-2020 Task 2: Exploiting CT-BERT for Identifying COVID-19 Information on the Twitter Social Network*. 2020. arXiv: 2009.02935 [cs.CL].

[98] V. N. Vapnik and A. Ya. Chervonenkis. "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities." In: *Measures of Complexity: Festschrift for Alexey Chervonenkis*. Ed. by Vladimir Vovk, Harris Papadopoulos, and Alexander Gammerman. Cham: Springer International Publishing, 2015, pp. 11–30. ISBN: 978-3-319-21852-6. DOI: 10.1007/978-3-319-21852-6_3. URL: https://doi.org/10.1007/978-3-319-21852-6_3.

[99] Ashish Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[100] Rutvik Vijjali et al. *Two Stage Transformer Model for COVID-19 Fake News Detection and Fact Checking*. 2020. arXiv: 2011.13253 [cs.CL].

[101] Rutvik Vijjali et al. *Two Stage Transformer Model for COVID-19 Fake News Detection and Fact Checking*. 2020. arXiv: 2011.13253 [cs.CL].

[102] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[103] Andreas Vlachos and Sebastian Riedel. "Fact checking: Task definition and dataset construction." In: *Proceedings of the ACL 2014 workshop on language technologies and computational social science*. 2014, pp. 18–22.

[104] Nguyen Vo and Kyumin Lee. *Where Are the Facts? Searching for Fact-checked Information to Alleviate the Spread of Fake News*. 2020. arXiv: 2010.03159 [cs.IR].

[105] Lei Wang et al. "Ponzi scheme detection via oversampling-based Long Short-Term Memory for smart contracts." In: *Knowledge-Based Systems* 228 (2021), p. 107312.

[106] Helena Webb et al. "'Digital Wildfires' a challenge to the governance of social media?" In: *Proceedings of the ACM web science conference*. 2015, pp. 1–2.

[107] Evan Williams, Paul Rodrigues, and Valerie Novak. *Accenture at CheckThat! 2020: If you say so: Post-hoc fact-checking of claims using transformer-based models*. 2020. arXiv: 2009.02431 [cs.CL].

[108] Steven Lloyd Wilson and Charles Wiysonge. "Social media and vaccine hesitancy." In: *BMJ global health* 5.10 (2020), e004206.

[109] Thomas Wolf 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. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[110] Lianwei Wu et al. *Different Absorption from the Same Sharing: Sifted Multi-task Learning for Fake News Detection*. 2019. DOI: 10.48550/ARXIV.1909.01720. URL: https://arxiv.org/abs/1909.01720.

[111] Li Yang and Abdallah Shami. "On hyperparameter optimization of machine learning algorithms: Theory and practice." In: *Neurocomputing* 415 (Nov. 2020), pp. 295–316. DOI: 10.1016/j.neucom.2020.07.061. URL: https://doi.org/10.1016%2Fj.neucom.2020.07.061.

[112] Aston Zhang et al. *Dive into Deep Learning*. 2021. DOI: 10.48550/ARXIV.2106.11342. URL: https://arxiv.org/abs/2106.11342.

[113] Jie Zhou et al. "Graph neural networks: A review of methods and applications." In: *AI open* 1 (2020), pp. 57–81.

[114] Xinyi Zhou and Reza Zafarani. "A Survey of Fake News." In: *ACM Computing Surveys* 53.5 (Sept. 2020), pp. 1–40. DOI: 10.1145/3395046. URL: https://doi.org/10.1145%2F3395046.

[115] Yiming Zhu et al. *Can ChatGPT Reproduce Human-Generated Labels? A Study of Social Computing Tasks*. 2023. arXiv: 2304.10145 [cs.AI].

[116] Yukun Zhu et al. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. DOI: 10.48550/ARXIV.1506.06724. URL: https://arxiv.org/abs/1506.06724.

Appendix A

Source Code

The source code, dependencies, and other related material used in this thesis are publicly available on the following GitHub repository: https://github.com/rohullaa/master-thesis. This includes the *slurm* files used to execute the code on IPUs on ex3, preprocessing steps, code for each experiment setup, and Jupyter Notebook for visualization.

Appendix B

Published Papers

B.1 Evaluating TF-IDF and Transformers-based models for Detecting COVID-19 related Conspiracies

Evaluating TF-IDF and Transformers-based models for Detecting COVID-19 related Conspiracies

Rohullah Akbari[1,*]

[1]*Simula Research Laboratory, Norway*

Abstract

The proliferation of misinformation and conspiracy theories on online social media platforms has become a significant concern for public health and safety. To effectively combat this issue, a new generation of data mining and analysis algorithms is essential for early detection and tracking of these information cascades. In this paper, we employed a multifaceted approach for detecting and identifying conspiracy theories and misinformation spreaders related to the Coronavirus pandemic. Specifically, we utilized Text-Based Detection (Task 1) through a combination of TF-IDF-based and Transformers-based methods, Graph-Based Detection (Task 2) through a graph convolutional network, and alternative Transformers-based methods to improve the results of Task 1. Our efforts have yielded promising results, with our best models achieving an impressive MCC score of 0.705 for Task 1, 0.041 for Task 2, and 0.698 for Task 3.

1. Introduction

The COVID-19 pandemic and the associated lockdown formed the basis for many false news stories and conspiracy myths. Spontaneous and intentional digital FakeNews wildfires over online social media can be as dangerous as natural fires. The *FakeNews* Task at the MediaEval challenge 2022 targeted the detection of misinformation and its spreaders in tweets. More precisely, this task focuses on analyzing tweets, public user properties, and their connections related to Coronavirus conspiracy theories to detect conspiracies and misinformation spreaders. The description of the task and more information about the dataset can be found in [1]. The detection and verification of COVID-19-related misinformation using machine and deep learning techniques have been addressed in a number of papers [2, 3, 4, 5, 6]. An overview of previous work shows that COVID-Twitter-BERT (CT-BERT) is best suited for building the most successful model for COVID-19-related misinformation and conspiracy detection [4, 7].

2. Text-Based Misinformation and Conspiracies Detection

2.1. The TF-IDF approach

In this section, we will create nine distinct TF-IDF models for each of the nine categories. We are interested to see if the TF-IDF technique can outperform the CT-BERT model, and if not, how close it can come. This approach is based on using *TfidfVectorizer* and Stochastic Gradient Descent classifier (SGD) from the *scikit-learn* framework [8]. SGD is a simple but very efficient approach to fit linear classifiers such as linear Support Vector Machines (SVM). SGD does not belong to any particular family of machine learning models; it is only an optimization

MediaEval'22: Multimedia Evaluation Workshop, January 13–15, 2023, Bergen, Norway and Online

*Corresponding author.

†These authors contributed equally.

rohullaa@uio.no (R. Akbari)

technique. Often, an instance of *SGD Classifier* has an equivalent estimator in the Scikit-learn API, potentially using a different optimization technique. For example, logistic regression is produced when **SGDClassifier(loss='log loss')** is used. The TF-IDF approaches in previous works have been only executed with unigrams [7]. This leads to mislaid learning since there could be important information in the bigrams and trigrams. We can see in Table 2 that N-grams such as "bill gate" and "new world order" could be very important for the classification of the conspiracies. Based on this, we have chosen to implement the TF-IDF with various N-grams including unigrams, bigrams, trigrams, and other ranges. In addition to that, we have also chosen to implement the SGD with different loss functions and penalties (see Table 1 for the parameters).

Table 1
Chosen parameters for the TF-IDF approach. Note that SGD with *hinge* loss is equivalent to linear SVM, SGD with *log* loss is equivalent to LogReg, etc.

Name of parameter	Parameter values
Ngrams	(1,1),(1,2), (1,3),(1,4) (2,2),(2,3), (2,4), (3,4)
SGD loss	hinge, log, modified_huber, squared_hinge, perceptron
SGD penalty	L1 , L2, Elastic net and none

Table 2
Top 5 most common bigrams and trigrams in the dataset of Task 1. These sequences have been generated after the removal of stopwords.

N-grams	Occurences
deep state	222
world order	103
new world	102
bill gate	98
population control	78
new world order	101
years ago cbs	13
cbs show 60	13
interview retired cdc	13
qr code system	12

2.2. Transformers-based approaches

The first Transformers approach (*One-for-All*) is based on training one CT-BERT model for classifying all of the conspiracy categories at once (see Figure 1). The CT-BERT is fine-tuned with nine different weighted Cross Entropy loss functions. The weights are computed by taking into account the number of samples in a specific category and dividing it by the numbers of each of the subcategories in that category. The optimizer used in this approach is AdamW [9]. Before feeding the text data into the model, we preprocessed it by converting the emojis into their textual meaning. Furthermore, the training of the model was done with 5-fold Cross validation and the model with the best test MCC score was chosen. The *One-for-One* approach is based on training nine separate CT-BERT models for the nine categories (the approach is shown in Figure 2). In this approach, we are not using any weighted loss function. Other than that, we are applying the same loss function, optimizer, and preprocessing method. The training of the model was done with stratified 5-fold cross-validation and the model with the best MCC score was chosen.

3. Graph-Based Conspiracy Source Detection

For this task, we applied a simple node classification where the nodes are representing the user's label for whether they are a misinformation spreader or not. We created a network for each of the users that had a label. The network consisted of all of the other users that had an

Figure 1:
The One-for-All approach for Task 1.

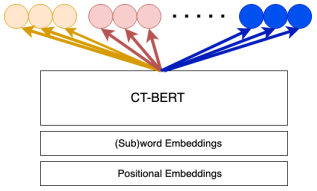


Figure 2:
The One-for-One approach for Task 1.

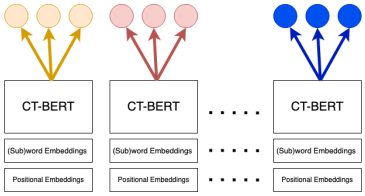
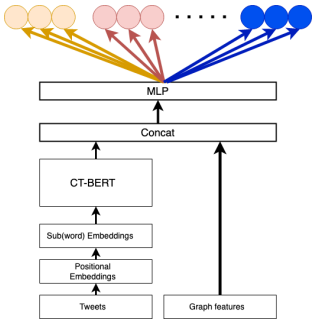


Figure 3:
The CT-BERT-Graph approach for Task 3.



edge directed to the main user and the users with low-weight values were removed. We chose to work with graph convolutional network (GCN) [10]. The implementation was done by using the *GCNConv* class from the *torch_geometric* library with PyTorch.

4. Graph and Text-Based Conspiracy Detection

In this section, we will examine whether we can improve the results from Section 2 by combining the data from Section 2 and Section 3. The output of the classifiers will be enriched by combining text with numerical features. We are proposing an approach that consists of training the CT-BERT with the text data and concatenating the last layer of the CT-BERT with the user information such as **verified_account**, **description_length**, **num_favourites**, **num_followers**, **num_statuses**, **num_friends** and **location_country**. The concatenating layer is then driven through a multilayer perceptron (MLP) and then processed into an output layer (see Figure 3). Our second approach is based on extending the text data with tweeters' statistics and then feeding it into the One-for-All approach 2.2. The numerical features that have been inserted in the text are separated with [SEP] token, e.g.

```
Tweet_text [SEP] 0 [SEP] 159 [SEP] 2812 [SEP] 566
[SEP] 1426 [SEP] 1041 [SEP] 3
```

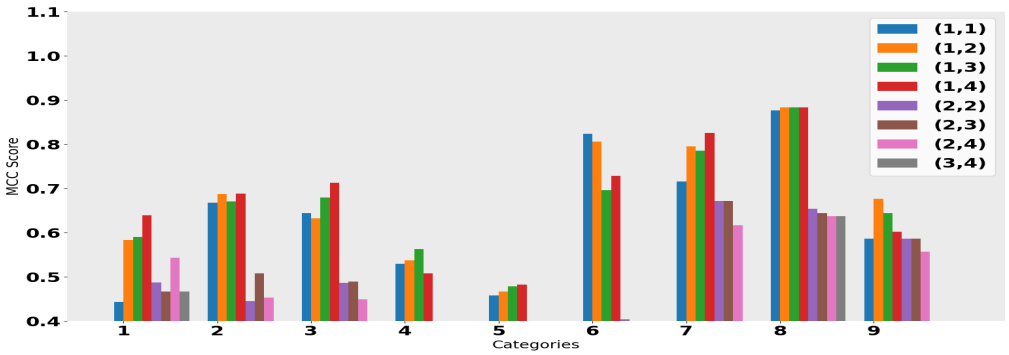
5. Results

As expected, the TF-IDF approach obtained a lower MCC score than the Transformers-based approaches (see Table 3). The One-for-One approach achieved the best score from all submitted runs. The TF-IDF approach does quite well for some of the categories, especially for the **Population reduction** and the **New World Order**. Bigrams such as "population control" and "bill gate" are very important for Population reduction, and "world order" and "new world" are obviously talking about the New World Order category (Table 2). Furthermore, we can see that the N-range such as (2,3), (2,4), and (2,4) did not do well and the dominating range is (1,4) (Figure 4). As a result, unigrams are crucial for the classification of conspiracies since the N-gram ranges without it performed poorly. We submitted only one run for Task 2 which resulted in an MCC score of **0.041** and clearly states that our implementation was not successful. The main reason for the poor performance could be the fact that we removed all the neighbors of the main user node that had low edge values. The combination of CT-BERT with numerical

Table 3
Official MCC scores per category for Task 1 and Task 3. Note that the One-For-All (Task 3) is the same as described in Section 1 but with extended data as described in Section 4.

Category	TF-IDF	One-for-All	One-For-One	One-For-All (Task 3)
Suppressed cures	0.484	0.737	**0.793**	0.563
Behaviour and Mind Control	0.504	0.698	0.700	**0.706**
Antivax	0.529	**0.726**	**0.726**	0.616
Fake Virus	0.378	**0.644**	0.628	0.628
Intentional Pandemic	0.353	0.545	0.592	**0.616**
Harmful Radiation/ Influence	0.617	0.723	**0.729**	0.695
Population reduction	0.710	0.825	0.795	**0.887**
New World Order	0.731	0.778	0.738	**0.850**
Satanism	0.414	0.638	0.663	**0.715**
Average	0.524	0.702	**0.705**	0.698

Figure 4: The plot is showing the performance of the different N-grams ranges. The MCC scores in this plot are from the validation dataset.



features resulted in an MCC score of **0.423**. This approach worsened the predictions, as the test MCC score went below the scores of Table 3. The One-for-All with extended text features achieved an MCC score of **0.698**. None of the approaches in this task improved the outcome of Task 1. However, the One-for-All technique in Task 3, was able to perform better for some of the categories (see Table 3).

6. Discussion and Outlook

We successfully implemented three approaches for Task 1; one TF-IDF approach and two Transformers-based approaches. We experimented with different N-gram ranges and found out that the N-gram range (1,4) was best suited for most of the categories. The best MCC score (**0.705**) was found with the One-for-One approach. We presented two approaches for improving the Task 1 results but none of them improved the results from Task 1.

References

[1] K. Pogorelov, D. T. Schroeder, S. Brenner, J. Moe, A. Maulana1, J. Langguth, Combining tweets and connections graph for fakenews detection at mediaeval 2022, in: roceedings of MediaEval 2022 CEUR Workshop, 2022.

[2] M. S. Al-Rakhami, A. M. Al-Amri, Lies kill, facts save: Detecting covid-19 misinformation in twitter, IEEE Access 8 (2020) 155961–155970. doi:10.1109/ACCESS.2020.3019600.

[3] A. Wani, I. Joshi, S. Khandve, V. Wagh, R. Joshi, Evaluating deep learning approaches for covid19 fake news detection, in: Combating Online Hostile Posts in Regional Languages during Emergency Situation, Springer International Publishing, 2021, pp. 153–163. URL: https://doi.org/10.1007%2F978-3-030-73696-5_15. doi:10.1007/978-3-030-73696-5_15.

[4] A. Glazkova, M. Glazkov, T. Trifonov, g2tmn at constraint@AAAI2021: Exploiting CT-BERT and ensembling learning for COVID-19 fake news detection, in: Combating Online Hostile Posts in Regional Languages during Emergency Situation, Springer International Publishing, 2021, pp. 116–127. URL: https://doi.org/10.1007%2F978-3-030-73696-5_12. doi:10.1007/978-3-030-73696-5_12.

[5] P. Patwa, S. Sharma, S. Pykl, V. Guptha, G. Kumari, M. S. Akhtar, A. Ekbal, A. Das, T. Chakraborty, Fighting an infodemic: COVID-19 fake news dataset, in: Combating Online Hostile Posts in Regional Languages during Emergency Situation, Springer International Publishing, 2021, pp. 21–29. URL: https://doi.org/10.1007%2F978-3-030-73696-5_3. doi:10.1007/978-3-030-73696-5_3.

[6] G. K. Shahi, D. Nandini, FakeCovid- A Multilingual Cross-domain Fact Check News Dataset for COVID-19, ICWSM, 2020. URL: https://doi.org/10.36190/2020.14. doi:10.36190/2020.14.

[7] Y. Peskine, G. Alfarano, H. Ismail, P. Papotti, R. Troncy, Detecting covid-19-related conspiracy theories in tweets (2021). URL: https://2021.multimediaeval.com/paper65.pdf.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, the Journal of machine Learning research 12 (2011) 2825–2830.

[9] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, 2017. URL: https://arxiv.org/abs/1711.05101. doi:10.48550/ARXIV.1711.05101.

[10] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, 2016. URL: https://arxiv.org/abs/1609.02907. doi:10.48550/ARXIV.1609.02907.