Rumour Detection in the Wild: A Browser Extension for Twitter

Andrej Jovanović

4th Year Project Report
Artificial Intelligence and Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

Rumour detection, particularly on social media, has gained significant popularity in recent years. This is due to the disastrous effects rumours have had on society when allowed to propagate virally. As such, the machine learning community has made significant contributions in investigating automatic methods to detect rumours on such platforms. However, these state-of-the-art models are often deployed by social media companies; ordinary end-users cannot leverage the solutions in the literature for their own rumour detection.

To address this issue, this dissertation puts forward a novel browser extension that is capable of performing rumour detection on Twitter. In addition to classifying the rumour status of every tweet, we enhance the user's experience through providing news articles that are semantically related to the tweet. Initial results from a user study confirm that this browser extension provides benefit to users in identifying rumours on Twitter. Additionally, we examine the performance of our browser extension, and the associated rumour detection model, on out-of-distribution (OOD) and imperfect data. Our experiments show that the rumour detection model's state-of-the-art performance decays dramatically if it is evaluated on OOD, or if it is unable to represent the textual content of the tweets in a tweet cascade sufficiently. To this end, additional infrastructure for the browser extension is required to ensure its usability when deployed as a live service for Twitter users at large.

Research Ethics Approval

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: 2023/260884
Date when approval was obtained: 2023-01-30
The participants' information sheet and a consent form are included in the appendix.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Andrej Jovanović*)

Acknowledgements

I would like to thank Dr. Björn Ross, my supervisor, and Dr. Xue Li for their continued support throughout my dissertation. Their encouragement to search for thought-provoking, meaningful and novel research questions has impacted not only the quality of this dissertation, but my future career path. Without their guidance, I would have not considered a career in research as a viable opportunity. To my peers in our dissertation group: thank you for providing much needed comic relief, and a platform to bounce off ideas.

To Senad Ibraimoski, Alexandru-Petre Cazan and Gregor Kerr: thank you for providing me with the space to grow and be curious during my internship at J.P. Morgan Chase. Without the three of you, much of the technical feats of this dissertation would have never been realised.

I want to thank my dear friends who have always been a phone-call away in times of need. I want to thank Alessandro, specifically, for inspiring me academically. It has been an honour to learn alongside you.

To the late Dr. Richard Jenks: thank you for inspiring a young, 16-year-old boy to pursue a career in Computer Science. Who knows where I would be if I had not been enrolled in your class.

To Anka. Thank you for always keeping me level-headed, and for reminding me to laugh and to find enjoyment outside of academics. Life certainly is more entertaining when you step away from the screen.

Finally, I want to thank my parents, without whom I would have never had the opportunity *to complete the exercises left for the reader*. Thank you for your unwavering and unconditional love and support throughout my undergraduate studies.

Table of Contents

Chapter 1

Introduction

1.1 Motivation for misinformation detection on social media

Since the birth of the Internet, our daily interactions were forever transformed. Amongst the trove of benefits, one of the most influential effects is that we are now able to share and consume information at an unprecedented rate. This is particularly visible if we focus on journalism and media at large. Previously, newspaper companies and similar organisations (e.g. universities, governments) were solely responsible for producing news-related content and informing society. Generally speaking [1], one could regard information shared by the aforementioned entities as true. The produced content was vetted through journalistic and/or academic peer-review processes. Furthermore, the organisation publishing said content had a vested interest in sharing "true news/information": their reputation depended on it. This paradigm still exists today. However, with the introduction of the Internet, this blanket assumption no longer holds true.

This is due to the advent of social media, which are defined as *"Internet-based channels that allow users to opportunistically interact and selectively self-present, either in real-time or asynchronously, with both broad and narrow audiences who derive value from user-generated content and the perception of interaction with others."* [7]. These technologies, most notably, have democratised content-sharing [2]. The traditional barriers of participating in the media have been removed. To produce content, one only needs to have an account on a particular social media platform (e.g. Twitter) and an Internet connection. As such, any individual is able to share news of any kind, regardless of the veracity and the intention, on a global platform instantaneously. This democratisation has brought many benefits when related to news-sharing. Particularly, individuals who are involved with a newsworthy event are able to share information as it happens in real-time. This allows news to propagate faster, with the intention of informing the wider public and inciting action from the relevant stakeholders. Furthermore, news-consumers benefit from having news published on social media. Instead of having to read through various articles, they are now able to receive updates of breaking news

[1] We do not attempt to make a comment on news/information that was motivated by a political agenda.

through their social media feeds and via push notifications. This effect is heightened by the fact that news organisations provide users with personalised content suggestions based on personal interests.

However, with this lower barrier of entry to news-content production, the overall quality of information on social media as a whole has degraded [48]. Users of social media could inadvertently share misinformation, or malicious actors could deliberately publish falsehoods [25]. This has led to very damaging consequences for society in the past. Sharma et al. [46] have provided examples of these effects in the financial, political and social domains. When sharing misinformation, we achieve the opposite effect to what was intended with sharing news content on social media: the public is misinformed rather than informed. The condition is further exacerbated by the fact that humans are more likely to share information that is false [52]. Furthermore, researchers have found that users who regularly interact with claims that are unverified are more likely to interact with claims that are intentionally false. [4]. To this end, it is crucial that automatic detection systems are designed to identify misinformation, as humans alone cannot cope with the velocity and volume of information being shared on social media platforms [14].

1.2 Application domain

This work will focus on developing a client-side rumour detection system for breaking news on Twitter. Particularly, this will take the form of a browser extension on Google Chrome. The reasons behind this decision are threefold. Firstly, we focus our attention on Twitter as a specific social media platform. This social media platform, often coined as a microblogging service, has features that allow it to be treated as a news-source in its own right when compared to other social media sites. [35]. This idea is substantiated by a study conducted by Rosentiel et al. [42] that shows that 86% of users in their study use Twitter to keep up-to-date with the news, with 40% of users using the platform to be notified about breaking news. As such, Twitter stands out as the most interesting social media platform for our investigation as it is typically used for the interaction with various news media [8, 20].

Secondly, we focus on the problem of rumour detection in breaking news. The transition away from misinformation detection is natural as rumour detection is a subdiscipline. Referring specifically to breaking news, rumours are particularly pertinent due to the lack of manual verification of claims made in real-time. The information surrounding a breaking news event can be thought of as a rumour due to the fact that during this period, the truth value of the reported content is unknown. It is only *post factum* that the truths and falsehoods are explicitly defined. This idea is illustrated when examining the ”Pizzagate” scandal. The conspiracy theory was only debunked (truth values were assigned to the claims) after the rumour had virally propagated on social media [46, 53]. Additionally, rumour detection is a less-visited area of research from a practical point-of-view, which provides the basis of the work that we propose in this dissertation.

Thirdly, we intend to create a client-side application in the form of a browser extension.

When examining misinformation detection practically, it is often social media companies, who manage the platforms that we refer to in this paper, that implement their own misinformation detection and fact-checking systems [43, 25]. However, the issue with this is that these technologies are server-side [22]. As such, content-consumers do not have access to the service directly. This takes away the autonomy of an individual who wishes to perform rumour detection for themselves, for example. In creating this application, we address this issue, allowing users to have access to an open-source rumour detection system that leverages the advancements made in state-of-the-art research. This addresses the challenges mentioned by Fernandez et al. [14]. Extending the rumour detection capabilities of the extension, it will also provide news articles that are relevant to the tweet in question. News articles are a typical resource that one would use to determine the status of a rumour. This addition enhances the experience for the user, informing them of the context in which the tweet occurs.

1.3 Research Aims

This project will attempt to answer the following three research questions (RQ):

RQ.1: Can we create a browser extension that is able to detect rumours on Twitter and enhance this experience with a related news feed?

RQ.2: Does the browser extension help humans detect rumours more effectively?

RQ.3: How does the rumour detection model, and the browser extension more generally, perform on out-of-distribution (OOD) and imperfect data?

1.4 Contributions

The major contributions of this project are as follows:

1. The full infrastructure for a rumour detection browser extension on Twitter and the deployment of these services using Google Cloud Platform and the Google Chrome store [2].
2. Created the infrastructure surrounding the browser extension in such a way that it can be ported easily to a different application domain.
3. A critical evaluation of the browser extension as tool, in the form of a user study, where users were asked to perform a series of rumour detection tasks on Twitter.
4. Analyse the performance of a state-of-the-art rumour detection model on out-of-distribution and imperfect data, which are pertinent to the successful deployment and functioning of the browser extension.

[2] We make the source code of this entire dissertation publicly available on GitHub: `https://github.com/maddox-j/rumour_detection_in_the_wild`

Chapter 2

Background and Related Work

In this chapter, we outline some definitions that are necessary to understand this dissertation in section 2.1. We provide an overview over the techniques used in the field of rumour detection, from seminal papers to state-of-the-art in section 2.2. Finally, we examine any related browser extensions or web applications that have been put forward thus far in section 2.3. These will serve to motivate the contribution of a novel rumour detection extension for Twitter, and the benefit it would bring to users as it utilises the predictive performance of the state-of-the-art solutions.

2.1 Defining Key Terms

The use of the word *misinformation* has become oversaturated with its rampant use on social media. Consequently, many such related terms used in the literature, and in the news, have been treated as synonyms and used interchangeably when, in fact, they have distinct definitions. It will thus be of great importance to define each of the terms used in this dissertation. We will lean on much of the work outlined in [55] to create a formal understanding for the reader.

As put forward by Wu et al., *misinformation* is normally used an umbrella term to encompass related topics such as: disinformation, fake news and rumours [55]. It has a convenient definition capable of encapsulating any other derived or similar definitions. Formally, and in this work, *misinformation* is known as:

Definition 2.1.1. [Misinformation] Any false or inaccurate information, albeit spread unintentionally [55].

A rumour is a subcategory of misinformation. Formally, a rumour is:

Definition 2.1.2 (Rumour)**.** Any piece of information that propagates whose veracity is unverified at that instance in time [37, 48, 11].

Contrastingly, we can define a non-rumour to be:

Definition 2.1.3 (Non-rumour)**.** Any piece of information that propagates whose veracity is known at that instance in time.

From these definitions, we could think of a non-rumour as a claim that is typically well-known and non-controversial (e.g. *the grass is green.*). Furthermore, we notice that in accordance with the definitions of 2.1.1 and 2.1.2, a rumour could eventually become a true or false claim once there has been enough information to assign it a veracity tag (truth label). Within this context, a rumour could fall into one of three categories:

Definition 2.1.4 (True rumour)**.** A claim, that was initially a rumour, whose information is found to be truthful after fact-checking.

Definition 2.1.5 (False rumour)**.** A claim, that was initially a rumour, whose information is found to be false after fact-checking.

Definition 2.1.6 (Unverified rumour)**.** A rumour whose truth value has not yet been determined.

Since we are considering rumour detection on Twitter in this work, we define rumour detection as follows:

Definition 2.1.7 (Rumour detection)**.** A task whereby we wish to determine whether the claim made by a particular tweet is a rumour or not. If the claim is a rumour, we additionally wish to determine the veracity tag of said tweet.

2.2 Related Work - Rumour Detection

As discussed in section 1, there has been a severe uptake in public interest to tackle misinformation, and more specifically rumour detection. Broadly speaking, there have been two distinct solutions proposed, each of which will be described below.

2.2.1 Manual solutions

Traditionally, the task for solving rumour detection was done by humans. An annotator would analyse each example of a potential rumour, manually assigning it to the relevant class. This paradigm is often seen with many independent rumour-debunking websites such as Snopes[1] and PolitiFact[2]. This is effective (the human annotators are generally quite accurate) and suitable on a relatively small scale. However, there are two major flaws when one extends this solution to a social media website such as Twitter [46] [59].

1. To classify any instance, an individual requires a large amount of time to gather supporting evidence to motivate their decision. This criterion renders manual approaches inutile when tackling the volume and velocity of misinformation on social media [14], particularly Twitter. This issue is especially pertinent when we consider that misinformation spreads much faster, and deeper, than true news [52]. Therefore, proposing a timeous solution is of utmost importance.
2. In addition to the above, individuals have to have, generally, some level of proficiency in the subject domain that they are verifying. This makes it difficult

[1] https://www.snopes.com/
[2] https://www.politifact.com/

for manual rumour-debunking services to recruit a sufficient number of individuals to cope with the scale of misinformation detection.

As such, while manual solutions are effective on the level of individual claims, this solution does not scale to tackle misinformation in social media. As such, automatic rumour detection solutions prove more valuable.

2.2.2 Automatic solutions

Automatic rumour detection systems leverage the use of machine learning (ML) and natural language processing (NLP) to tackle the volume and velocity typically seen with misinformation detection on social media [14]. Models exploit different aspects of the problem in order to create performant classifiers. Examples of these include: extracting informative features based on content, analysing the propagation patterns of misinformation on social networks or modelling the reaction cascade to a particular post. However, these models can be split into two further categories: traditional ML or deep learning.

2.2.2.1 Traditional ML methods

In this section, we will use traditional ML methods to encompass those that involve some sort of feature engineering. This is unlike deep learning methods that are able to learn informative features automatically.

Seminal papers in the domain of rumour detection on Twitter focussed on engineering informative features for the task [37, 8, 57]. These features typically fell into the following categories: i) *content-based* - features based on the rumoured tweet itself, ii) *user-based* - features based on the user who posted the rumour, iii) *topic-based* - aggregate statistics computed based on the previous two feature sets and iv) *propagation-based* - features created based on the propagation graph of the rumour. Certain papers calculated features that were specific to the microblog platform that they were analysing [37, 57]. These features are then fed into traditional, ”simple” classifiers, such as decision trees or log-linear models. These seminal papers motivated the use of automatic rumour detection techniques that were to follow as they achieved encouraging results. Additionally, whilst these approaches were state-of-the-art for their time, deep learning approaches have been shown to be far superior, as we will show in section 2.2.2.2.

Following this work, researchers began to refocus their attention, in this rumour detection task, on temporal features. Kwon et al. showed that that temporal features, ones that can describe the periodic bursts that are typical to rumours, are highly predictive of rumours [24]. With temporal features, models are able to capture how rumours change over time, rather than viewing them as single, static events [31]. Applying these new features to traditional classifiers such as random forests and support vector machines (SVM), the authors showed that their work outperformed the state-of-the-art techniques at the time. This opened a new method of identifying rumours, especially since temporal features are easier to calculate than structural or linguistic features [24].

These features gave rise to a paradigm known as early rumour detection. Detecting rumours early, before they have had a chance to propagate virally, is imperative; this

will mitigate, to an extent, the disastrous consequences outlined in section 1 [23, 31]. As such, rumour detection needs to be conducted without access to the features that are meaningful only once the rumour has circulated for a while [59]. The work of Kwon et al. showed that certain handcrafted features that had been used in the literature up to that point are more informative at different time steps of the rumour's propagation [23]. For example, participants that express doubt towards a particular tweet in the first few days of circulation are an early marker of whether the tweet should be considered a rumour. Ma et al. applied time series modelling to the rumour detection ask to leverage the information gained from the social context of the tweet [31]. This gave rise to another temporal approach which is similar to Kwon et al. [24]. Ma et al. showed that a model using these time-series-based features is able to preform well in the early detection of rumours. Similarly, Wu et al. focused on the temporal nature of the problem by modelling the message propagation pattern as a tree [3] [54]. Their results showed that models that make use of these novel features are able to outperform state-of-the-art baselines, and preform early detection of rumours. Models based on these tree-like structures will be seen later in the deep learning approaches in section 2.2.2.2 due to their predictive performance.

The work of Zubiaga et al. highlighted the transition from traditional ML to deep learning (DL) applied specifically to rumour detection on Twitter [60]. Apart from using hand-crafted content-based and social-based features, their model employed a *word2vec* representation of the words in the tweet they wanted to classify. Zubiaga et al. coupled this approach with a conditional random field algorithm (CRF), a sequential classifier. This can again be thought of interpreting the rumour detection task temporally. This algorithm exploits the sequential nature of a tweet by representing it along with a collection of related tweets that have preceded it [60]. Using a sequential classifier proved to outperform traditional, non-sequential baselines in their paper. This would inspire a move to sequential neural network approaches seen in section 2.2.2.2.

The major collective flaw of these models is the feature engineering: the process is detail-specific, it can introduce biases and is incredibly laborious, as pointed out in [5, 30]. As such, state-of-the-art solutions turn to deep learning for the automatic feature representations, increased model complexity and subsequent increase in performance for rumour detection.

2.2.2.2 Deep learning approaches

One of the first applications of deep learning to rumour detection was seen in Ma et al.'s work with recurrent neural networks (RNNs) [30]. This paper leveraged the fact that text streams on social media, in this case Twitter, can be expressed as a time series [31, 24]. As such, an RNN is able to make use of the temporal signals that are typical of rumour propagation. This is a similar to the sequential approach taken by the traditional ML methods seen in section 2.2.2.1. On a rumour detection dataset for Twitter, their most performant RNN with two layers comprised of gated recurrent units (GRUs) achieved an accuracy of 88.1%. This model outperformed an SVM based on time-series structures which achieved 80.8% accuracy. This shows that DL approaches are much

[3] The paper experimented on Sina Weibo, a Chinese microblogging platform similar to Twitter.

more performant than traditional machine learning approaches that were state-of-the-art at the time.

Building on the success of this paper, Ma et al. proposed a recursive neural network (RvNN) [33]. Unlike RNNs, RvNNs are able to embed both content-based and propagation-based information due to their tree-like structure, as seen in [54]. The authors posited that the propagation-based information is important to rumour detection as the types of responses a tweet receives may be indicative of whether the tweet is a rumour or not. For example, repliers will tend to disagree with a tweet that supports a false claim, or denies a true claim, and *vice versa* [33]. Similar evidence was observed in the work by Kwon et al. [23]. Ma et al. showed that their RvNN model is able to outperform all baselines, including their previously developed RNN with two layers of GRUs [33]. Following the success of the "Attention is all you need" paper [51], Ma et al. reimplemented their RvNN in [33] with an attention mechanism [29]. Instead of treating each post equally in the top-down or bottom-up composition of the tweet, the attention mechanism is able to place greater emphasis on posts that are more informative. Unsurprisingly, the revision found that with the addition of a global attention mechanism, the RvNN is able to outperform all other baselines [29].

Building on this approach, Bian et al. proposed a bidirectional graph convolutional network (BiGCN)[4]. Not only is the BiGCN able to model the propagation properties of a rumour, as a RvNN does [29, 33], it is also able to model the dispersion properties [5]. The authors posited that these two aspects are crucial when attempting to model rumours. Bian et al. showed that their novel model is able to outperform previous state-of-the-art baselines, including those mentioned in this background, with an accuracy of 88.6% and 88.0% on the Twitter15 and Twitter16 datasets respectively. Additionally, this trend was confirmed with results on the early detection task.

2.3 Related Work - Browser Extensions

Gupta et al. were one of the first to create a practical system for tweet credibility (rumour detection) [8, 16]. The paper introduced TweetCred, a browser extension for Google Chrome, that is able to assign a credibility score for each tweet on a user's feed. The credibility score was calculated on an SVM-rank model using 45 handcrafted features. The extension's efficacy was judged on two axes: latency and credibility agreement with users. For latency, the credibility score was provided in less than six seconds for 84% of users, while only 43% of users agree with the credibility score provided [16]. There are two main flaws with this solution. Firstly, the last time that the extension had been maintained was in 2015 [5]. As such, the extension is out-of-date and does not leverage the latest state-of-the-art in rumour detection. Secondly, the extension does not provide relevant articles for each tweet, which would improve the rumour detection experience for the user. Our novel contribution will address both of these issues.

Popular web applications in the realm of rumour detection include those that are

[4]This paper will be further explained in section 3

[5]https://chrome.google.com/webstore/detail/tweetcred/fbokljinlogeihdnkikeeneiankdgikg

able to monitor and visualise the propagation of rumours on social media [45, 34, 40]. These platforms are often built to help researchers understand how rumours propagate on social media, and to help journalists detect which rumours are worthy of reporting. The browser extension we create does not tackle the visualisation of rumour propagation. Our work focuses on providing rumour detection capabilities for users, and not journalists. In this light, we wish to detect rumours on the level of individual tweets, and not at a global event level. Secondly, in the case of the RumourLens application put forward in [40], their rumour detection algorithm is impoverished. The technique they employ involves searching for expressions that are commonly used in controversial claims to identify rumours [40]. Instead, the browser extension in this paper will employ state-of-the-art rumour detection algorithms based on deep learning as explored in section 2.2.2.2.

Thilakarathna et al. created a browser extension for Twitter, called Veritas, that is able to detect fake news on the social media platform [50]. Like our work, the researchers use a centralised server architecture to provide fake news detection for tweets. Furthermore, Thilakarathna et al. used an ensemble method that combined two fake news detection models: one based on tweet content and another based on tweet context [50]. Most notably, their architecture involves a model trainer pipeline to ensure that their neural models are constantly up-to-date. Apart from not focusing on rumour detection, this work has a significant flaw. The machine learning methods they have chosen seem to be arbitrary; they do not use the state-of-the-art models in fake news detection specifically. Our work is an improvement in this regard as we directly leverage work from the rumour detection literature.

The work of Kydd et al. explored a very similar tool to the one we propose [25]. The researchers created a machine-learning-powered browser extension focusing on clickbait detection. Similar to our work, Kydd et al. [25] made use of automatic detection solutions using a DL model. Furthermore, the researchers deployed this service to users via a browser extension that provides warning labels. However, the paper specifically focused on informing users of potential clickbait articles, unlike this dissertation which will focus on rumours. Secondly, the architecture that we use to deliver the rumour detection model's results is different. Kydd et al. [25] made use of a Native Manifest, which would allow a web browser to interface with a local application that housed their predictive model written in TensorFlow. Instead, we make use of a centralised web server, which will be explored further in section 4.3.

2.4 Background for the Browser Extension

2.4.1 Twitter API

The Twitter Developer API [6] is a platform that allows developers to have direct access to Twitter's platform, and associated data. This access is intended to facilitate the creation of new tools, as is the case of our Twitter rumour detection extension. To use the Twitter API, developers need to create a developer account. Then, the developers will create

[6] https://developer.twitter.com/en/docs/platform-overview

a project and app which allows them to generate secret keys and tokens. These will be used to authenticate all requests to the Twitter API server. Particularly, our browser extension makes use of the API endpoints seen below. These endpoints will allow for all the necessary data to be retrieved from the Twitter API to perform rumour detection on unseen tweets successfully.

Tweets lookup returns information regarding a requested tweet ID. This is achieved through specifying the particular ID in the query of the URL [7] seen below:

```
https://api.twitter.com/2/tweets?ids=<id>
```

Search tweets returns the direct replies, and the replies of replies, linked to a given tweet ID. Specifically, these are tweets that are made in the last seven days. The particular ID is specified as the *conversation_id* parameter in the URL. Additionally, we are able to specify optional query parameters. Two parameters that will be of particular importance are *until_id* and *max_results*. The former allows us to specify that we wish to receive tweets that are older than the given *until_id*, whilst the latter, a number between 10 and 100, allows us to specify the number of results to be returned by the server. Furthermore, this API endpoint has a rate limit of 450 requests per 15-minute window [8]. The URL is seen below:

```
https://api.twitter.com/2/tweets/search/recent?query=
    conversation_id:<conversation_id>&until_id=<until_id>&
    max_results=<limit>
```

Quote tweets returns the quote tweets and retweets to a given tweet ID, which is specified by the *tweet_id* parameter. As with the *search tweets* endpoint, we are able to specify additional optimal parameters. Two that will be of particular importance for this dissertation will be *max_results* and *pagination_token*. The former's function is the same as in *search tweets*, whilst the *pagination_token* allows us to move through pages of results indexed by the server. Furthermore, this API endpoint has a rate limit of 75 requests in a 15-minute window[9]. The URL is seen below:

```
https://api.twitter.com/2/tweets/<tweet_id>/quote_tweets?max_results=<
    limit>&pagination_token=<token>
```

2.4.2 NewsAPI

NewsAPI.org provides a REST API service that allows developers to find news articles returned in a JSON format. In order to use this service, an API key needs to be generated such that all requests can be authenticated. Particularly, this dissertation makes use of the *everything* endpoint, filtered according to keywords. This searches for articles, created by over 80,000 sources, made in the last five years [10]. This service will be used to find news articles that are semantically related to the tweet we wish to classify.

[7] Tweet lookup documentation
[8] Tweet replies documentation
[9] Quote tweets documentation.
[10] NewsAPI documentation.

Chapter 3

Application Prerequisites

This chapter will outline some of the key aspects of the rumour detection browser extension, including the dataset on which the model was trained (section 3.1), the model powering the rumour detection (section 3.2) and the pretraining methods used for this model (section 3.3).

3.1 Datasets

The dataset that was used to train the rumour detection model in section 3.2 was the *Twitter16* dataset released in [32]. This is an expanded version of the dataset that was released in [30]. The original dataset, which was framed as a binary classification problem for rumours vs. non-rumours, contained tweets from 778 reported events between March and December 2015. However, the data in its raw form suffered from a class imbalance favouring the rumour class. To mitigate this, Ma et al. augmented their data by adding additional examples of non-rumours taken from [8, 24] to obtain 498 rumours and 494 rumours in total. However, an issue with this data is that it did not contain the propagation cascades for each tweet, which Ma et al. [32] wished to model. As such, to create the Twitter16 dataset, Ma et al. focused only on the popular source tweets from the original dataset. The reason behind this decision is that unpopular tweets, although they could be rumours, are not impactful to a wider audience [32]. Once they had isolated this subset of popular tweets, the researchers created the propagation structure of each tweet by retrieving its retweets, quote tweets and replies. This was repeated recursively for each retrieved tweet. In order to abide by the definition of the rumour task in 2.1.7, Ma et al. needed to map the binary labels that were originally used into a four class problem. To achieve this, Ma et al. cross-referenced the news event mentioned in each tweet with a fact-checking website such as Snopes.com [32]. Ma et al. assigned tweets mentioning an unverified or a non-rumour event the corresponding label. Tweets that referred to a false rumour event were assigned a false label if the tweet did not deny the false rumour. Otherwise, the tweet was assigned a true label. The same procedure was followed for those tweets that referred to a true rumour event [32]. The dataset has the following statistics, as outlined in Table 3.1., with an example of a false rumour in Figure 3.1.

Figure 3.1: Two examples of false rumour tweets from the *Twitter15* (top) and *Twitter16* (bottom) datasets [1].

For evaluating our third RQ in section 6, we make use of an additional dataset, Twitter15. This dataset was based on the dataset released by Liu et al. [27] that contained 421 true and 421 false newsworthy events that occurred up to March 2015. Liu et al. did not state how many tweets were associated with each event. In order to make this dataset consistent with Twitter16, Ma et al. followed the same procedure specified above to obtain the Twitter15 dataset used in their paper [32]. The statistics for this dataset can be seen in Table 3.1, which can be used to approximate the number of tweets in the original [27] dataset. An example of a false rumour is seen in Figure 3.1. We also note that some of the veracity tags that were originally assigned to this dataset may have changed when Ma et al. gathered the additional data need to represent the propagation structure.

Statistic	**Twitter15**	**Twitter16**
# of users	276,663	173,487
# of source tweets	1,490	818
# of threads	331,612	204,820
# of non-rumours	374	205
# of false rumours	370	205
# of true rumours	372	205
# of unverified rumours	374	203
Avg. time length / cascade (Hours)	1,337	848
Avg. # of posts / cascade	223	251
Max # of posts / cascade	1,768	2,765
Min # of posts / cascade	55	81

Table 3.1: Statistics of the *Twitter16* dataset as in [32].

Furthermore, in order for these datasets to be used with the rumour detection model in section 3.2, each dataset has an associated dictionary of the 5000 most frequently occurring tokens, after tokenization. Each token in the dictionary has a corresponding ID number. Additionally, the cascade structure for each tweet, and thus the data, is

[1] Twitter15 tweet, Twitter16 tweet

Listing 3.1: An example tweet cascade taken from the Twitter15 data whose textual feature representations have been shortened for the sake of brevity.

```
723365789378584578 None 1 2 15 485:1 ...
723365789378584578 1 2  2 15 572:1 ...
723365789378584578 1 3  2 15 3445:1 ...
```

encoded in a particular format, as specified in [2], which can be seen in Listing 3.1. Each tweet cascade is contained in a text file titled $\{tweet\_id\}.txt$, where *tweet_id* is the tweet ID of the root tweet in the cascade. Each text file is composed of rows, where each row corresponds to a particular tweet in the tweet cascade. Each tweet is encoded according to the following tab-separated columns:

1. The ID of the root tweet of the cascade to which the current tweet belongs.
2. The non-zero index of the parent tweet of the current tweet, if any.
3. The non-zero index of the current tweet
4. The total number of parents in the tweet cascade.
5. The maximum length of all texts in the cascade.
6. A space-separated representation of the textual content of the specific tweet. This representation is specified by *index:count* pairs. The index of a particular token is obtained from the dictionary mentioned in section 3.1, if it exists, and the count is the number of times that token occurs in the tweet. Each tweet is appended with an $<end>$ token so that every tweet will have an *index:count* pair.

3.2 Rumour detection model

This section will explore the Bi-Directional Graph Convolutional Network (BiGCN) proposed by Bian et al. [5] which was selected as the rumour detection model for this browser extension. The reason for this selection is two-fold. Firstly, according to the paper's reported results, this model achieved state-of-the-art performance when compared to previous models, as mentioned in section 2.2.2.2. Secondly, the code related to this paper was publicly available on GitHub [3], and written in PyTorch, for which there is ample community support.

As mentioned in section 2.2.2.2, the BiGCN models both the propagation and dispersion structures of a rumour [5]. Algorithmically, this means that the model leverages both the top-down and bottom-up views of a rumour's tweet propagation structure (tweet cascade). A tweet cascade is a tree structure that represents a tweet event (a source tweet and all the tweets that interact with it). This is visualised in Figure 3.2. Observing (a), the root node of the cascade is the source tweet. Its child nodes represent the replies and quote tweets that respond directly to this source tweet. Each sub-cascade can be

[2] https://github.com/majingCUHK/Rumor_RvNN
[3] https://github.com/TianBian95/BiGCN

thought of as a new tweet cascade. The tweet cascade bottoms out at the leaf nodes, which are tweets that do not have any replies or quote tweets associated with them. We can represent each tweet, and its associated cascade mathematically. Every tweet is represented by c_i where $c_i = \{r_i, w_1^i, w_2^i ..., w_{n-1}^i\}$, where r_i is the root node (tweet) of the tweet cascade (event) and each w_j^i is a responsive post in the cascade.

In this basic form, the tweet cascade can be thought of as an undirected graph. This particular structure can be recreated from $\{r_i, w_1^i, w_2^i ..., w_{n-1}^i\}$ by not imposing a direction on the edges in the graphical structure. When creating the top-down structure, which models the propagation structure, we impose an ordering on the edges to create a directed graph. Each edge would be directed from the source tweet to its response tweet, as seen in (b) in Figure 3.2. Mathematically, we define this graphical structure as G_i for each i^{th} tweet. As such, Bian et al. define each tweet c_i to be $c_i = \{r_i, w_1^i, w_2^i ..., w_{n-1}^i, G_i\}$ [5]. For the bottom-up representation, which models the dispersion structure, the directionality is reversed; each edge points from the response tweet to the source tweet [5] as seen in (c) in Figure 3.2. The BiGCN model aims to solve the rumour detection task. Specifically, the model attempts to learn a function f, such that $f : c_i \rightarrow y_i$, where y_i is the label associated to the tweet cascade [5], for all events in the training dataset 3.1.

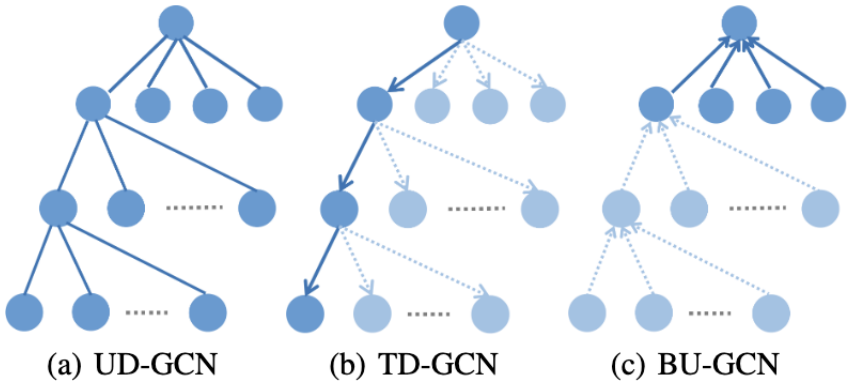


Figure 3.2: A graphical view of a tweet cascade (a), it's top-down (b) and bottom-up (c) representations taken from [5].

The BiGCN model creates this function f by leveraging the top-down representation received from G_i, and converts it to an adjacency matrix A_i. Each row of this matrix corresponds to a tweet in the graph, with the first row being the root tweet. To this, Bian et al. apply DropEdge [41, 5] to produce the adjacency matrix A_i'. This acts as a noising step to the input to prevent overfitting. After DropEdge has been applied, the representation for the bottom-up adjacency matrix is the transpose of A_i', which reverses the directionality of the graph. This corresponds to step one in Figure 3.3. Furthermore, the model receives a feature matrix X_i for each tweet cascade. This matrix represents the features (index:count pairs) for each tweet in the cascade. The first row of this feature matrix, $\mathbf{x}_0^i$, corresponds to the feature vector of the root tweet. Each $\mathbf{x}_j^i$ is then the feature vector of the j^{th} responsive tweet.

These top-down and bottom-up adjacency matrices can be viewed as the inputs to the model along with the tweet-feature matrix. These input adjacency matrices are fed into their respective graph convolutional networks (GCN), comprised of two layers

each. Both GCNs share the same tweet feature matrix. Dropout [49] is used on each GCN layer, again to prevent overfitting, with ReLU used as the activation function after each layer. This corresponds to step two in Figure 3.3. Additionally, Bian et al. perform root feature enhancement after each GCN layer [5]. This can be viewed as a step akin to residual layers in convolutional neural networks, which would correspond to concatenating the input of the previous hidden layer to the input of the following hidden layer [17]. However, Bian et al. instead concatenate a matrix of n tiled copies of the first row from the previous hidden layer's matrix, with the first concatenation matrix being comprised of tiled copies of the root tweet feature vector $\mathbf{x}_0^i$. This is in an effort to make more use of the information contained from the source tweet [5]. This is step three in Figure 3.3. After the two GCN layers, the output from each GCN is passed through its respective mean pooling layer. These two flattened representations are concatenated into a single vector, which is then passed through several fully connected layers, ending in a final softmax layer. The output vector can be thought of as a probability distribution over the number of classes defined in the training data 3.1. In order to predict a class, we choose the *arg max* of this final vector. The full BiGCN model can be visualised in Figure 3.3, with the final step corresponding to step four in the figure. We have deliberately omitted many of the mathematical details of the BiGCN algorithm as they are not imperative to the aim of this paper[4].

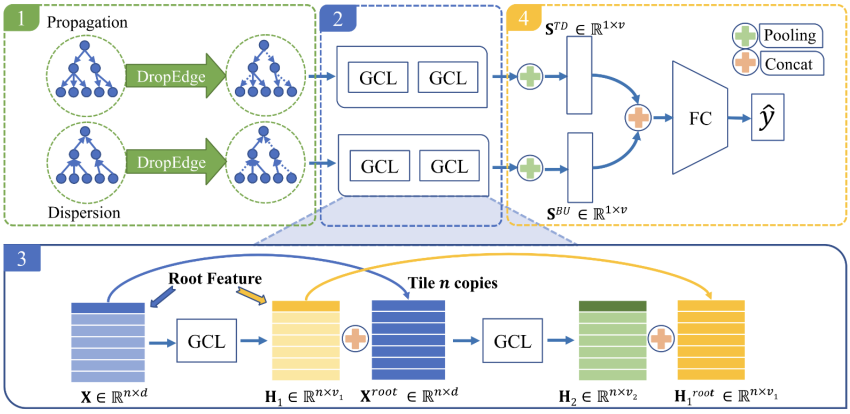


Figure 3.3: A visual representation of the BiGCN model taken from [5].

3.3 Pretraining the Model

We train the BiGCN model, using the *Twitter16* data, as specified in [5] to reproduce the paper's model. The dataset is split randomly into five parts which is used for 5-fold cross validation. The BiGCN, with a hidden dimension size of 64, is trained using stochastic gradient descent with the Adam optimiser ($\eta = 5 \times 10^{-4}$) to minimise the cross entropy loss. The model is trained for 200 epochs, with early stopping on the validation loss and patience set to 10 epochs. DropEdge rate is set to 0.2, dropout rate is set to 0.5 and L2 regularisation is applied to all model parameters with $\lambda = 1 \times 10^{-4}$. Once training is complete, we save the model weights so that they can be used when performing rumour detection on unseen tweets for the rumour detection extension.

[4]For the reader that is interested in these details, we point them to the original paper [5].

Chapter 4

Implementation (RQ.1)

This chapter will outline the key information and the various designs decisions behind creating a usable Google Chrome browser extension for rumour detection on Twitter (RQ.1). Section 4.1 explores how the rumour detection model, explored in section 3.2, is used for real-time inference, and how data is retrieved from the Twitter API. Section 4.2 explores how the web server finds news articles that are semantically related to the source tweet of the cascade. Section 4.3 shows how the centralised web server for the browser extension is created and deployed. Section 4.4 shows the user interface of the browser extension. Section 4.5 explains how the individual components of the browser extension are combined, as seen in Figure 4.1, and details how a user would interact with the system.

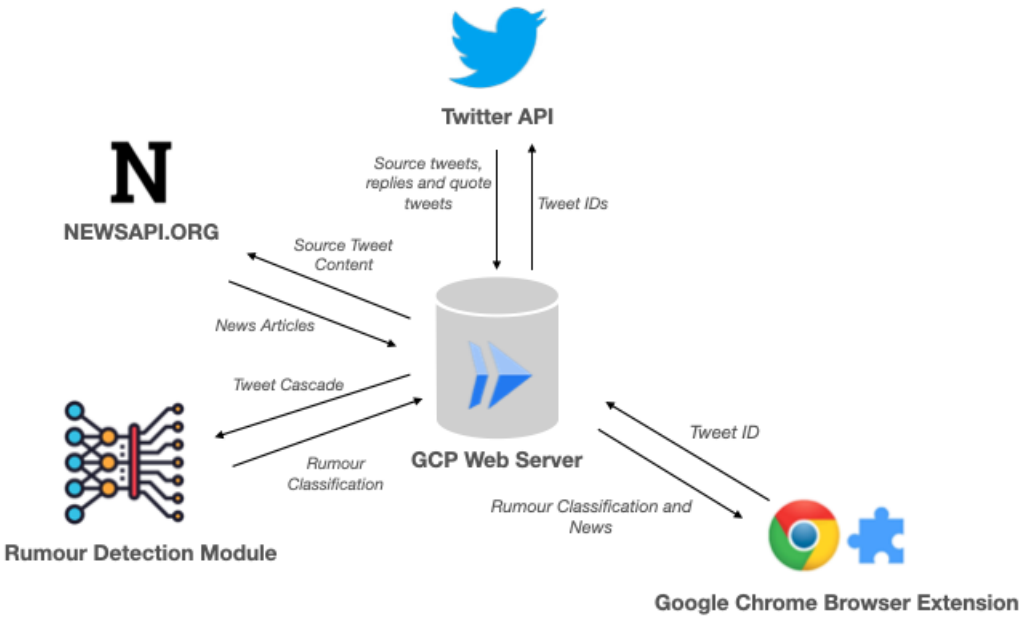


Figure 4.1: Diagram depicting the architecture behind the rumour detection browser extension. Icons are taken from the following sources [1]

[1]Google Cloud Icon Library, Google Chrome Wikipedia, NewsAPI, Machine Learning Icons and Chrome Extension

4.1 Rumour Detection Model Inference

4.1.1 Data Collection from Twitter API

As we saw in section 3.2, our rumour detection model classifies the rumour status of a certain tweet based on information from its tweet cascade. In order to perform inference on new tweets, we need to retrieve the full tweet cascade given a source tweet. Based on our API specification covered in 4.3.2 and 4.5, the web server will receive a single tweet ID via a POST request. This tweet ID refers to the source tweet on which we wish to perform rumour detection. To collect all the raw data regarding the source tweet and the tweets in its cascade, we need to query the Twitter API. We develop an algorithm that captures all the raw data from this API, which we define below:

First, we retrieve data regarding the source tweet from the Twitter API using the *tweets lookup* endpoint, where we feed the source tweet ID as the parameter. If we are able to retrieve information from the API, we may proceed to build the cascade. Conversely, if the tweet is not found, the algorithm fails. Creating the rest of the cascade, and finding news articles that are semantically related (section 4.2), are dependent on the information from the source tweet. Without this information, our browser extension would not function as intended.

Once we have retrieved the source tweet, we reconstruct the tweet cascade. We begin this process by fetching all the replies linked to the source tweet using the *search tweets* endpoint (2.4.1). We approach this chronologically. First, we retrieve the 100 most recent replies. This is achieved by setting the *max_results* parameter to 100. This parameter setting ensures that our API calls are as efficient as possible. Once this is done, we find the tweet ID of the oldest tweet in our returned results. This is passed to a new *search tweets* endpoint request, where we set *unitl_id* to the oldest tweet ID. This returns the next 100 most recent replies. We repeat this process until there are no further replies. This algorithm also ensures that there are no duplicates. During this iterative process, our browser extension may reach the endpoint rate limit (2.4.1). If a selection of replies have already been gathered by our browser extension server, we cache these results and perform under-represented inference. As seen in section 6.3.2, the rumour detection model is able to cope with this limitation. However, if we are unable to retrieve any replies, we fail.

We continue building the tweet cascade through retrieving the quote tweets and retweets related to the root tweet. This is done similar to the reply retrieval above. We utilise the *quote tweets* endpoint, specifying the root tweet ID as the *tweet_id* parameter, and setting *max_results* to 100 for the same reasons as above. When we receive the first 100 quote tweets as a response from the API server, we additionally receive a pagination token. This token allows us to iterate through the ”pages” of results, ensuring that we are able to retrieve all the quote tweets and retweets from the Twitter API. We repeat the *quote tweets* endpoint requests, using the new pagination tokens we receive as the *pagination_token* parameter, until a new pagination token is not provided. This indicates that we have retrieved all the results, and we terminate this subroutine. Similar to the *search tweets* endpoint, we may reach the *quote tweets* endpoint rate limit during our retrieval. To mitigate this, we utilise the same logic as used in the reply retrieval

subroutine.

Designing the two subroutines in this way maximises the performance of our browser extension. We are able to minimise the number of API calls we make to the Twitter API to reconstruct the relevant cascade. This would have otherwise been a major bottleneck in the system if it were done naively. Once we have retrieved all the replies, retweets and quote tweets, the algorithm terminates as we have successfully retrieved all the raw data required to construct the tweet cascade. The algorithm is summarised in Algorithm 1.

4.1.2 Preprocessing the Raw Data

Once we have retrieved the raw data for the tweet cascade as specified in section 4.1.1, this data needs to undergo preprocessing such that the input is in the correct format to be utilised by the rumour detection model for inference. This format is specified in section 3.1. Obtaining a list representation, which is needed to obtain the tweet indices, involves flattening the cascade structure we have scraped from the Twitter API. Due to the nature of the problem, we can leverage a few of the properties to alleviate the computational requirements. Firstly, the tweets in the tweet cascade should be ordered chronologically. As such, this creates a topological ordering of the tweets as nodes, where the parents of any reply or quote tweet would have been created prior to it. We can easily retrieve the tweet ID of the parent as this is included in the raw data representation of each tweet when retrieved from the Twitter API. Once the tweet has been preprocessed, it is passed to the BiGCN to classify the rumour status of the tweet.

4.2 Fetching Relevant News Articles

As mentioned in section 1.2, in addition to providing a rumour classification for a tweet, we also wish to provide articles that are relevant. Particularly, we want for these articles to be semantically related to the source tweet (root tweet of the cascade). As specified in section 2.4.2, we are able to fetch news articles from NewsAPI.org using a keyword-based query. As such, the challenge lies in producing keywords that capture the semantic meaning of the source text. For this task, we made use of the open-source KeyBERT tool [2]. This package leverages embeddings that are created using a Sentence-BERT architecture [39, 9], particularly the *sentence-transformers/all-MiniLM-L6-v2* found on HuggingFace [3], to generate the semantically related keywords. Before passing this text to KeyBERT, we first preprocess the text by passing it through NLTK's tweet tokenizer [4]. This package allows us to remove all Twitter handles and convert all tokens to lowercase. Additionally, we remove all stop words for English, punctuation and any links. Once KeyBERT receives this text, it generates a document-level representation for which we wish to find keywords. In this case, this is the preprocessed source tweet text. Next, candidate keyword phrases are extracted from N-gram sequences in the document text, and word embeddings are computed for these. In our case, we specified

[2] https://maartengr.github.io/KeyBERT/

[3] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[4] https://www.nltk.org/api/nltk.tokenize.casual.html

that all candidate phrases should be between one and three grams. Finally, KeyBERT returns the candidate phrases that are most similar to the document text using the cosine similarity metric, and have been re-ranked using Maximal Marginal Relevance (MMR) [6] to increase the diversity of the keyword phrases. MMR achieves this by measuring the degree of dissimilarity of the current keyphrase, relative to the document text, to all the other keyword phrases that have already been selected. The aim behind increasing the diversity is to provide the user with a wider-range of news articles that would be relevant to the original source tweet text. These keywords are then passed onto the NewsAPI to retrieve news articles that are relevant to these keyword phrases.

4.3 Web-server

In building this browser extension, we decided to implement a centralised web server. This web server powers most of the functionality contained in the browser extension. Its primary functions are: i) accessing the Twitter API in order to retrieve the relevant information needed to model a Tweet cascade, ii) perform rumour detection on the retrieved Tweet cascade with the BiGCN model outlined in section 3.2 and iii) retrieve the news articles that are relevant for the source Tweet using the NewsAPI[5].

4.3.1 Web Server Framework

Due to the fact that the BiGCN in 3.2 was written using PyTorch, we opted to implement the web server in Python. This would allow us to treat the rumour detection model, news article fetcher, Twitter API fetcher and web server code as four separate modules. This allows for easy integration, without the worry of cross-language dependencies. In such a ”Pythonic” environment, two web server frameworks emerge as candidates: Flask and Django. Ghimire [15] provides a comparative study of these two frameworks, where Flask is a microframework, allowing for quick development. Django, alternatively, provides an entire web application framework. For the purposes of this dissertation, we opted for a Flask application: its simplicity is more suited to the requirements of building a proof-of-concept browser extension.

4.3.2 Web Server API

The API of this web server is very simplistic as we only rely on the web server to perform rumour detection on a particular Tweet. The API provides the following routes:

/ The server root receives no arguments, and is the default route of the web server, acting as its landing page. It also specifies the privacy policy of the web server, and the browser extension at large, which is necessary for deploying a browser extension to the Google Chrome store (section 5.1.2).

/inference This path receives, through a POST request, a particular Tweet ID as an argument. This ID is a 64-bit unsigned integer[6] that uniquely identifies a tweet.

[5]https://newsapi.org/

[6]https://developer.twitter.com/en/docs/twitter-ids

It returns a JSON response with the model's classification, or an error, and the news articles relevant to the source Tweet, if any are found.

The web server was designed in this way such that it could be loosely coupled from the browser extension and the specific rumour detection model. This means that the web server can be developed independently of the browser extension itself. Primarily, this allows for rumour detection model to be interchangeable. This is imperative for the browser extension; we want to provide state-of-the-art performance even when newer, more powerful models are released. Additionally, this allows for the browser extension architecture itself to be used in a different context other than rumour detection on Twitter. This will be a recommendation for future work, and will be further discussed in section 7.

4.3.3 Deploying the Web Server

4.3.3.1 Choosing a Centralised Architecture

In order to deploy the web server that will support the rumour detection browser extension, we opted to use a centralised server. The reason for this is two-fold:

1. By creating the centralised web-server, we are able to abstract much of the functionality of this service away from the end-user, as done with TweetCred [16]. In order to interact with the service, the user only needs to provide a specific unique tweet ID to the API endpoint outlined in section 4.3.2. This simplifies much of the user's interaction with the browser extension.
2. We wish to offload as much of the computational complexity onto the web server rather than onto the browser extension itself. As seen in sections 3.2 and 4.2, the rumour detection service requires the use of a complex deep learning model. As such, these models require a great deal of memory and processing power to be run efficiently. Deploying these in a decentralised setting, where the models are run in the user's web browser, would not be computationally feasible.

4.3.3.2 Deployment Platforms

In order to simulate a real browser extension, we decided to deploy the web server on Google Cloud Platform (GCP)[7]. GCP provides users with an array of cloud computing resources and storage solutions. Fortunately, on GCP, there are a number of ways one could deploy the Python-based Flask web server we had created in section 4.3. The main options include [8]:

Compute Engine This deployment strategy involves creating a cloud-based virtual machine (VM). To run the Flask server, the VM instance would have to be running, along with the Python script executed inside the instance. Furthermore, firewall rules would have to be implemented to allow incoming and outgoing HTTP and HTTPS requests and responses.

[7] https://cloud.google.com/

[8] https://cloud.google.com/blog/topics/developers-practitioners/where-should-i-run-my-stuff-choosing-google-cloud-compute-option

App Engine, unlike Compute Engine, App Engine is a serverless platform, described as a Platform-as-a-Service (PaaS). It is used to deploy complete web applications, where network, application and database scaling are handled by the platform. This service allows developers to deploy code, which does not necessarily have to be containerised.

Cloud Run, much like the App Engine, is a serverless platform. However, the difference is that Cloud Run requires you to deploy containerised web applications. Furthermore, the benefit of this service is that infrastructure scaling is handled automatically by GCP. As such, one could bundle necessary files for the web server files into a container image, and the rest of the deployment is managed with GCP. Additionally, Cloud Run is a newer service that builds on the App Engine infrastructure.

4.3.3.3 Containers vs. VMs

As seen above, we have two avenues when considering the deployment of our web-server: containers and VMs. However, as explored in [12], containerised applications currently prove to be the *de facto* standard when deploying web applications and other services. Containers, unlike VMs seen in the compute engine, have a small performance and storage overhead leading to faster start times and an increased computational efficiency. This is due to the fact that containers are far more lightweight than VMs; they do not have to load in a full OS kernel [1, 12]. Containers, instead, virtualise the operating system [47]. As such, you need to specify only the dependencies required for the particular application to run, where the container will be supported by the device hardware [12]. Furthermore, containerised applications are very portable as you have prespecified the environment in which your application will run. The only prerequisite is that the host environment is able to support the specific container. This is unlike in a VM, where you would have to preconfigure the correct environment for your specific application. It is for these reasons that we decided to deploy the Flask web server on Cloud Run.

4.3.3.4 Docker

Given this decision, we choose Docker as our containerisation platform. Docker currently emerges as the front-runner for containerization across different operating systems due to the ease of creating a Docker image and subsequently a Docker container [3, 12]. A Docker image, as described in the Docker documentation, is an immutable, read-only file, that acts as a template for creating a specific Docker container. Each Docker image is composed of a base image. The base image is used to specify the specific environment in which an application will be run inside a Docker container. In the case of our web server, we use a base image that specifies the correct Python version that is to be used for the server container.

This base image can be additionally customised, however, through overlaying additional image layers. This additional specification comes from a *Dockerfile*, which is a text document that specifies the instructions to build a Docker image. This file can be used to install additional dependencies, and add other external files, on top of the base image environment. In our browser extension, this involves onboarding the rumour detection module and web server code, and installing the dependencies for these. Once

the specifications have been outlined in the *Dockerfile*, you are able to build your unique Docker image. A Docker container is simply an instance of the Docker image that is running. This process is depicted in Figure 4.2.

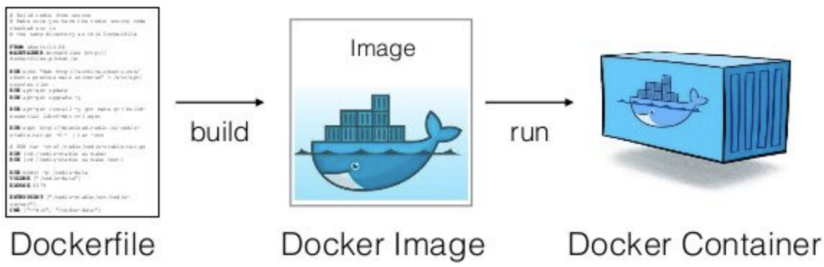


Figure 4.2: Diagram depicting how an application moves from a Dockerfile specification to a runnable container, taken from [9].

4.3.4 Deploying the Containerised Web Server GCP

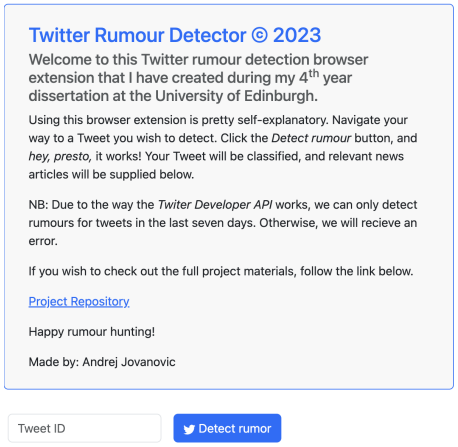
Once a local version of the Docker image has been created, it needs to be uploaded to Google Cloud Platform so that it can be successfully deployed to Cloud Run. To do so, we upload the Docker image to the Google Container Registry. This service is used as a cloud-based private storage solution for Docker images. Once uploaded, there is automatic integration with the Cloud Registry and the Cloud Run service. As such, Cloud Run is able to deploy a container from the prespecified Docker image containing our server automatically.

We deployed the web server with one CPU and a memory limit of one GB. These resources were the minimum requirements needed to support our browser extension infrastructure. In the current state, when the web server does not receive any traffic for 15 minutes, the container running the web server shuts down. As such, there is a delay when rebooting the server once a new request is received. In a real production setting, this is not desirable. However, for the purposes of this dissertation, where the web server is used only to support the user-study, it is sufficient.

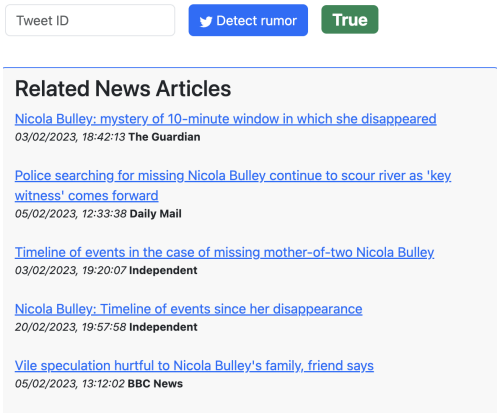
4.4 Google Chrome Browser Extension

The final component of the browser extension architecture is the browser extension itself that is visible to the end-user. Generating the browser extension involved the use of HTML, CSS and JavaScript; the same technologies used to create a web application. This creates a web page that is delivered as a browser extension in Google Chrome. Specifically, we made use of the Bootstrap CSS framework[10]. This framework allowed us to create an aesthetically pleasing web page through using predefined components such as buttons and spinners. Screenshots of the browser extension can be seen in Figure 4.3

[9] https://cto.ai/blog/docker-image-vs-container-vs-dockerfile/
[10] https://getbootstrap.com/



(a) The browser extension's landing page.



(b) The browser extension with results: a rumour classification label of "True" and a list of five semantically related news articles.

Figure 4.3: Screenshots showing the graphical user interface (GUI) of the browser extension.

The user interface is relatively straightforward. In Figure 4.3a, we see that the home page has a description that explains how the browser extension functions. Additionally, we see a detect rumour button that the user clicks when they wish to perform rumour detection. In Figure 4.3b, we see the browser extension interface once results from the rumour detection server have been received. A banner showing the rumour classification label returned from the rumour detection module is shown besides the rumour detection button. Additionally, a list of articles that are semantically-related to the source tweet are displayed, with a clickable link that takes you to the article in question.

4.5 Stitching it all Together: Application Workflow

Once each of the individual components of the rumour detection infrastructure have been established, we are able to view this as a collective unit as seen in Figure 4.1. As such, we describe how a typical user would interact with the browser extension.

1. A user is browsing Twitter via Google Chrome. Once they identify a particular tweet on which they wish to perform rumour detection, the user opens the browser extension and clicks the "Detect rumour" button seen in Figure 4.3a. Once the button has been pressed, the tweet ID is extracted from the URL of the tweet and the client's web browser sends a POST request to the web server, with the specific tweet ID as a parameter.
2. Once the web server receives this POST request, three functions occur sequentially:
   (a) Using the tweet ID, the web server interacts with the Twitter API to retrieve the source tweet, and its respective tweet cascade as outlined in section 4.1.2.

(b) Once the tweet cascade has been retrieved and preprocessed, this is then fed into the rumour detection model seen in section 3.2, and inference is performed. This returns a particular rumour classification label.

(c) Finally, using the source tweet, the web server finds semantically-related keywords and retrieves relevant news articles from NewsAPI.org, as outlined in section 4.2. This returns a list of relevant news articles to the web server.

3. Once the rumour classification label and the relevant news articles have been retrieved, these are returned to the user as a JSON object in response to the original POST request.

4.6 Results and Discussion

Similar to the work done by Gupta et al. [16], we wish to evaluate the browser extension's performance with respect to response time. This is calculated as the amount of time taken for our browser extension to respond to a particular rumour detection request. In our experiment, we measure the response time across 50 randomly selected newsworthy tweets of varying size. We view the CDF of the response times in Figure 4.4.

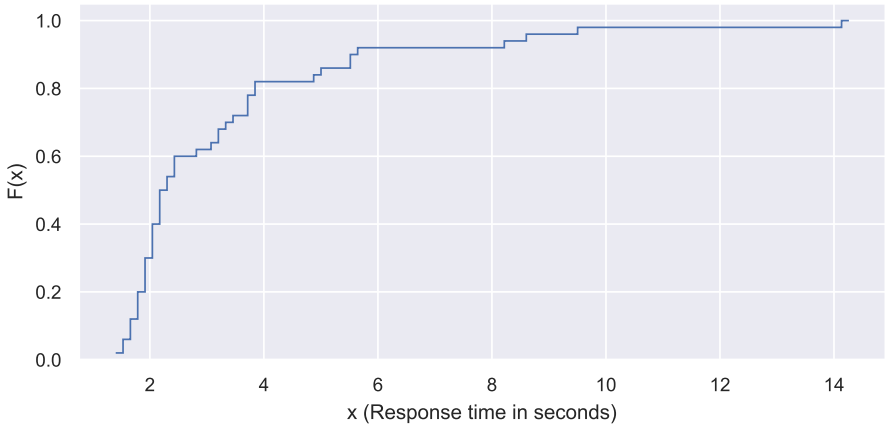


Figure 4.4: Cumulative frequency curve for response time taken for rumour detection across 50 tweets.

Observing Figure 4.4, we see that our browser extension is able to provide a response for 90% of requests in six seconds or less. This is an improvement over the 82% of requests reported by Gupta et al. [16]. However, their experiment analysed the CDF across 5.4 million requests [16]. Due to time constraints with our dissertation, our sample size is significantly smaller. Yet, we can make some initial comparisons between the two solutions. Our work retrieves the entire tweet cascade for a tweet, and predicts the rumour status using the BiGCN model. TweetCred, on the other hand calculates 45 handcrafted features based on the tweet itself, and uses SVM-rank to assign a credibility score. Given the fact that our approach uses far more information per tweet, and a more

sophisticated and computationally intensive model [11], this is still an encouraging result for our browser extension.

Unfortunately, the other browser extensions mentioned in section 2.3 do not report results for a response time experiment. As such, we are unable to compare our extension to other solutions on this particular axis.

4.6.1 Limitations

The browser extension has a few limitations that are linked directly to its interaction with the Twitter API. As noted in section 2.4.1, each of the API endpoints that we use have a specific rate limit attached to their use. As such, we note that requests to the API server could fail in two scenarios: i) when the browser extension server has already received some data or ii) prior to the browser extension being able to retrieve any data. The first case is less problematic: in section 6.3.2, we show that the rumour detection model is still able to maintain its predictive performance with an under-represented tweet cascade. The second case, however, means that the user of the browser extension will be unable to use the service for a period of 15 minutes. We have attempted to mitigate this limitation by minimising the amount of API calls. However, this is still a consideration for high-frequency users. With the current Twitter API specification, there is no solution to removing the rate limit.

Another limitation arises from the changing access requirements for the Twitter API. At the time of writing, Twitter API users will no longer be able to make use of the GET API endpoints, which include the endpoints that we described in section 2.4.1, under the free tier. Instead, users will have to pay $100 per month [12]. As such, this limits the use-case of the browser extension that we have created. However, the flexible architecture that we have created allows the browser extension to be ported to a different context. Possible directions for future work are discussed in section 7.3.

A final limitation of our browser extension is linked directly to the way the BiGCN represents textual features. Namely, the tweet is represented only by those tokens that can be encoded, as an *index:count* pair, if they occur in the training data dictionary (section 3.1) [5]. As a result, situations may arise where tweets cannot be represented at all, apart from the <END> tag. We posit that this would impact on the predictive performance of the rumour detection model due to the importance of textual features in rumour detection models as seen in section 2.2.2. We carry out a related experiment in section 6.3.1.

[11] Most notably, our browser extension was able to classify a tweet cascade with over 579 retweets, 343 quote tweets and 454 replies in $\sim$14 seconds.

[12] https://developer.twitter.com/en/portal/products/basic

Chapter 5

Application Evaluation: User Study (RQ.2)

After building the browser extension architecture (RQ.1), we conduct a user study in order to evaluate the efficacy of our tool (RQ.2). This is achieved through asking participants of the user study to perform several rumour detection tasks using the browser extension. The architectural prerequisites and the distribution of the browser extension are specified in sections 5.1.1 and 5.1.2 respectively. The design of the user study is specified in section 5.1.3, with the associated results and discussion in section 5.2. The participants' information sheet and consent form are provided in Appendix B and C respectively. The full user study questionnaire is provided in Appendix D

5.1 User Study

5.1.1 Preparing the Browser Extension

In order to launch an effective user study, we had to tweak the supporting browser extension server architecture slightly. This would allow the user study participants to use the browser extension only for the user study, and for no other purposes. To achieve this, we restricted the unique *tweet IDs* to which the server would respond. If the requested *tweet ID*, included in the arguments of the POST request, was not included in the prespecified allow-list of *tweet IDs*, the browser extension would return an error.

Furthermore, in order to speed up the browser extension, we cached the representations of the prespecified tweets. This removes the need for the browser extension to communicate directly with the Twitter API as it has a local store of all the data. Additionally, this creates a fair user study as all participants will receive the same rumour status classifications based on the underlying tweet cascades.

5.1.2 Distributing the Browser Extension

In order to distribute the browser extension for the user study, we uploaded it to the Google Store. The motivation behind this was to simplify the download procedure for

users when completing the user study. Uploading an extension to the store requires the browser extension to be approved to ensure that the extension is safe to use, an adequate privacy policy has been implemented, and does not infringe on any terms and conditions. The links to the privacy policy and browser extension can be found at the following links: [1] and [2]. Once the extension had been approved, we released it for public download. However, the browser extension could only be accessed via a specific URL that was given to participants during the user study. Again, this was done to restrict the browser extension's use to the user study only.

5.1.3 Preparing the User Study

The purpose of the user study is to evaluate how helpful users find the browser extension, and whether the browser extension indeed aids them in detecting rumours on Twitter. This will allow us to assess whether the rumour detection browser extension is a viable tool that would help Twitter users more generally. We also allow the participants to provide possible areas of improvement for the browser extension. The user study questionnaire was created using Google Forms. This platform allowed us to create the user study with question branching, which was useful if any errors were encountered during the *setup* phase, seen below. Furthermore, it allows us to collect participant responses anonymously, and to share the study through a URL included in an email. The user study is broken up into four main components:

Setup The first part of the questionnaire is devoted to providing a step-by-step guide to setting up the browser extension and the server architecture for the user study. Firstly, the user is provided with detailed instructions to install the browser extension on their local Google Chrome browser. After which, the user is asked to confirm if they were able to do so successfully. If not, we allow the user to specify the reasons for which they were not able to install the browser extension. Then, the user is asked to confirm that they are able to access the web server's privacy policy. This step is necessary to ensure that participants are able to access the web server as it may have shut down, as explained in section 4.3.4. Finally, the user is asked to perform a sample rumour detection task to ensure that the service is functioning properly.

Rumour Detection Tasks Once the participant has successfully set up the browser extension, they are then asked to complete rumour detection across five prespecified tweets. For each tweet, the participant indicates whether they believe that the tweet contains a rumour *prior* to using the browser extension. After performing rumour detection, the user is asked to evaluate whether the browser extension supported their experience in any way. This could be through flagging potential rumours, or providing news articles that gave additional informative context for the participant. Finally, the user is allowed to add further comments for each rumour detection task, if they wish to specify their reasoning behind their responses. Of the five tweets, the first four were chosen to be news-related so that they could be maximally-aligned to the training data. However, the fifth tweet is more of a tabloid-style article which was included to evaluate the model's performance on unseen data.

[1] Web Server Privacy Policy
[2] Google Chrome Browser Extension

Global Feedback Once the participant has completed the five rumour detection tasks, they are asked to comment on their overall experience using the browser extension. Furthermore, the participants are given the opportunity to provide any additional feedback that they may have for the browser extension.

Deletion Whilst the browser extension is not harmful, it does not provide any use for the user outside this user study. As such, we lead the user through the steps required to delete the browser extension.

5.2 Results and Discussion

The user study had a total of 19 participants, all of whom were able to successfully install and delete the browser extension. Recruiting participants for this study was challenging as there was no other way to incentivise participants other than through the intrigue of the project itself. Nevertheless, the results of this study were very informative in assessing how useful this browser extension would be if it were deployed live. Additionally, the participants in the user study provided valuable suggestions for future work. In this study, we did not include a control group. The reason behind this decision is that for the browser extensions mentioned in section 2.3, either their use-case was not directly compatible with rumour detection, or the browser extension was not actively maintained. This latter condition would result in an unfair comparison. Furthermore, many of the browser extensions mentioned in section 2.3 did not conduct a user study. In the case of [50, 25, 45, 40], the tool that they propose was a secondary contribution alongside the architecture behind the solution. As a result, the authors did not focus on the usability of their service. This motivates the unique contribution that this user study has in our work. We provide the results from each section of the user study, and an associated discussion, below:

5.2.1 Agreement on Browser Extension Aid

As mentioned in section 5.1.3, we asked each user study participant to respond to two questions for each rumour detection task: i) whether they had reason to believe that the tweet contained a rumour prior to the rumour detection task and ii) whether the browser extension supported their experience in any way. For each of these questions, we wanted to measure the inter-annotator agreement across the 19 participants in our user study using Randolph's kappa [38] as a metric [3]. The results from the user study are presented below.

In Figure 5.1, we see that participants had varying opinions on whether they found the tweets to be rumours prior to preforming rumour detection. The Randolph's kappa [38] score across the 19 annotators was $\kappa = 0.345$. This shows that discerning whether a tweet is a rumour or not is a difficult task for a human, motivating the need for supportive tools such as this browser extension. Furthermore, prior to the rumour detection task, we see that users believed that tweets four and five were examples of tweets that contained

[3]This metric measures multi-annotator agreement over a number of categories. The score $-1 \leq \kappa \leq 1$, where -1 show unanimous disagreement, 1 shows unanimous agreement and 0 is no agreement.

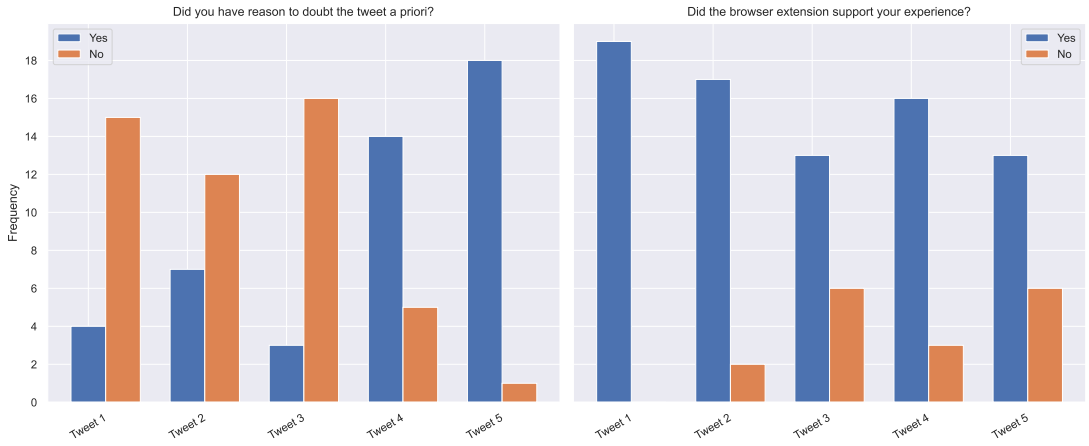


Figure 5.1: Results from the rumour detection tasks in the user study.

a rumour. Conversely, tweets one, two and three were believed to be examples of tweets that contained a non-rumour.

Reviewing the performance of the browser extension, we see that, generally, annotators found that the browser extension supported their rumour detection experience. The Randolph's kappa [38] score was $\kappa = 0.443$ which supports the fair agreement on the browser extension's positive impact on their rumour detection experience. We also see that the participants in our user study agree more on the browser extension's support than our the status of rumours *a priori*. Furthermore, an interesting result is that this kappa score is similar to the user agreement on the credibility score (43%) that Gupta et al. obtained for TweetCred [16]. Whilst the questions posed to the users are different, these results show that there is some benefit to be gained through using additional rumour detection tools. However, there needs to be additional measures put into place to make users more confident in the tool's performance, which would lead to higher user agreement.

Specifically, we find that the users were most satisfied with the browser extension's performance on Tweet 1. According to the participants, this was due to the fact that the browser extension provided additional context through a diverse range of news articles, confirming their initial beliefs. A similar trend can be seen with Tweet 2. However, this tweet was a false claim, which was contrary to the participants' beliefs *a priori*. As such, the participants reported that the browser extension was able to debunk the claim, and provide related articles that supported the debunking. However, we see that there was less agreement on the browser extension's support in tweets 3 and 5. In tweet 3, the browser extension failed to provide a diverse set of related news articles. Instead, it provided the news article on which the source tweet was based. However, we expected this to be the case. The news event in question was Nicola Sturgeon's resignation from office, which at the time was a novel event. As such, this is more an issue with the novelty of the claim, and the NewsAPI service not having access to a diverse set of related news articles, and not the browser extension itself. However, for Tweet 5, the reason for the drop in user experience is that the browser extension was not able to debunk a tabloid tweet that was obviously false. Furthermore, it was unable to find any

related news articles because the textual content of the tweet was not linked to any news report. From this, we see that the browser extension has difficulty performing rumour detection on tweets that lie outside the training data distribution. This issue is addressed in more detail in section 6.

5.2.2 Global Feedback

Additionally, to the aforementioned rumour detection tasks, we wanted to determine the global feedback of the browser extension from the participants in the user study. For this global feedback, we asked the users to rate their agreement to three questions on a five-point Likert scale. The questions and the distributions of results across the five Likert categories can be seen in Figure 5.2. In the results, we see that the feedback for the browser extension is generally positive. 78.95% of the participants in the user study agreed or strongly agreed with finding the browser extension to be useful. 89.17% of the participants agreed or strongly agreed with recommending this browser extension to friends or family, and 84.21% of the participants would use the browser extension again in their personal time.

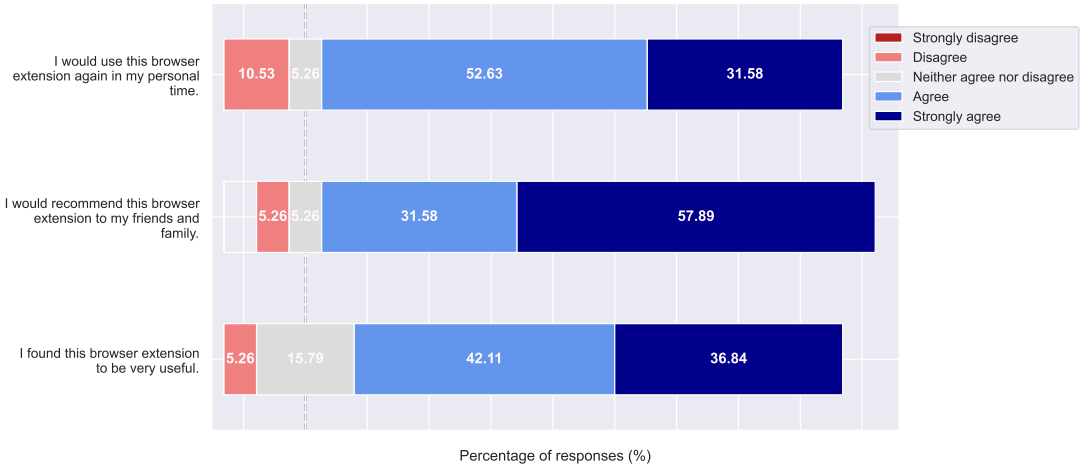


Figure 5.2: Results from global feedback in the user study.

These results show that the participants of the user study, generally speaking, find the browser extension to be a useful tool in their daily social media use. However, we see that there was a small portion of users who either disagreed or were ambivalent to the three statements. A potential reason for this is that this study did not require the participant to be a Twitter user. As such, the study could have attracted participants who do not use Twitter frequently, or those users that do not have a Twitter account at all. These users would not see the need to have access to a tool such as this browser extension as they would have no use for it personally. This explanation can be extended to whether the participant would recommend the browser extension to their friends or family.

5.2.3 Additional Feedback

As had been mentioned in the design of the user study, we also provided the participants a chance to provide any additional feedback for the browser extension. Across the 19 participants, there were two areas of feedback that were very significant: i) explainability of the rumour detection model, and ii) label definitions.

With regard to this the first feedback point, users of the browser extension commented on the fact that it would be useful to know how the rumour status label for a tweet is generated. The topic of explainable artificial intelligence (AI) has gained much traction in recent years due to the success of deep neural networks (DNNs). The issue, in terms of explainability, of these models is that they are a black-box: they cannot be explained by the neural network itself, nor by a human [56]. The BiGCN, the rumour detection model used in this system, falls in this class of model. In the context of the browser extension, we could potentially sidestep this issue of explainability by providing the user an explanation of which data is gathered by the model, and the theoretical hypothesis behind the model's decision-making. However, this still does not explain what exactly, with regard to the tweet cascade, did the neural model prioritise to make its assessment on the rumour status of the model. This interpretation would be far more useful to the end-user, rather than a general, theoretical comment on the rumour detection model itself. As such, we agree that extending explainable AI methods to the BiGCN [26, 58], and deep learning-based rumour detection models more generally, is a worthwhile area of research. However, this is beyond the scope of this dissertation, and we leave it for future work.

The second point of feedback raises an issue in that the users of the browser extension had difficulty discerning the difference between various veracity labels assigned by the rumour detection model. Specifically, the difference between the labels of "non-rumour" and "true" was particularly troublesome. As mentioned in section 2.1, the various veracity labels used by the rumour detection model are often misused due to their nuanced differences. For the everyday user who is not versed in rumour detection literature, these nuances are not clear *a priori*. As such, we accept the validity of this issue. Fortunately, rectifying it is very simple: we could attach additional explanations, or the definitions outlined in section 2.1, for each label.

5.2.4 Limitations

A limitation of this user study lies in the small sample size. Furthermore, the participants themselves could be biased in their evaluation of the browser extension because of their relation to the author; the browser extension was shared via a university mailing list. However, due to the anonymity of the study, we hope that the participants of the user study would be objective in their assessment of the browser extension. Nevertheless, we find that these initial results support and encourage the viability of a Twitter rumour detection extension. However, a potential direction for future work would be extending the browser extension to a wider, more diverse audience.

Chapter 6

Model Performance on OOD and Imperfect Data (RQ.3)

In RQ.1 and RQ.2, we created our browser extension and evaluated its efficacy through a user study, respectively. In RQ.3, we conduct experiments to assess how well the rumour detection model will perform on out-of-distribution (OOD) and imperfect data. We provide a motivating example as to why these situations are frequently occurring for the rumour detection browser extension in section 6.1. We conduct baseline experiments to prove our OOD hypothesis, and provide simple solutions to mitigate this issue in section 6.2. Furthermore, we assess the performance of the rumour detection model when it receives imperfect data in sections 6.3.1 and 6.3.2. Our experiments create a greater understanding of the work done by Bian et al. [5], Additionally, they highlight important issues that affect machine learning models once deployed.

6.1 Motivation

As seen in section 3.2, the browser extension uses a state-of-the-art rumour detection model to classify, automatically, the rumour status of any tweet based on its cascade structure and textual content. It is generally well accepted that machine learning methods are able to generalise well (if trained appropriately) to data that is unseen, but comes from a similar distribution to the training data. However, as we saw in section 5.2, the rumour detection model did not perform well when we provided it with an example that was not news-like. This example would be considered OOD relative to the data on which the model was trained. This is confirmed by the work in [21, 18, 13]. Another pertinent consideration relates to how concept drift occurs on social media, and the effect that this would have on predictive performance for rumour detection models as shown by [19]. We posit that these aforementioned situations would be frequently occurring for a rumour detection browser extension. End-users would be interested in performing rumour detection on a multitude of topics other than those that are news-related (e.g. sports trade rumours, celebrity gossip etc.). Furthermore, the nature of discourse on Twitter frequently changes as new events occur. As such, the users would expect that the extension, and specifically the machine learning model, would perform

well in these above situations. As such, the model would need to cope with OOD in some manner. We explore a related experiment in section 6.2. Furthermore, due to the limitations of the Twitter API mentioned and the BiGCN model mentioned in section 4.6.1, the inference data provided to the rumour detection model may be incomplete. We explore these experiments in sections 6.3.1 and 6.3.2.

6.2 Dataset Mixing with Discussion

To simulate the rumour detection model performing inference on OOD data, we leverage the two datasets mentioned in section 3.1. Specifically, we train two versions of the BiGCN that was used as the rumour detection model in our browser extension: one on the Twitter15 dataset, and the other on the Twitter16 dataset. Once we had trained the two models, we assessed their performance on their own data, through reporting the results from each model's 5-fold cross validation. We also evaluate each model on the alternate dataset by performing inference across five randomly generated folds. For both models, we followed the training procedure outlined in section 3.3. The performance metrics that we consider are those used by Bian et al. in their work [5]: the model's accuracy, and the F1 score for each of the four class categories mentioned in section 3.1. The results of this experiment are visualised below:

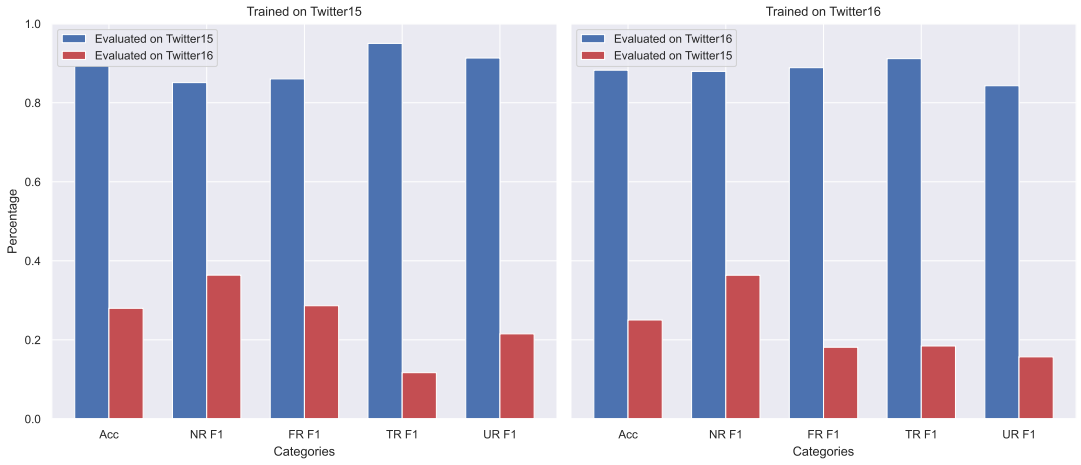


Figure 6.1: Result from the baseline cross-data evaluation experiment for the Twitter15 and Twitter16 models.

As seen in Figure 6.1, both the Twitter15 and Twitter16 model are able to perform extremely well when generalising to data that is similar to its training data. The results obtained are in accordance with the results reported in the original paper by Bian et al. [5]. However, when we evaluate the models on the alternate dataset, the performance of both the Twitter15 and Twitter16 models dramatically decays. These results confirm that the BiGCN model suffers from the issue mentioned in section 6.1. As a result, such a model, with this naive setup, would be ill-suited to be deployed on a production server and used with the rumour detection browser extension.

Consequently, we then raise the following question: would a model, trained on both sets of data, be able to perform equally well on both datasets? To set up this experiment,

we generate a third dataset by selecting and combining a portion p_{T15} of the Twitter15 dataset, and a portion p_{T16} of the Twitter16 dataset. This gives rise to a TwitterMix dataset. We then train three separate models: i) a Twitter15 model on the leftover $1 - p_{T15}$ of the Twitter15 data, ii) a Twitter16 model on the leftover $1 - p_{T16}$ of the Twitter16 data and iii) a TwitterMix model on the TwitterMix data. We follow the same training, and evaluation structure as set out at the beginning of section 6.2. The results of the experiment, across the performance metrics mentioned above, where we set $p_{T15} = p_{T16} = 0.5$ are visualised in Figure 6.2

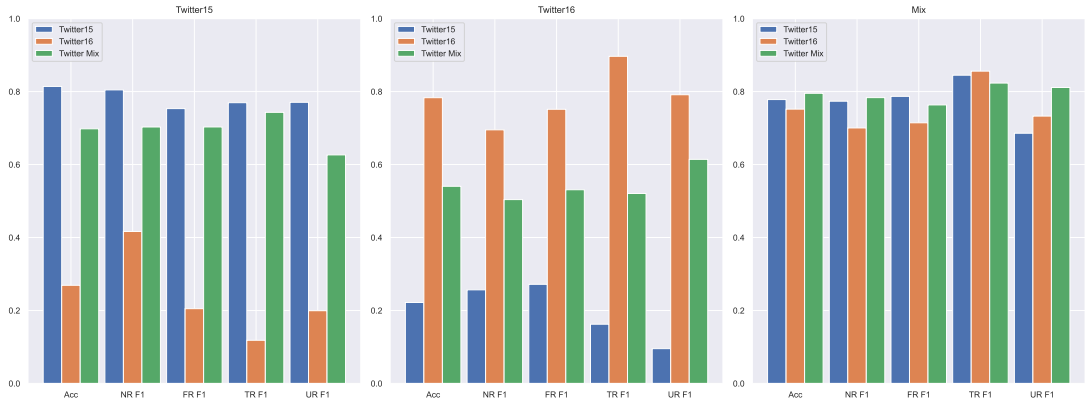


Figure 6.2: Result from the dataset mixing experiment for the Twitter15 and Twitter16 models, with $p_{T15} = p_{T16} = 0.5$.

In Figure 6.2, we still observe the same effect as was seen in the baseline experiments for the Twitter15 and Twitter16 models, as was expected. However, we find that the TwitterMix model is able to generalise far better to OOD data than its respective counterparts. This can be seen by the fact that this model's performance degrades significantly less when tested on OOD data (Twitter15 and Twitter16 in this case). Even though this is degradation is quite noticeable on Twitter16 data, this shows that a simple technique of training a model on multiple sources of data is able to improve its performance. Furthermore, we see that Twitter15 and Twitter16 models are able to perform well on the TwitterMix data, which was the desired effect. The TwitterMix dataset is composed of examples that were originally from the Twitter15 and Twitter16 datasets, which can now be considered to be "in-distribution" for both models. As such, each of the models that we have trained is able to predict those examples that originated from their training distribution very well. Furthermore, it is likely that both models are able to predict certain tweets from the alternate dataset well too, leading to great performance on the TwitterMix data. We preform additional experiments (seen in Appendix A) where we alter the proportion of p_{T15} and p_{T16}. The experiments show that the extent to which these models preform well on the TwitterMix dataset is indeed dependent on the training data proportions, whilst the TwitterMix model is able to generalise better than the Twitter15 and Twitter15 model in all cases.

These experiments show that performing dataset mixing, as a simple baseline, helps the BiGCN generalise to data that was originally OOD. Additionally, this highlights the importance of model monitoring and updating required when deploying the BiGCN [36, 50]. With adequate solutions, the model will be able to mitigate the effects of

concept drift and OOD when performing rumour detection on Twitter data [19, 28, 36]. The work of Thilakarathna et al. incorporates this feature with their training engine mechanism as they are able to update their models regularly with incoming data [50]. However, exploring such strategies is beyond the scope of this dissertation, and is left for future work.

6.3 Data Ablation Study

As mentioned in section 6.1, for the browser extension to provide accurate rumour classification labels, it needs to be able to represent the tweet, and its associated cascade, as accurately as possible. Without the complete cascade and/or textual feature information, the predictive performance of the classifier could be impacted. This is due to how the training data has been collected and represented. To this end, we provide two experiments in sections 6.3.1 and 6.3.2.

6.3.1 Textual Feature Ablation with Discussion

One condition where a certain tweet cascade is under-represented is if the model is unable to represent the textual features of a tweet. The BiGCN model, as mentioned in section 3.1, cannot represent textual content that does not occur in the training data dictionary. To simulate this effect, we run a textual ablation study. First, we generate new versions of the Twitter15 and Twitter16 datasets according to some textual ablation proportion. In these new datasets, we randomly replace, according to the specified proportion, the textual features of a given tweet with *0:1*, which is the corresponding *index:count* pair for an <END> tag. Once the new datasets have been created, we train a Twitter15 and Twitter16 model, and report the performance on five-fold cross validation as done in section 6.2. This was done for the textual ablation proportions: 0%, 50%, 70%, 90%, 100%. We view the results of this experiment in Figure 6.3.

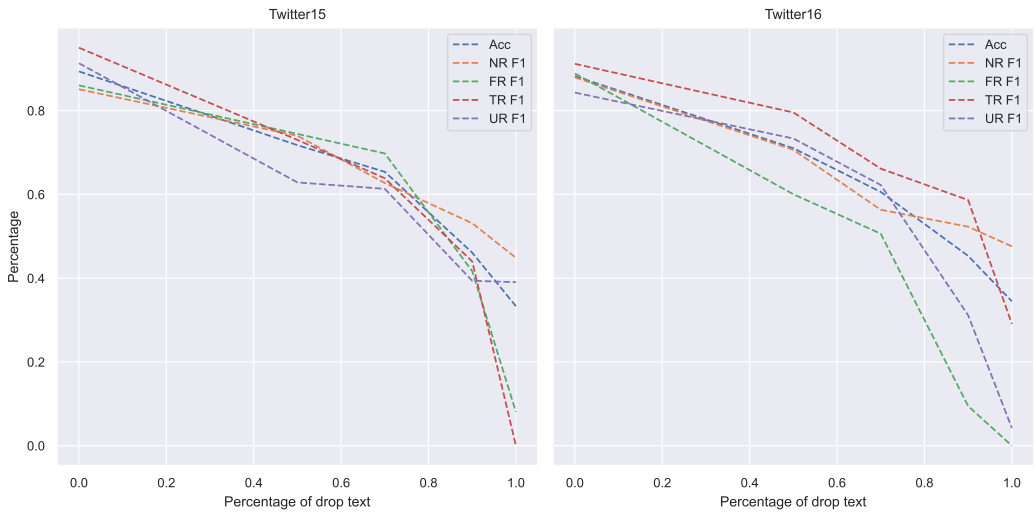


Figure 6.3: Results from the textual ablation experiments, across varying ablation proportions, for the Twitter15 and Twitter16 datasets.

Unsurprisingly, we see that the performance of the two models dramatically decays as we increase the proportion of tweets that have their textual features removed. These results were expected due to the importance of textual features to rumour detection models. In all the automatic methods discussed in section 2.2.2, textual features have provided an important signal in predicting the rumour status of a particular tweet. As such, we observe the same effect with the BiGCN model. However, these results still serve to stress the importance that without the proper, and full, representation of tweet cascades, the performance of the rumour detection model drops dramatically. Furthermore, these results extend our understanding of the capabilities of the BiGCN architecture: a complementary experiment was not included in the original paper [5].

6.3.2 Node Ablation with Discussion

Another condition where an impoverished tweet cascade representation could occur is if the web server is unable to retrieve the full cascade from the Twitter API. This condition is a realistic concern for the browser extension due to the limitations mentioned in section 4.6.1. Similar to the textual feature ablation experiment 6.3.1, we first generate new versions of the Twitter15 and Twitter16 datasets according to some node ablation proportion. However, instead of removing textual features, we randomly remove nodes, and their descendents, from every tweet cascade [1]. This was done to simulate the scenario where certain tweets would not be retrieved by the Twitter API. Once these new datasets have been created, we train a Twitter15 and Twitter16 model and report the performance on five-fold cross validation as done in section 6.2. We repeat this experiment for the following node ablation proportions: 0%, 50%, 70%, 90% and 99.9%. The results of these experiments are specified in Figure 6.4.

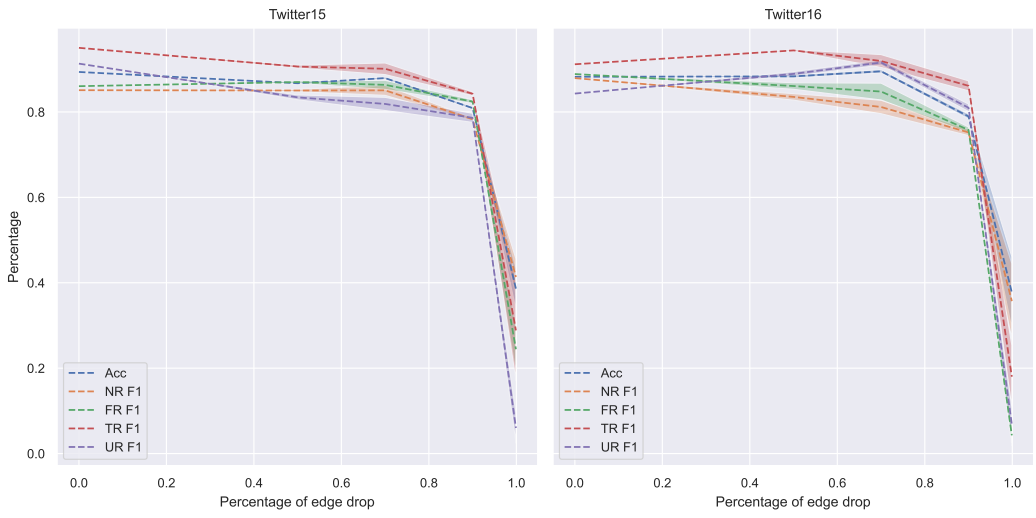


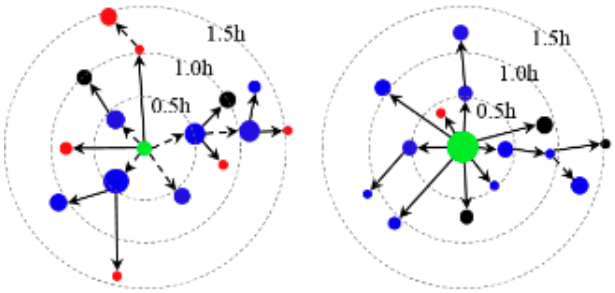
Figure 6.4: Results from the node ablation experiments, across varying ablation proportions, for the Twitter15 and Twitter16 datasets.

Surprisingly, we did not observe the same effect as was seen in the textual ablation

[1] We did not remove the root nodes from the tweet cascades as the label for each tweet cascade is tied to the root node.

experiments (6.3.1). Instead, we see that the BiGCN was able to was able to maintain, and sometimes improve, performance relative to the baseline, across both datasets. This was achieved until over 90% of the tweets in each cascade had been removed. After this point, the models' predictive capability sharply decreased - once enough tweets had been removed, both model lose enough signal from the input data to accurately classify the rumour status. However, we can contrast these results with the early rumour detection study in the original paper by Bian et al. [5]. In their work, Bian et al. examine the BiGCN's performance on the task of early rumour detection. This experiment aims to show that the model is able to learn to predict the rumour status of the tweet cascades in the early stages of its propagation (i.e, where there are not many nodes in the cascade structure). Although our experiment does not remove tweets from the cascade temporally (as we do not have access to the timestamp for each tweet), we are, essentially, creating an experiment that is very similar to the early detection experiment in [5]. As such, we observe very similar result curves when we compare the two experiments.

However, an interesting result is that in both datasets, we observed that the models were able to score better than the baseline even with fewer tweets representing the cascade. A possible reason for this is that when randomly removing a tweet, and its descendents, from a cascade, we have no rules enforcing what type of tweets are removed from the cascade. If we observe the rumour tweet in Figure 6.5, tweets that express doubt in response to a root tweet indicates that the tweet is potentially a rumour [23]. As such, removing the tweets that express support make the tweet seem more rumour-like. Similarly, removing the tweets that express doubt from the non-rumour would make this tweet more non-rumour like. As such, these situations would make each of the tweet cascades seem more like a prototypical example of their respective class.



(a) A rumor (b) A non-rumor

Figure 6.5: Prorogation structure of two source tweets taken from [32]. Red nodes express doubt, blue nodes express neutrality and black nodes express support. The green node is the root of the cascade.

A similar effect could, by chance, be observed in our node ablation experiments. By randomly removing certain nodes, we inadvertently simplify the rumour detection task for that tweet as it would seem more prototypical of its class. These experiments show that the rumour detection model is able to maintain its baseline performance even though it does not have access to the full tweet cascade for a given tweet. This is essential for the functioning of the rumour detection browser extension due to the limitations specified in 4.6.1, and the importance of early rumour detection.

Chapter 7

Conclusion

In this chapter, we provide a summary of the main achievements in this dissertation in section 7.1. In section 7.2, we summarise the results of this dissertation, and we propose steps for future work in section 7.3.

7.1 Contributions

The major contributions of this project are as follows:

1. The full infrastructure for a rumour detection browser extension on Twitter and the deployment of these services using Google Cloud Platform and the Google Chrome store.
2. Created the infrastructure surrounding the browser extension in such a way that it can be ported easily to a different application domain.
3. A critical evaluation of the browser extension as tool, in the form of a user study, where users were asked to perform a series of rumour detection tasks on Twitter.
4. Analyse the performance of a state-of-the-art rumour detection model on out-of-distribution and imperfect data, which are pertinent to the successful deployment and functioning of the browser extension.

7.2 Summary of Results

In this dissertation, we have explored the practical deployment of state-of-the-art rumour detection models for Twitter in the form of a browser extension. Our goal was to provide a service that was useful for the end user: it would allow them to leverage the benefits of said models during their ordinary social media browsing. We were successful in this task, confirmed both by the impressive latency we reported in section 4.6 and the positive feedback from the user study in section 5.2. However, our user study also highlighted areas of potential improvement for the browser extension. These suggestions show that there are many intriguing research avenues to explore in extending the proof-of-concept system we have created in this dissertation.

Additionally, we also experimented with the performance of our model on OOD and imperfect data. Our results show that this model is able to cope with under-represented tweet cascade structures. However, it struggles to maintain its state-of-the-art predictive performance when evaluated on out-of-distribution data or when it is not able to represent the textual content of the tweets in the cascade sufficiently. These results are interesting for two reasons. Firstly, these experiments increase our understanding of the work done by Bian et al. [5] and highlight the limitations of this model. Secondly, the results show that additional mechanisms need to be put in place to ensure that the browser extension functions optimally once deployed. As such, these motivate many of the suggestions of future work in section 7.3.

7.3 Future Work

We successfully answered the three research questions that were set out as the main objectives of the project. However, there are many interesting research questions that have arisen as a result of our work, and are worthy of future exploration. These research extensions fall into two domains: i) extensions to the BiGCN model and ii) extensions to the browser extension itself.

7.3.1 Improvements to the rumour detection model

As mentioned in section 6.3.1, one of the flaws of the BiGCN architecture lies in the way it represents the textual content of each tweet. Due to the way the dictionary is computed, we may see that a large portion of a tweet's textual content could be ignored due to the fact that its tokens are not found within the dictionary formed through a naïve tokenization process. In the neural machine translation (NMT) literature, researchers have faced the same problem. As a solution, Sennrich et al. propose byte-pair-encoding [44]. This tokenization process improves NMT quality by reducing the number of unknown words encountered. Instead of using the naïve tokenization, sub-word tokenization methods could be explored for the BiGCN to see whether they would improve the predictive performance of the model. Furthermore, another issue with the BiGCN model, as reported by the participants in our user study, is that it lacks explainability. This takes away from the benefits provided by the predictive performance of the BiGCN as users are unable to interpret the decision process behind the model's classification. Fortunately, there is an array of interesting approaches in applying explainable AI techniques to graphical neural networks [26, 58]. As such, future research could explore applying these techniques to the BiGCN specifically.

7.3.2 Improvements to the rumour detection extension

This project has successfully shown that deploying a rumour detection extension for Twitter is both feasible and provides benefit for the end user. However, since this was a proof-of-concept system filling a gap in the research (2.3), there are many avenues to explore in improving the system.

In section 4.6.1, we commented on the limitations that were related to future Twitter

API use. Due to the design of the browser extension architecture, a possible direction for future work would be to deploy the same service, albeit in a different context. Instead of focussing on Twitter, a similar use-case would be found with Sina Weibo, for example. The browser extension could focus on fake news detection, rather than on rumour detection. Each of these disciplines have their own state-of-the-art solutions in the literature, and exploring practical tools that would leverage their performance would be a worthwhile research direction.

As shown in section 6.3.1, when deploying a machine learning model, monitoring the model's performance, and updating its weights, is a pertinent consideration. Particularly in a context such as Twitter, the rumour detection model used for the browser extension needs to be able to keep up with recent developments in the data, and be aware when it is deviating from it is typical behaviour [36]. Fortunately, this is a relatively common issue faced when deploying machine learning solutions, and there are a number of strategies to ensure that the model is performing optimally.

Firstly, we could expand the machine learning pipeline used by the browser extension to include a model that performs outlier detection. This purpose of this model would be to flag up potential tweets that are OOD prior to passing these instances to the BiGCN model. As such, this would act as a prescreening method at inference time, ensuring that the model classifies data that originates from the training data distribution [21]. This could alleviate the drastic decay in model performance observed in section 6.2. However, this additional model would aid the browser extension under the assumption that the overall data distribution has not changed, but not in the case of concept drift. With concept drift, the entire distribution of the data would have fundamentally changed. As such, every new tweet will seem as an outlier, rendering our browser extension inutile. In this case, we would have to ensure that our rumour detection model is continually updated to cope with these effects [36]. Online retraining, as shown by the work in [19], has been shown to be effective in minimising the effects of concept drift. Furthermore, Diethe et al. [10] show a more sophisticated paradigm with their continual learning approach. Future work of this browser extension could explore these research avenues in order to ensure that our system functions well once it has been deployed.

Finally, once the browser extension had been developed further, the scope of the user study could be expanded to a larger, more diverse audience.

Bibliography

[1] Babak Bashari Rad, Harrison Bhatti, and Mohammad Ahmadi. An introduction to docker and analysis of its performance. *IJCSNS International Journal of Computer Science and Network Security*, 173:8, 03 2017.

[2] Benjamin Bates. Yochai benkler. the wealth of networks: How social production transforms markets and freedom. *Journal of Media Economics*, 20:161–165, 05 2007.

[3] Ouafa Bentaleb, Adam S. Z. Belloum, Abderrazak Sebaa, and Aouaouche El-Maouhab. Containerization technologies: taxonomies, applications and challenges. *The Journal of Supercomputing*, 78(1):1144–1181, Jan 2022.

[4] Alessandro Bessi, Antonio Scala, Luca Rossi, Qian Zhang, and Walter Quattrociocchi. The economy of attention in the age of (mis)information. *Journal of Trust Management*, 1(1):12, Dec 2014.

[5] Tian Bian, Xi Xiao, Tingyang Xu, Peilin Zhao, Wenbing Huang, Yu Rong, and Junzhou Huang. Rumor detection on social media with bi-directional graph convolutional networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(01):549–556, Apr. 2020.

[6] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, page 335–336, New York, NY, USA, 1998. Association for Computing Machinery.

[7] Caleb T. Carr and Rebecca A. Hayes. Social media: Defining, developing, and divining. *Atlantic Journal of Communication*, 23(1):46–65, 2015.

[8] Carlos Castillo, Marcelo Mendoza, and Barbara Poblete. Information credibility on twitter. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, page 675–684, New York, NY, USA, 2011. Association for Computing Machinery.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[10] Tom Diethe, Tom Borchert, Eno Thereska, Borja Balle, and Neil Lawrence. Continual learning in practice, 2019.

[11] Nicholas DiFonzo and Prashant Bordia. Rumor, gossip and urban legends. *Diogenes*, 54(1):19–35, 2007.

[12] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. Virtualization vs containerization to support paas. In *2014 IEEE International Conference on Cloud Engineering*, pages 610–614, 2014.

[13] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1802–1811. PMLR, 09–15 Jun 2019.

[14] Miriam Fernandez and Harith Alani. Online misinformation: Challenges and future directions. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 595–602, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[15] Devndra Ghimire. Comparative study on python web frameworks: Flask and django.

[16] Aditi Gupta, Ponnurangam Kumaraguru, Carlos Castillo, and Patrick Meier. Tweetcred: Real-time credibility assessment of content on twitter. pages 228–243, 11 2014.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[18] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

[19] Benjamin D. Horne, Jeppe Nørregaard, and Sibel Adali. Robust fake news detection over time and attack. *ACM Trans. Intell. Syst. Technol.*, 11(1), dec 2019.

[20] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: Understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, WebKDD/SNA-KDD '07, page 56–65, New York, NY, USA, 2007. Association for Computing Machinery.

[21] Janis Klaise, Arnaud Van Looveren, Clive Cox, Giovanni Vacanti, and Alexandru Coca. Monitoring and explainability of models in production, 2020.

[22] Sachin Kumar, Rohan Asthana, Shashwat Upadhyay, Nidhi Upreti, and Mohammad Akbar. Fake news detection using deep learning models: A novel approach. *Trans. Emerg. Telecommun. Technol.*, 31(2), feb 2020.

[23] Sejeong Kwon, Meeyoung Cha, and Kyomin Jung. Rumor detection over varying time windows. *PLOS ONE*, 12(1):1–19, 01 2017.

[24] Sejeong Kwon, Meeyoung Cha, Kyomin Jung, Wei Chen, and Yajun Wang. Prominent features of rumor propagation in online social media. In *2013 IEEE 13th International Conference on Data Mining*, pages 1103–1108, 2013.

[25] Marc Kydd and Lynsay A. Shepherd. Deep breath: A machine learning browser extension to tackle online misinformation, 2023.

[26] Yiqiao Li, Jianlong Zhou, Sunny Verma, and Fang Chen. A survey of explainable graph neural networks: Taxonomy and evaluation metrics, 2022.

[27] Xiaomo Liu, Armineh Nourbakhsh, Quanzhi Li, Rui Fang, and Sameena Shah. Real-time rumor debunking on twitter. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 1867–1870, New York, NY, USA, 2015. Association for Computing Machinery.

[28] Jesus L. Lobo, Javier Del Ser, Albert Bifet, and Nikola Kasabov. Spiking neural networks and online learning: An overview and perspectives. *Neural Networks*, 121:88–100, 2020.

[29] Jing Ma, Wei Gao, Shafiq Joty, and Kam-Fai Wong. An attention-based rumor detection model with tree-structured recursive neural networks. *ACM Trans. Intell. Syst. Technol.*, 11(4), jun 2020.

[30] Jing Ma, Wei Gao, Prasenjit Mitra, Sejeong Kwon, Jim Jansen, Kam-Fai Wong, and Meeyoung Cha. Detecting rumors from microblogs with recurrent neural networks. 07 2016.

[31] Jing Ma, Wei Gao, Zhongyu Wei, Yueming Lu, and Kam-Fai Wong. Detect rumors using time series of social context information on microblogging websites. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 1751–1754, New York, NY, USA, 2015. Association for Computing Machinery.

[32] Jing Ma, Wei Gao, and Kam-Fai Wong. Detect rumors in microblog posts using propagation structure via kernel learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 708–717, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[33] Jing Ma, Wei Gao, and Kam-Fai Wong. Rumor detection on Twitter with tree-structured recursive neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1980–1989, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[34] Panagiotis Takas Metaxas, Samantha Finn, and Eni Mustafaraj. Using twittertrails.com to investigate rumor propagation. In *Proceedings of the 18th ACM Conference Companion on Computer Supported Cooperative Work & Social Computing*, CSCW'15 Companion, page 69–72, New York, NY, USA, 2015. Association for Computing Machinery.

[35] Soo Jung Moon and Patrick Hadley. Routinizing a new technology in the newsroom: Twitter as a news source in mainstream media. *Journal of Broadcasting & Electronic Media*, 58(2):289–305, 2014.

[36] Andrei Paleyes, Raoul-Gabriel Urma, and Neil D. Lawrence. Challenges in deploying machine learning: A survey of case studies. *ACM Comput. Surv.*, 55(6), dec 2022.

[37] Vahed Qazvinian, Emily Rosengren, Dragomir R. Radev, and Qiaozhu Mei. Rumor has it: Identifying misinformation in microblogs. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1589–1599, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics.

[38] Justus Randolph. Free-marginal multirater kappa (multirater κfree): An alternative to fleiss fixed-marginal multirater kappa. volume 4, 01 2010.

[39] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[40] Paul Resnick, Samuel Carton, Souneil Park, Yuncheng Shen, and Nicole Zeffer. Rumorlens: A system for analyzing the impact of rumors and corrections in social media. 2014.

[41] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification, 2019.

[42] Tom Rosenstiel, Jeff Sonderman, Kevin Loker, Maria Ivancin, and Nina Kjarval. Twitter and the news: How people use the social network to learn about the world. *Online at www. americanpressinstitute. org*, 2015.

[43] Shayan Sardarizadeh. Instagram fact-check: Can a new flagging tool stop fake news? *BBC*, 09 2019.

[44] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[45] Chengcheng Shao, Giovanni Luca Ciampaglia, Alessandro Flammini, and Filippo Menczer. Hoaxy: A platform for tracking online misinformation. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, page 745–750, Republic and Canton of Geneva, CHE, 2016. International World Wide Web Conferences Steering Committee.

[46] Karishma Sharma, Feng Qian, He Jiang, Natali Ruchansky, Ming Zhang, and Yan Liu. Combating fake news: A survey on identification and mitigation techniques. *ACM Trans. Intell. Syst. Technol.*, 10(3), apr 2019.

[47] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and Y. C. Tay. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th*

International Middleware Conference, Middleware '16, New York, NY, USA, 2016. Association for Computing Machinery.

[48] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *SIGKDD Explor. Newsl.*, 19(1):22–36, sep 2017.

[49] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, jan 2014.

[50] Madusha Prasanjith Thilakarathna, Vihanga Ashinsana Wijayasekara, Yasiru Gamage, Kavindi Hanshani Peiris, Chanuka Abeysinghe, Intizar Rafaideen, and Prathieshna Vekneswaran. Hybrid approach and architecture to detect fake news on twitter in real-time using neural networks. In *2020 5th International Conference on Information Technology Research (ICITR)*, pages 1–6, 2020.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[52] Soroush Vosoughi, Deb Roy, and Sinan Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[53] Wikipedia. Pizzagate conspiracy theory — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Pizzagate%20conspiracy%20theory&oldid=1127273007`, 2022. [Online; accessed 16-December-2022].

[54] Ke Wu, Song Yang, and Kenny Q. Zhu. False rumors detection on sina weibo by propagation structures. In *2015 IEEE 31st International Conference on Data Engineering*, pages 651–662, 2015.

[55] Liang Wu, Fred Morstatter, Kathleen M. Carley, and Huan Liu. Misinformation in social media: Definition, manipulation, and detection. *SIGKDD Explor. Newsl.*, 21(2):80–90, nov 2019.

[56] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. Explainable ai: A brief survey on history, research areas, approaches and challenges. In Jie Tang, Min-Yen Kan, Dongyan Zhao, Sujian Li, and Hongying Zan, editors, *Natural Language Processing and Chinese Computing*, pages 563–574, Cham, 2019. Springer International Publishing.

[57] Fan Yang, Yang Liu, Xiaohui Yu, and Min Yang. Automatic detection of rumor on sina weibo. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, New York, NY, USA, 2012. Association for Computing Machinery.

[58] Hao Yuan, Haiyang Yu, Shurui Gui, and Shuiwang Ji. Explainability in graph neural networks: A taxonomic survey, 2022.

[59] Zhe Zhao, Paul Resnick, and Qiaozhu Mei. Enquiring minds: Early detection of rumors in social media from enquiry posts. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1395–1405, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[60] Arkaitz Zubiaga, Maria Liakata, and Rob Procter. Exploiting context for rumour detection in social media. In Giovanni Luca Ciampaglia, Afra Mashhadi, and Taha Yasseri, editors, *Social Informatics*, pages 109–123, Cham, 2017. Springer International Publishing.

Appendix A

Additional Content

A.1 Algorithms

```
    Input: A tweet ID as a String.
    Output: A list containing all the tweets in the tweet cascade.
main fetch_tweets_for_cascade(root_tweet_id):
    all_replies = fetch_replies(root_tweet_id)
    if all_replies is empty then
        Fail
    all_quotes = fetch_quotes(root_tweet_id)
    if all_quotes is empty then
        Fail
    return all_replies + all_quotes
replies fetch_replies(root_tweet_id):
    all_replies = [ ]
    replies_content = fetch_replies_from_api(root_tweet_id)
    if Error in replies_content then
        Fail
    while replies_content not empty or error do
        sort replies_content chronologically
        all_replies.extend(replies_content)
        oldest_id = replies_content.pop(0)
        replies_content = fetch_replies_from_api(root_tweet_id, oldest_id)
    end
    return all_replies
replies fetch_quotes(root_tweet_id):
    all_quotes = [ ]
    quote_content, next_token = fetch_quotes_from_api(root_tweet_id)
    if Error in quote_content then
        Fail
    while next_token not empty or error do
        sort quotes_content chronologically
        all_quotes.extend(replies_content)
        oldest_id = quotes_content.pop(0)
        quotes_content, next_token = fetch_quotes_from_api(root_tweet_id, next_token
    end
    return all_quotes
```

Algorithm 1: Generating a tweet cascade for a given source tweet.

A.2 Figures

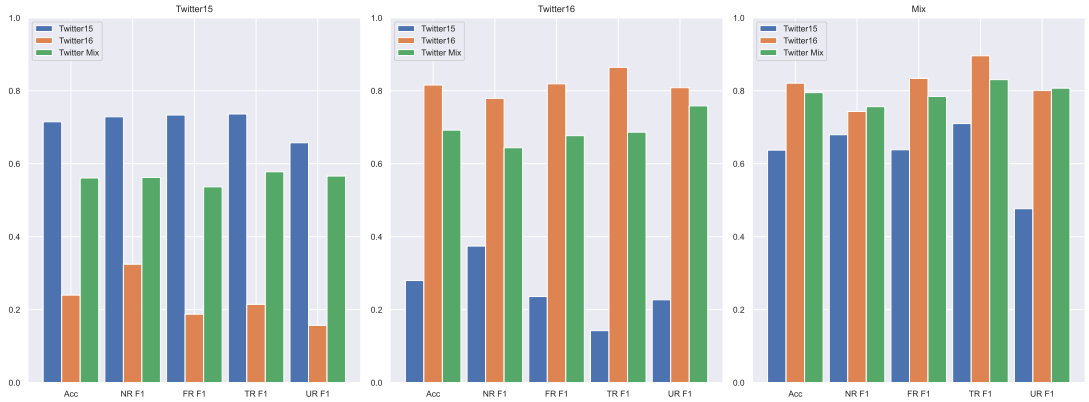


Figure A.1: Result from the dataset mixing experiment for the Twitter15 and Twitter16 models, with $p_{T15} = 0.3$ and $p_{T16} = 0.7$.

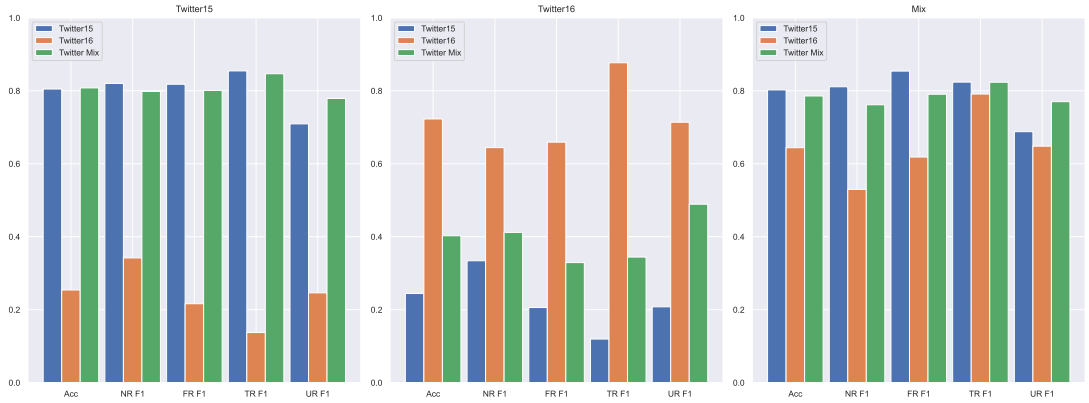


Figure A.2: Result from the dataset mixing experiment for the Twitter15 and Twitter16 models, with $p_{T15} = 0.7$ and $p_{T16} = 0.3$.

Appendix B

Participants' information sheet

This study was certified according to the Informatics Research Ethics Process, RT number 2023/260884. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

Andrej Jovanovic - A fourth year student at the University of Edinburgh, studying Artificial Intelligence and Computer Science. He is completing his fourth year dissertation, titled "Rumour Detection in the Wild: A Browser Extension for Twitter", under the supervision of Dr Björn Ross,

Dr Björn Ross - Lecturer (Assistant Professor) in Computational Social Science at the University of Edinburgh School of Informatics, in the Institute for Language, Cognition and Computation.

What is the purpose of the study?

The goal of this study is to assess the quality of the browser extension that I am creating as a part of my fourth year dissertation. Said browser extension is used to detect rumours on Twitter. Human participants in the user study will assess the quality of the browser extensions after interacting with it, answering questions such as: "would you recommend this extension to a friend?" or "would you use this extension again?". Data analysis will be minimal; it will simply be used to determine the success of the browser extension based on the users' responses. The results of this study could be used to improve the front-end features of the browser extension, if such suggestions are given, and to guide any recommendations for future work.

Why have I been asked to take part?

Research target group are Twitter users, that have Google Chrome installed on their desktops, that are interested in detecting rumours when they interact with the social media platform.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study

at any time, up until your responses are submitted, without giving a reason. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

You will interact with a browser extension that is used to detect rumours on Twitter. Using this extension, you will be asked to detect rumours on pre-specified tweets. Once you have finished using the extension, you will be asked to complete a survey, indicating whether you thought certain tweets were rumours or not. Furthermore, you will be asked to answer questions, indicating your response on a Likert scale supported by additional information in a text box if necessary. Questions will include "would you recommend this browser extension to a friend?. This will be a once-off survey that will require 15–20 minutes of your time.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

No

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a maximum of 4 years. All potentially identifiable data will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team: Dr Björn Ross and Andrej Jovanovic.

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. For more details, including the right to lodge

a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Andrej Jovanovic - s1900682@ed.ac.uk If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Andrej Jovanovic - s1900682@ed.ac.uk

General information

For general information about how we use your data, go to: edin.ac/privacy-research

Appendix C

Participants' consent form

Consent was collected through the Google Form immediately before the user study was provided to the participants. In order to proceed to the user study, participants were required to indicate their consent to the following statements:

- I allow my data to be used in future ethically approved research.
- I agree to take part in this study.

Appendix D

User Study Questionnaire

The participants' information sheet and consent forms were included in the Google Form prior to the user study questions.

D.1 Installing the Browser Extension

Before we begin the user case study, I will walk you through how to download the Google Chrome extension for yourselves.

Step 1 - Installing Google Chrome

If you have Google Chrome installed, you may proceed to step 2.

Please download Google Chrome at the following URL address.

Once you have successfully installed Google Chrome, please proceed to step 2.

Step 2 - Installing the Browser Extension

Once you have downloaded Google Chrome, please download the Twitter Rumour Extension at the following link.

Please ensure that you open this link in Google Chrome, and not in your default web browser.

1. Once you have accessed the link, click "Add to Chrome".
2. Confirm that you wish to add the browser extension.
3. The browser extension is now installed. Now we need to pin it to our toolbar. Click the puzzle piece in the top right-hand corner of your toolbar. There, you should see "Twitter Rumour Detection" with a pin besides it. Click it. This has now added the extension to your toolbar.

Congratulations! You have successfully added the browser extension to your toolbar.

Step 3 - Setting up the server

In order to start the server for this service, please visit the following link. Your connection will seem to ”hang”, as if you have no internet connection. This is normal - the server is just booting up after some well-deserved rest. When you are greeted with the welcome message, you have successfully booted the server!

Final Step - Testing the Browser Extension.

Once you have completed all the above steps, it is time to test out the extension! Please follow the following link to access a tweet: Please ensure that you have opened the link in Google Chrome, and not in your default web browser.

1. Open the browser extension in the top right-hand corner of your toolbar.
2. Click the button ”Detect rumour”
3. Click ”OK” on the alert that pops-up. This is meant to show you that the extension has successfully extracted the tweet ID you are wanting to use.

Voilà! You should be greeted with a classification and if you scroll down, a list of relevant news articles to the tweet. Congratulations on making it here!

That is all the setup done! See you in the next section.

Were you able to successfully install the extension?

- Yes
- No

D.2 User Case Study

For the user case study, I will ask you to preform rumour detection on Twitter at five predefined URLs. The task is as follows. For each tweet, you will need to:

1. Open the tweet in Google Chrome
2. Prior to preforming rumour detection using the browser extension, comment on whether you believe the tweet contains a rumour or not.
3. Preform rumour detection using the browser extension.
4. Comment on whether the browser extension helped you in determining the rumour status of the tweet, or was helpful in providing more context to the tweet itself.

Once you have performed rumour detection on all the tweets, there will be some more general questions associated to your experience using the browser extension.

NB: If you attempt to preform rumour detection on other tweets, you will receive an error.

Please ensure that you open these links in Google Chrome so that you are able to detect the rumours

D.2.1 Tweet 1

Link to tweet.

Tweet 1: Before performing rumour detection, did you have any reason to believe that this tweet contained a rumour?

- Yes
- No

Tweet 1: After performing rumour detection, did the extension support your experience in any way? From flagging up potential rumours, to providing informative context.

- Yes
- No

Tweet 1: Please add any comments related to the above two questions, if any.

D.2.2 Tweet 2

Link to tweet.

Tweet 2: Before performing rumour detection, did you have any reason to believe that this tweet contained a rumour?

- Yes
- No

Tweet 2: After performing rumour detection, did the extension support your experience in any way? From flagging up potential rumours, to providing informative context.

- Yes
- No

Tweet 2: Please add any comments related to the above two questions, if any.

D.2.3 Tweet 3

Link to tweet.

Tweet 3: Before performing rumour detection, did you have any reason to believe that this tweet contained a rumour?

- Yes
- No

Tweet 3: After performing rumour detection, did the extension support your experience in any way? From flagging up potential rumours, to providing informative context.

- Yes
- No

Tweet 3: Please add any comments related to the above two questions, if any.

D.2.4 Tweet 4

Link to tweet.

Tweet 4: Before performing rumour detection, did you have any reason to believe that this tweet contained a rumour?

- Yes
- No

Tweet 4: After performing rumour detection, did the extension support your experience in any way? From flagging up potential rumours, to providing informative context.

- Yes
- No

Tweet 4: Please add any comments related to the above two questions, if any.

D.2.5 Tweet 5

Link to tweet.

Tweet 5: Before performing rumour detection, did you have any reason to believe that this tweet contained a rumour?

- Yes
- No

Tweet 5: After performing rumour detection, did the extension support your experience in any way? From flagging up potential rumours, to providing informative context.

- Yes
- No

Tweet 5: Please add any comments related to the above two questions, if any.

D.2.6 Global Questions

I would use this browser extension again in my personal time.

- Strongly disagree
- Disagree
- Neither agree nor disagree
- Agree
- Strongly agree

I would recommend this browser extension to my friends and family.

- Strongly disagree
- Disagree
- Neither agree nor disagree
- Agree
- Strongly agree

I found this browser extension to be very useful.

- Strongly disagree
- Disagree
- Neither agree nor disagree
- Agree
- Strongly agree

What suggestions do you have to improve the browser extension, if any?

Can you recall a time when you had wished that you had access to such a tool as this rumour detection extension for Twitter? If so, what was that scenario, and how would have an additional aided your experience? (NB: This does not have to be restricted to rumour detection on Twitter).

D.3 Deleting the Browser Extension

Thank you for completing the user study. Your effort is much appreciated.

In order to finish this user study, please delete the browser extension from your extensions list.

In order to do so, please:

1. Click the puzzle piece in the top right-hand corner of your Google Chrome toolbar.
2. Click the ”Manage extensions” button.
3. Find the ”Twitter Rumour Detection” browser extension and click ”Remove”.
4. Confirm that you wish to remove it.
5. Voilà. We are all done!

Did you manage to remove the browser extension?

- Yes
- No

D.4 Difficulty Installing the Browser Extension

Why were you unable to install the browser extension?