# [CM3070]
# Final Project

# Preliminary Report

Yau Feng Yuen, Gabriel

# [1.0] Introduction

## [1.1] Template

[CM3060] Natural Language Processing – "Fake News Detection"

## [1.2] Project Overview

The spread of misinformation in the modern era presents significant and multifaceted challenges. Fake news and its ability to rapidly propagate falsehoods and distort perceptions has eroded trust, destabilised institutions, and undermined democratic processes.

This project aims to address this issue by leveraging Natural Language Processing (NLP) techniques to design a scalable, accurate & user-friendly fake news detection system that will empower a diverse range of primary users – including journalists and educators – to swiftly and reliably access the credibility of news they consume.

# [1.3] Motivation

This project is motivated by the need to address the real-world consequences of fake news that span several domains:

1. **Erosion of trust in journalism**

   The increasing prevalence of fake news undermines public confidence in legitimate news sources. Trustworthy journalism is vital for an informed society, and the erosion of trust in journalism polarises communities, fosters hostility and impedes constructive discourse.

2. **Economic disruptions**

   Fake news can manipulate markets and harm businesses. For instance, baseless rumours about a company's fiscal health can lead to a sudden crash in stock prices, unfairly & disproportionately affecting vulnerable stakeholders, retail investors and small businesses.

3. **Public health risks**

   The COVID-19 pandemic highlighted how misinformation can incite panic – for example, in Singapore, misinformation about supply shortages prompted the local populace to hoard masks and staple foods unnecessarily, straining supply chains and increasing social anxiety unnecessarily.

4. **Political impact**

   Disinformation campaigns are often weaponised to distort public opinion, promote political agendas and undermine elections. These campaigns erode the public's faith in governance.

With the increasing volume of misinformation and the speed of which it is spread, the demand for tools to quickly and effectively verify the credibility of digital news is higher than ever before. This project is tailored to address the distinct needs of two primary user groups:

1. **Journalists**

   Journalists need tools to verify the credibility of sources and combat the spread of false information. An effective solution will  improve the credibility and impact of journalistic work in safeguarding the integrity of public discourse.

2. **Educators**

   Educators need tools to empower individuals to navigate constantly-evolving, complex digital landscapes by enhancing media literacy and critical thinking ability.

# [1.4] Related Projects

## [1.4.1] DataFlair: Detecting Fake News with Python and Machine Learning

DataFlair uses a passive-aggressive classifier to train a model on a small dataset of 7796 rows. [1]

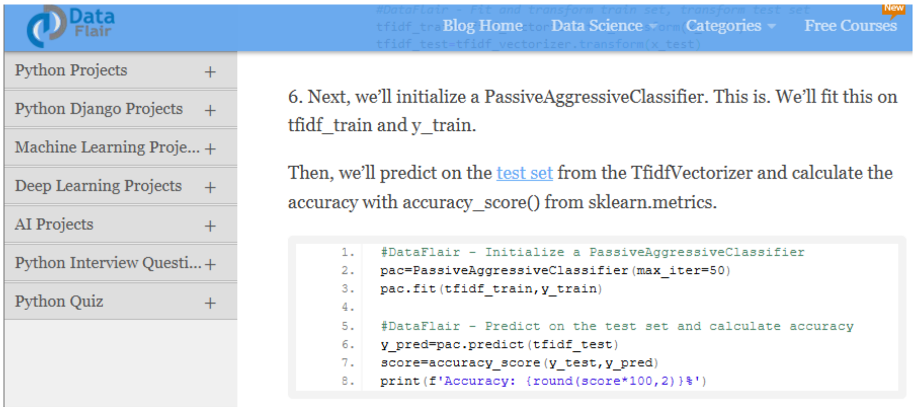



*Figure 1: Screenshot of project tutorial on DataFlair [1]*

While this project serves as an excellent technical demonstration for beginners, the pipeline demonstrated in this project is too simple for deployment in real-world applications. The model is also trained on an extremely small dataset, which further limits its ability to generalise to real-world applications.

As such, DataFlair's project emphasises the need for a solution that leverages state-of-the-art technologies, and is trained on larger datasets that would not inherently inhibit the model's ability to generalise.

# [1.4.2] ClaimBuster

ClaimBuster applies Natural Language Processing techniques to identify and evaluate factual claims primarily within political discourse.

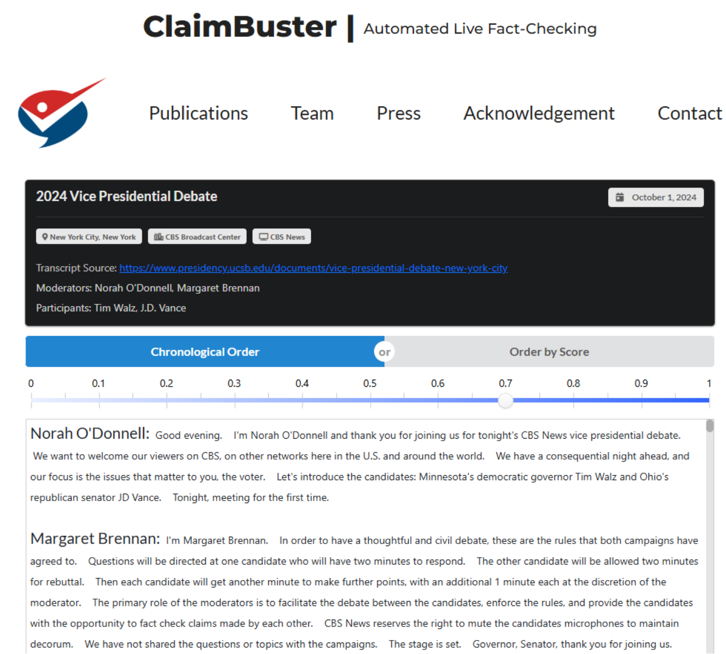


*Figure 2: Screenshot of ClaimBuster's user interface [2]*

While ClaimBuster is effective in detecting factual claims, its narrow focus on political contexts limits its applicability to broader domains, underscoringhe need for a solution that is generalisable and scalable across various topics and misinformation domains.

### [1.4.3] Snopes

Snopes is a widely-recognised platform for human fact-checking that relies on human subject-matter experts to manually verify claims, evaluate sources, and analyse misinformation across various topics.

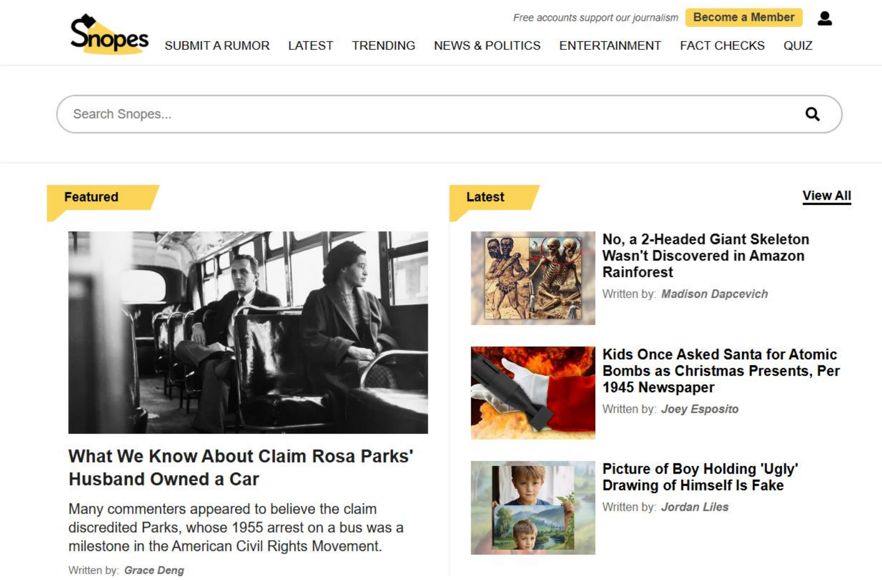



*Figure 3: Screenshot of Snopes' homepage [3]*

While Snopes is highly credible and thorough, its manual nature is inherently slow and not scalable for real-time verification tasks. Thus, its labor-intensive nature highlights the need for algorithmic alternatives that can complement and augment human expertise.

# [1.5] Project Aims

This project aims to bridge the identified gaps by introducing a comprehensive solution with the following core contributions:

1. Scalability through transformer-based models

   This project aims to address the limitations of human-intensive methodologies by leveraging cutting-edge machine learning & deep learning techniques that are adaptable to high volumes of information, such as transformer models (e.g. BERT).

2. Cross-domain generalisability

   This project will be designed to detect misinformation over a wide range of topics and formats, to ensure its relevance and utility in addressing fake news.

# [2.0] Literature Review

Detecting and mitigating the spread of fake news has become a critical area of research within the discipline of natural language processing (NLP). This literature review examines existing approaches and technologies in fake news detection with a focus on their relevance to their implications on the design and implementation of my solution.

# [2.1] Machine Learning Approaches

Several projects have tackled the challenge of fake news detection with machine learning approaches.

## [2.11] Passive-Aggressive Classifier

The project brief included an example project by DataFlair that uses a small dataset of 7796 rows with news articles labelled as true/fake. TF-IDF vectorisation is first performed to pre-process text data, then a Passive-Aggressive Classifier is applied to train a machine learning model. [1]

This resource serves as an excellent technical demonstration for beginners as it introduces & explains the fundamental steps of data pre-processing, feature extraction, model training, & pipeline building.

However, this resource has too many limitations:

1. **The dataset is not representative of real-world scenarios** – it is too small and lacks variety, thus limiting the model's ability to generalise effectively.

2. **The model is excessively simplistic** – Passive-Aggressive Classifiers are easy to implement but is inadequate to handle nuanced language patterns & context in fake news, and lacks the sophistication of modern NLP models. This is further evidenced by Chang's (2024) comparison of machine learning & deep learning algorithms, where Passive-Aggressive Classifiers were ranked 7th in fake news detection performance compared to other algorithms and outperformed by all deep learning algorithms [4].

These limitations restrict the model's ability to handle real-time detection tasks or large-scale datasets and thus is not suitable for deployment in real-world settings.

## [2.12] Random Forest Classifier

Random forest classifiers aggregate multiple decision trees and have been widely employed in fake news detection research.

A potential problem with Random Forest Classifiers is that as their performance declines on imbalanced datasets. Huh (2021) demonstrated that standard Random Forest classifiers achieved a relatively high false-negative rate, a recall of 0.102 and precision of 0.365. This illustrates that 36.5% of predictions in the minority class are correct, and that standard Random Forest classifiers failed to predict approximately 90% of the minority class when presented with an imbalanced dataset. [5]

While it is possible to improve the performance of Random Forest Classifiers when dealing with imbalanced datasets, in Chang's 2024 comparison of ML and DL algorithms, random forest classifiers ranked 9$^{th}$, outperformed by all tested all deep learning algorithms. [4]

# [2.2] BERT

BERT (Bidirectional Encoder Representations from Transformers), first introduced by Devlin et al. (2019), is designed to pre-train deep bidirectional representations from unlabelled text by jointly conditioning on both left and right context in all layers [6], allowing BERT models to effectively model contextual relationships in textual data and discern nuanced linguistic patterns.

BERT's capabilities make it very suitable for fake news detection:

1. **BERT outperforms other algorithms.**

   Chang's (2024) comparison of machine learning & deep learning algorithms has shown that BERT is the highest-performing algorithm amongst those tested, with unprecedented scores of 99.95% accuracy, precision, recall & F1 score. [4]

2. **BERT is extremely flexible.**

   BERT's transformer architecture allows fine-tuning for task-specific applications, improving domain adaptability.

BERT models, however, have their limitations:

1. **BERT has high computational demands.**

   Transformer architectures like BERT have high computational demands. BERT's pre-training corpus included Google's BookCorpus and English Wikipedia, comprising 800 million and 2500 million words respectively for a total of 3300 million words [6].

   However, innovations such as MobileBERT, a thinner model that requires less computational power [7], have surfaced to meet the rising demand to leverage BERT's capabilities within smaller computational environments.

2. **BERT (and other transformer models) are inherently un-interpretable.**

   Korolev et. al (2023) asserts that state-of-the-art models such as BERT are highly parameterised black boxes [8]. The opacity of transformer-based models has driven interest & demand in explainable AI techniques such as attention visualisation.

Thus, BERT's unparalleled performance establishes it as a cornerstone for this project. However, in implementing BERT, it is important to address scalability concerns through the implementation of lighter variants such as MobileBERT that are less computationally intensive, and to address interpretability challenges by incorporating explainable AI techniques to foster user trust in the model.

# [2.3] Hybrid Approaches

Hybrid approaches to fake news detection typically combine content-based and context-based analyses. They often integrate multimodal data – such as text, images, social network metadata – and utilise diverse sources of information to enhance their accuracy and performance.

Hybrid approaches aim to provide a more comprehensive perspective on misinformation propagation through the integration of complimentary data streams.

# [2.31] ClaimBuster

ClaimBuster is a web-based, automated, live fact-checking tool developed by the University of Texas [9].

Hassan et al. (2017) detailed how ClaimBuster works in their published paper "ClaimBuster: The first-ever end-to-end fact-checking system". [10]

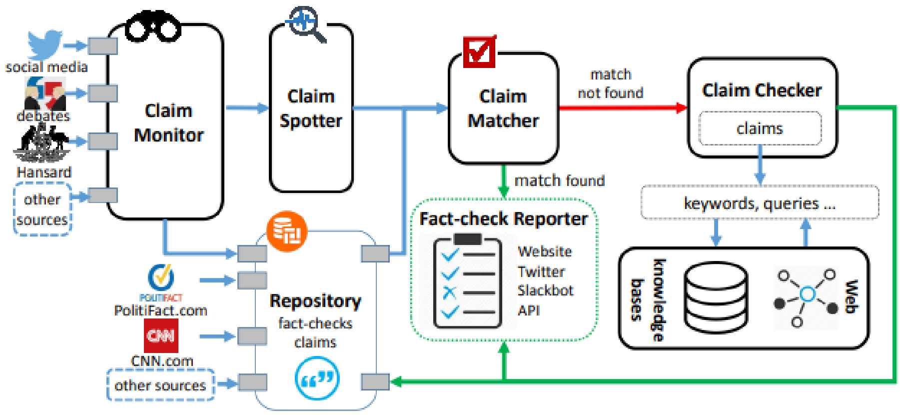



*Figure 4: ClaimBuster's system architecture [10]*

ClaimBuster implements:

1. A **'claim monitor'** that continuously monitors and retrieves texts from various sources including broadcast media through a decoding device to extract closed captions, social media, and websites (such as the transcripts of Australian parliament proceedings). [10]

2. A **'claim spotter'** that scores sentences' likelihood of containing a factual claim using a classification & scoring model. [10]

3. A **'claim matcher'** takes an important factual claim and searches a fact-check repository and returns fact-checks matching the claim. If a matching fact-check cannot be found, the claim checker queries external knowledge bases and the Web with a question generation tool, before a fact-check reporter finally delivers a report to users through the project website. [10]

ClaimBuster is effective in structured environments such as political debate analysis, where domain-specific knowledge bases facilitate precise fact-checking. Additionally, as pre-verified claims are combined with linguistic patterns, ClaimBuster offers a high degree of accuracy.

However, due to the manual effort required for database curation, scalability of ClaimBuster is constrained, and its applicability in rapidly-evolving contexts may thus be limited. Additionally, as it relies on domain-specific datasets, its ability to adapt to diverse misinformation scenarios may be restricted.

# [2.32] FaKnow

FaKnow is a library designed to standardise the development & evaluation of fake news detection algorithms by integrating various fake news detection algorithms. [11] The library includes a variety of widely-used models, categorised into content-based & social context-based approaches. Additionally, it also offers functionalities for data processing, model training, evaluation, visualisation and logging to enhance reproducibility and reduce redundancy in fake news detection research.

Graves and Cherubini (2016) emphasise that reproducibility is a persistent challenge in misinformation research due to the lack of standardised datasets and frameworks [12]. FaKnow addresses this critical issue by standardising the implementation of various fake news detection algorithms. Additionally, by encompassing both content-based and context-based models, researchers can use FaKnow to explore & evaluate various different approaches within a single platform.

However, FaKnow has several limitations within the context of my project:

1. **PyTorch framework dependency**

   As FaKnow is built on PyTorch, researchers using other deep learning frameworks (e.g. TensorFlow) may face challenges integrating existing models/workflows to this library. Dependency on a single framework may limit adoption amongst researchers with established preferences or institutional constraints.

2. **Continuous maintenance demand**

   As fake news detection algorithms rapidly advance, continuous updates to the library are required to incorporate the latest models and techniques. This presents maintenance challenges for both users and developers of FaKnow.

Ultimately, while FaKnow presents limitations in its PyTorch dependency and continuous maintenance demand to stay abreast of the latest advancements, its strengths in standardisation and usability make it a valuable tool.

# [2.4] Human-operated Fact Checking Websites

Human-operated fact checking websites such as PolitiFact, FactCheck.org, and Snopes, play a critical role in combating misinformation by employing teams of human experts to evaluate the truthfulness of news content and factual claims.

These websites have earned a wealth of perceived credibility and user trust in two ways:

1. **The rigorous methodologies that human fact-checkers employ ensure credibility due to the high accuracy in identifying and categorising false information.**

   For example, PolitiFact displays a "Truth-O-Meter" that provides a granular evaluation of claims which serves to aid public understanding of misinformation.

2. **These websites provide detailed context and explanations for their evaluations**.

   This in turn promotes media literacy and critical thinking amongst users, but also transparency which helps to foster user trust.

Liu et al. (2023) assert that users "doubt the ability of machines to adjudicate factual disputes and thus perceive AI-based fact-checkers as less credible than human-based fact-checking services". [13] Yang et al. argue that this is because automated methods are error-prone [14].

However, human fact-checking also has inherent limitations:

1. **Human fact-checking is not scalable.** Lin et al. (2023) assert that traditional manual fact-checking is time-consuming and labor-extensive, which cannot scale with the unprecedented amount of disinformation and misinformation on social media [15]. This limits their impact in countering real-time viral narratives.

2. **Perceived (or actual) biases in the evaluation of factual claims can undermine user trust**, especially amongst ideologically polarised audiences.

Ultimately, this is relevant to my solution as insights from the strengths of human-operated systems can help to inform the integration of human-like explainability features to foster user trust.

# [2.5] Evaluation

| Method/Approach | Strengths | Weaknesses | Example Use Case |
|---|---|---|---|
| Passive-Aggressive Classifier (ML) | - Simple implementation | - Limited generalisability<br>- Does not capture nuance | - DataFlair example project |
| Random Forest Classifier (ML) | - Robust to overfitting | - Poor performance on imbalanced datasets | - Academic experiments |
| BERT (Deep Learning) | - State-of-the-art contextual understanding | - High computational demand<br>- Lack of interpretability | - Cutting-edge NLP tasks |
| ClaimBuster | - High accuracy in structured contexts | - Limited scalability<br>- Limited domain applicability | - Political debate fact-checking |
| FaKnow | - Standardised evaluation<br>- Reproducible | - Requires continuous maintenance/updates<br>- PyTorch dependency limits flexibility | - Academic research |
| Human fact-checking | - High credibility and user trust | - Labour-intensive<br>- Not scalable | - Snopes, PolitiFact |

*Figure 5: Tabular summary of fake news detection methods*

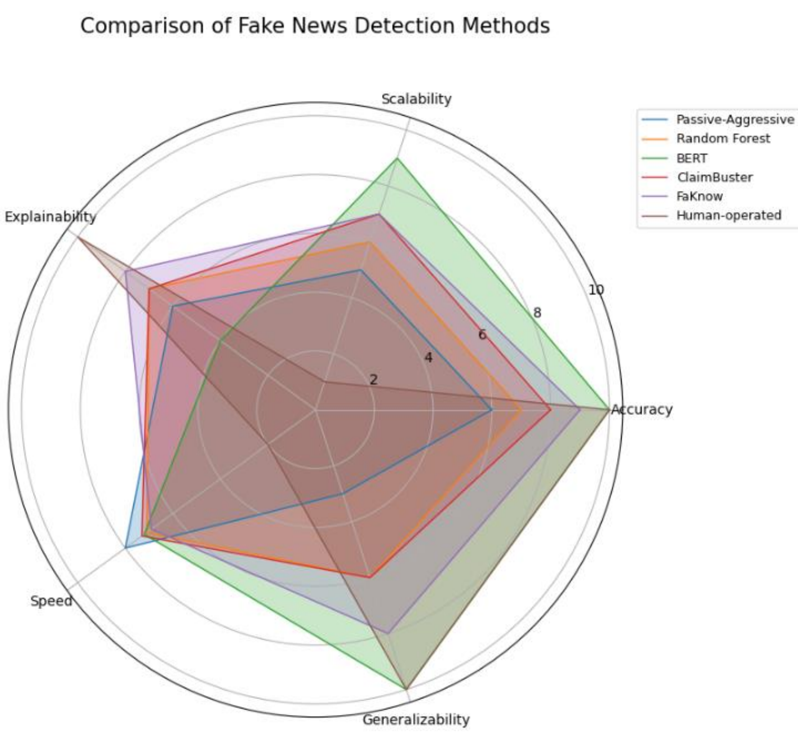



*Figure 6: Radar chart comparing key attributes of fake news detection methods*

My review of existing methodologies highlights several critical areas for innovation:

1. **Scalability**

   My solution should address the scalability limitations of human fact-checking that drives demand for algorithmic fact-checking in the first place, through incorporating optimising transformer architectures (i.e. BERT).

2. **Transparency**

   To foster user trust & credibility, the development of interpretable AI methods to address the "black box" nature of existing models is crucial. Ideally, these methods would be more human-like, such as through short paragraphs as opposed to attention heatmaps.

3. **Cross-domain applicability**

   My solution should be generalisable to address misinformation across various topics and formats.

# [3.0] Design

# [3.1] Project Overview

Project Template: [CM3060] Fake News Detection

This project aims to address the issue of misinformation by designing an AI-powered Fake News Detection system that will leverage advanced Natural Language Processing (NLP) techniques to classify user-inputted articles as fake or real. This will be delivered as a Command-Line Interface (CLI) application that will empower users to verify the credibility of online content in real-time.

To achieve that goal, the primary objective is to build a data pipeline to process and link data effectively, so it can be coupled with a query generation strategy that will deliver on accuracy.

# [3.2] Domain & Users

## [3.21] Domain

This project falls under the domain of information verification within the field of journalism & digital media.

The challenges this domain addresses include:

- identifying misinformation,
- improving public media literacy, and
- mitigating the societal impact of fake news.

# [3.22] Users

My proposed solution is built with the view that a diverse group spanning various demographics would find value in this solution.

These users would all have varied requirements and use cases, and as such I have elected to focus on groups of primary users:

1. **Journalists**

   My proposed solution will create value for journalists. Their demographic profile spans ages 25 ro 60, with a media scope that includes traditional newspapers, digital platforms, freelance journalism & multimedia reporting.

   Journalists operating in different regions will have different regional challenges. For example, journalists in the Middle East face misinformation challenges addressing religious & ethnic sensitivities, increasing the need for source validation tools, whereas Western journalists struggle to target digital disinformation – specifically, journalists in democracies such as the United States have challenges combating election-related fake news.

   Journalists would use my solution to:
   - fact-check sources,
   - build credibility in reporting, and,
   - investigate information cascades.

   Meanwhile, there are journalists that will not use my application. One such sub-demographic of journalists exists in highly state-controlled environments, such as North Korea, where independent news verification is not feasible or allowed. Additionally, journalists involved in creating or propagating fake news are certainly not going to use nor promote a fake news detection solution.

   With all the above factors considered, I expect journalists to be primary users and adopters of my solution as they have a direct need for tools that aid in fact-checking and verifying sources in their everyday work. Given the immediacy of their nature of work, they require efficient & accurate solutions.

2. **Educators**

Educators that are primarily based in academia or non-profit organisations would gain value from my solution, and are likely to heavily use and/or promote it.

Media literacy educators would be foremost adopters amongst this demographic, as they are responsible for equipping students & communities with tools to identify misinformation. They would promote adoption of the solution by recommending (and thus, marketing) it to students and workshop attendees alike.

That said, adoption of my solution would not be exclusive to media literacy educators. Educators spanning various disciplines, teaching various subjects to various demographics of students (ranging from high school to university-level students) have an interest in teaching students to identify fake news when consuming media, and they would also promote adoption of my solution to their students.

Not all educators should be expected to promote or adopt my solution. Educators in some disciplines, such as early childhood education, for instance, would face different challenges and their lesson plans would naturally have very different learning outcomes that would not include discerning news consumption.

As educators act as 'force multipliers' by teaching communities of students & professionals to critically evaluate news, they play a vital role in adopting and promoting my solution and are thus expected to be primary users & adopters of my solution.

On top of the primary users, I have identified groups of secondary users who are likely to adopt my application.

3. **Researchers (secondary users)**

Typically aged 30-60, researchers, often affiliated with universities or think tanks, often operate in data-rich environments with access to large datasets, computational resources and tools for statistical and NLP analysis.

While researchers - especially those investigating the spread & impact of misinformation, the role of algorithmic amplification, and its societal implications - will have a strong interest in analysing misinformation patterns, they are expected to be secondary users as their use of a CLI application may be occasional or project-specific rather than daily (as expected of primary users such as journalists).

Additionally, as their use case relates to academic analysis, tailoring tools to focus on academic rigor and reproducibility requires the implementation of advanced customisation ability that, if included, would make the solution excessively complex for primary users.

### 4. Content moderators for social media platforms (secondary users)

Content moderators working for social media platforms are tasked with identifying and removing misinformation from their platforms. They play a critical role in curbing the spread of fake news. Demographically, they tend to be professionals in their 20s to 40s, often with backgrounds in communication or technology.

Content moderators often handle large volumes of flagged content daily under strict organisational guidelines. They are secondary users as they already utilise other tools primarily, usually GUI-based systems integrated into a larger moderation platform. The proposed solution may be added to their workflow as a supplemental tool for cases where deeper analysis is required.

# [3.3] Justification for Design Choices

The design choices are meticulously aligned with user needs, and the demands of the domain.

## [3.31] Command Line Interface Application

Command-line interface (CLI) applications are lightweight and platform-independent, meaning that the application can operate seamlessly across various operating systems, ensuring broad accessibility for technically-proficient users.

While a CLI application is not ideal for non-technical users, the primary and secondary users of this application are expected to be technically proficient, and either already experienced with CLI or are easily teachable.

However, a possible avenue for future work is the inclusion of a Graphical User Interface (GUI) to cater to more audiences.

## [3.32] Bidirectional Representations from Transformers (BERT)

BERT is chosen as the machine learning model for several reasons – firstly, its ability to capture nuanced word meanings based on context will enhance classification accuracy for complex, ambiguous tests.

Additionally, BERT is expected to have high domain adaptability – fine-tuning BERT on a domain-specific dataset is expected to increase performance in fake news detection.

Benchmarks consistently demonstrate BERT's superiority over traditional NLP models, indicating it is likely the optimal model to implement in my solution. However, its high computational demands and lack of inherent interpretability needs to be addressed through using lightweight variants (e.g. MobileBERT) and explainable AI tools respectively.

# [3.4] Overall Structure

The project's architecture comprises the following layers:

1. **Data Layer**

The data layer incorporates datasets from multiple sources such as labelled misinformation repositories and domain-specific datasets. Structured storage mechanisms (e.g. SQLite) will be used to manage raw data and processed features for easy access & reproducibility.

The pre-processing pipeline is also implemented here – with tokenisation, stopword removal, stemming & lemmatisation, and feature engineering (TF-IDF vectorisation & embeddings) to transform raw data to model-ready input.

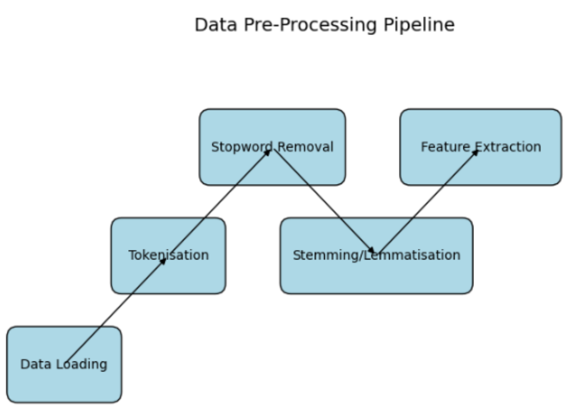



*Figure 7: Data pre-processing pipeline in data layer*

2. **Modelling layer**

The modelling layer comprises traditional machine learning models (e.g. logistic regression, SVM) for comparative benchmarking, as well as deep learning models (i.e. BERT) fine-tuned for fake news detection. These models are combined to improve overall accuracy and robustness.

Additionally, this layer also incorporates additional NLP techniques such as attention mechanisms to enhance context understanding.

3. **Application layer**

The application layer comprises the Command-Line Interface (CLI) to accept user inputs, process articles, and deliver results in real-time.

Additionally, it will also integrate modules like SHAP (Shapley Additive exPlanations) and/or LIME (Local Interpretable Model-Agnostic Explanations) for transparency. As these modules provide users with insights into model decisions, the added transparency enhances user trust and interpretability.

### 4. Evaluation & monitoring layer

This layer comprises the evaluation framework that will track performance metrics (i.e. precision, recall, F1 score, ROC-AUC) during training and deployment phases.

This layer will also include error analysis tools to identify common misclassifications and potential biases to guide iterative improvements, and tools to log system performance metrics to ensure reliability and scalability.

### 5. Deployment layer

The solution will be packaged into a deployable Python package, with dependencies managed through Conda.

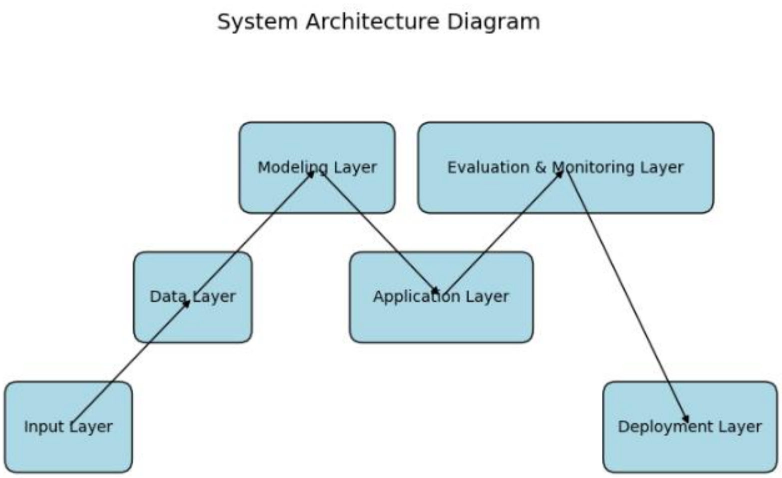



*Figure 8: System architecture diagram*

# [3.5] Technologies & Methods

| | |
|---|---|
| **Programming Language** | Python |
| **Key Libraries** | **Scikit-learn** – tools for traditional machine learning<br>**Transformers (Hugging Face)** – fine-tuning BERT for contextual NLP tasks<br>**Pandas** – for data cleaning & manipulation<br>**NumPy** – for numerical computation<br>**Matplotlib** – for data visualisation<br>**Click** – to simplify CLI creation & improve user interaction |
| **Evaluation Techniques** | Cross-validation<br>ROC curves<br>SHAP/LIME for explainability |

*Figure 9: Table of technologies & methods used*

# [3.6] Work Plan

## Milestones

| This column should be ordered sequentially. | The position column, charts milestones within the task chart. | Enter the date for a milestone in this column. | Enter a milestone description in this column. These descriptions will appear in the chart. |
|---|---|---|---|
| **No.** | **Position** | **Date** | **Milestone** |
| 1 | 1 | 16/12/2024 | Preliminary Report Submission |
| 2 | 2 | 7/1/2025 | Jan: Monthly Check In (approx) |
| 3 | 3 | 27/1/2025 | Draft Report Submission |
| 4 | 4 | 10/2/2025 | Feb: Monthly Check In (approx) |
| 4 | 4 | 3/3/2025 | Mar: Monthly Check In (approx) |
| 4 | 4 | 10/3/2025 | Written Examination |
| 5 | 5 | 24/3/2025 | Final Report Submission |

*Figure 10: Project milestones*

## Tasks

| This column should be ordered sequentially. | Enter the start date for each task below. For best results sort this column in ascending order. | Enter the end date for each task or activity below, in this column. | Enter tasks and/or activities in this column. |
|---|---|---|---|
| **No.** | **Start Date** | **End Date** | **Task** |
| 1 | 16/12/2024 | 16/12/2024 | Preliminary Report Submission |
| 2 | 17/12/2024 | 13/1/2025 | Dataset preparation & pre-processing pipeline |
| 3 | 2/1/2025 | 13/1/2025 | Refine project scope & finalise methodologies based on supervisor and grader feedback |
| 4 | 8/1/2025 | 26/1/2025 | Implement all models |
| 5 | 15/1/2025 | 16/1/2025 | Evaluate models using small test data subset |
| 6 | 16/1/2025 | 23/1/2025 | Refine models |
| 7 | 1/1/2025 | 16/1/2025 | Draft 1: Preliminary Report |
| 8 | 24/1/2025 | 9/2/2025 | Continue refining models |
| 9 | 28/1/2025 | 9/2/2025 | Document interim results & challenges between interim report completion and supervisor check-in |
| 10 | 11/2/2025 | 2/3/2025 | Integrate feedback |
| 11 | 11/2/2025 | 25/2/2025 | Finalise pipeline |
| 12 | 25/2/2025 | 9/3/2025 | Simple front-end |
| 13 | 9/3/2025 | 23/3/2025 | Final Report |
| 14 | 11/3/2025 | 23/3/2025 | Final tests & fixes |

*Figure 11: Project tasks*

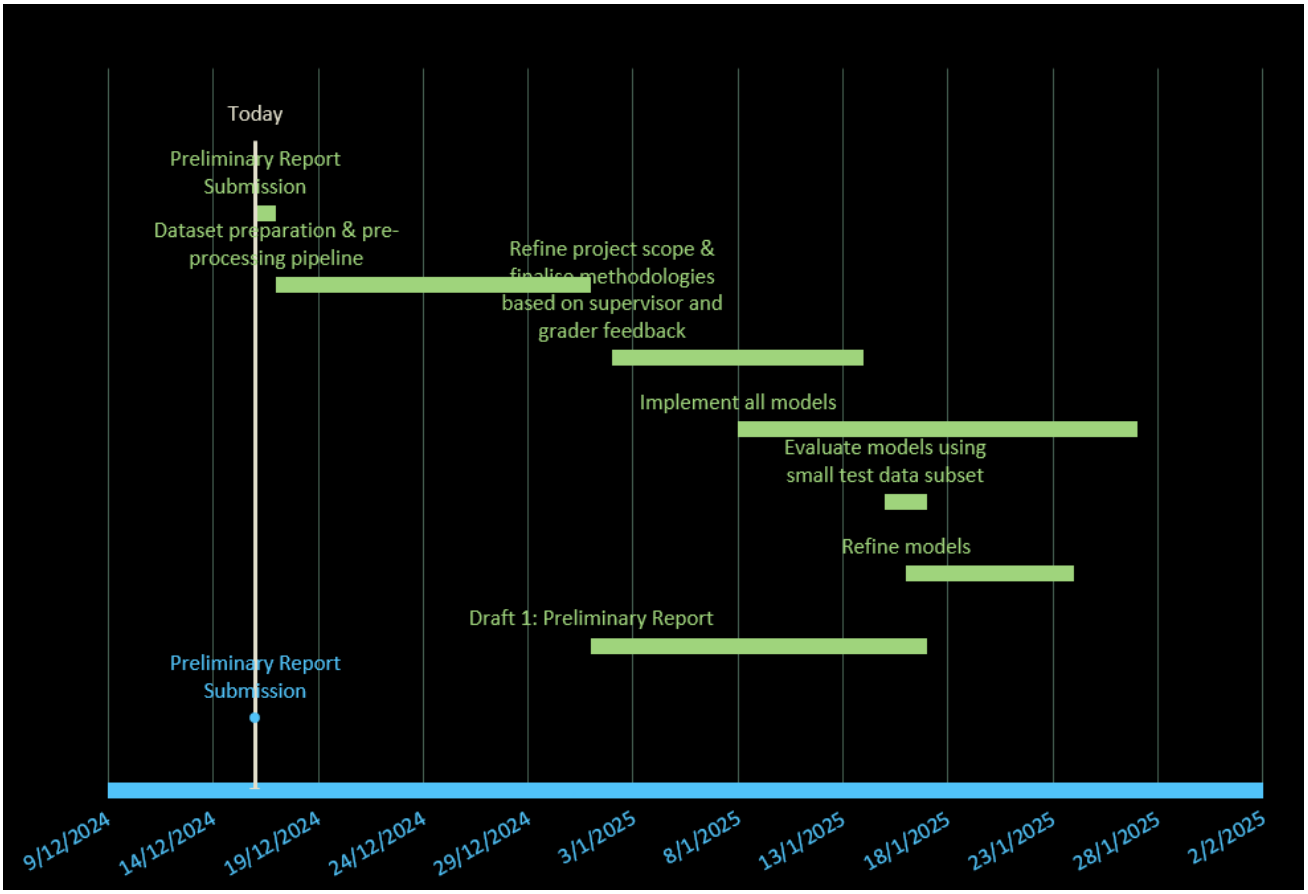



*Figure 12: Gantt chart for project timeline*

# [3.7] Testing & Evaluation

## [3.71] Test Plan

1. **Functional testing**

- Verify end-to-end functionality of the system, from input to output.
- Test BERT model on separate test dataset to evaluate metrics such as accuracy, precision, recall, F1 score.
- Additionally, use confusion matrices to analyse false positives/negatives.

2. **User testing**

- Collect feedback from target users on usability & effectiveness
- Conduct surveys with target users assess ease of use, clarity of results & overall satisfaction.
- Distribute a SUS (System Usability Scale) questionnaire to quantitatively evaluate usability
- Include task-based performance metrics (e.g. time on task, error rates) to measure efficiency.

3. **Security testing**

- Perform input validation tests to test again SQL injection and other vulnerabilities
- Ensure the application handles malformed or adversarial inputs gracefully

## [3.72] Evaluation Metrics

### Model Evaluation Metrics

1. Accuracy
2. Precision
3. Recall (Sensitivity)
4. F1 score
5. ROC-AUC Curve

### User Evaluation Metrics

1. System Usability Score (SUS) score – to assess overall usability
2. Interpretability Score – users rate the clarity of flagged explanations on a 1-5 scale
3. Time on task – to measure how efficiently users complete fake news identification tasks
4. Error rate – to track user misunderstandings of system outputs
5. Net Promoter Score (NPS) – to assess overall user satisfaction, and likelihood of recommending the system

# [4.0] Feature Prototype

## [4.1] Overview

The feature prototype demonstrates the implementation of fine-tuning a MobileBERT [16] model for the purpose of fake news detection.

MobileBERT is a variant of BERT chosen for its computational efficiency while retaining its performance in natural language processing tasks. The objective of this prototype is to train MobileBERT to classify political statements as either 'fake' or 'real' using a simplified, binarized version of the LIAR dataset's labels.

This prototype's implementation serves to assess the viability of employing MobileBERT in a fake news detection application with real-world use cases.

# [4.2] Implementation

## [4.21] Dataset Preparation

The LIAR dataset comprises political statements categorised into six classes – “pants-fire”, “false”, “barely-true”, “half-true”, “mostly-true” and “true” [17] – that encapsulate a spectrum of truthfulness. In order to simplify the classification task for this prototype, these classes were merged into simplified binary labels:

1. Fake – “pants-fire”, “false”, “barely-true”
2. Real – “half-true”, “mostly-true”, “true”

The dataset was then prepared over multiple steps:

1. **Data Loading**

```python
# load dataset to pandas DataFrame

# import pandas \o/
import pandas as pd

# load train, test, validation datasets
# for the purposes of this demo, we'll be using LIAR dataset :D
train_ds = "liar_dataset/train.tsv"
test_ds = "liar_dataset/test.tsv"
valid_ds = "liar_dataset/valid.tsv"

# now, i'll use pandas to read TSV files :D
# columns are as according to the README in liar_dataset directory :D

columns = [
    "id", "label", "statement", "subject", "speaker", "speaker_job_title",
    "state_info", "party_affiliation", "barely_true_counts", "false_counts",
    "half_true_counts", "mostly_true_counts", "pants_on_fire_counts", "context"
]

train_df = pd.read_csv(train_ds, sep='\t', names=columns)
test_df = pd.read_csv(test_ds, sep='\t', names=columns)
valid_df = pd.read_csv(valid_ds, sep='\t', names=columns)
```

Training, validation, and test splits were imported using pandas. This way, data would be cleanly separated, and evaluation of the model would be unbiased.

## 2. Label binarisation

```python
# binarising labels!

# since the labels have multiple classes,
# for the sake of this feature prototype,
# i'll just simplify them to binary true/fake labels :)

# map labels to binary classes! :D
# 'pants-fire', 'false', 'barely-true' -> fake (0)
# others -> real (1)

def binarise(df):
    # validate expected labels exist before applying transformation!
    expected_labels = ["pants-fire", "false", "barely-true", "half-true", "mostly-true", "true"]
    unexpected_labels = set(df['label']) - set(expected_labels)
    if unexpected_labels:
        raise ValueError(f"Unexpected labels found: {unexpected_labels}")
    df['label'] = df['label'].apply(lambda x: 0 if x in ['pants-fire', 'false', 'barely-true'] else 1)
    return df


train_df = binarise(train_df)
test_df = binarise(test_df)
valid_df = binarise(valid_df)
```

To transform multi-class labels to binary labels 'fake' (0) and 'real' (1), a mapping function was implemented to map "pants-fire", "false" and "barely-true" to 'fake' (0), and "half-true", "mostly-true" and "true" to 'real' (1).

```python
# print statement to check df structure!
print(train_df.head())
print(test_df.head())
print(valid_df.head())

# checking that all labels in dataset are valid
print("Unique labels in training data:", train_df['label'].unique())
assert set(train_df['label'].unique()) == {0, 1}, "Labels must be binary (0 or 1)"
```

Some basic validation work was also done to verify that all labels conformed to the expected binary format.

## 3. Tokenisation

Hugging Face's AutoTokenizer was employed to tokenise the text statements. This would ensure uniformity in length through truncation and padding to a maximum sequence length of 128 tokens.

This process preserves critical semantic features without compromising computational efficiency.

```python
# tokenise statements
# i'll tokenise statements using Hugging Face's tokeniser!

# import autotokeniser
from transformers import AutoTokenizer

# load tokeniser
tokeniser = AutoTokenizer.from_pretrained("google/mobilebert-uncased")

# tokenise data
def tokenise(df, tokeniser, max_length=128):
    return tokeniser(
        df['statement'].tolist(),
        truncation=True,
        padding=True,
        max_length=max_length,
        return_tensors="pt"
    )

train_encodings = tokenise(train_df, tokeniser)
test_encodings = tokenise(test_df, tokeniser)
valid_encodings = tokenise(valid_df, tokeniser)
```

# [4.22] Model Training

MobileBERT is then fine-tuned to perform binary classification in the task-specific setting on detecting fake news.

The fine-tuning pipeline included the following key components:

1. **Dataset creation**

```
# prepare our dataset for pytorch!
import torch

class LIARDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        item = {key: val[idx] for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

train_dataset = LIARDataset(train_encodings, train_df['label'].tolist())
test_dataset = LIARDataset(test_encodings, test_df['label'].tolist())
valid_dataset = LIARDataset(valid_encodings, valid_df['label'].tolist())
```

The tokenised text and corresponding labels were encapsulated into PyTorch-compatible datasets to enable it to be seamlessly integrated with PyTorch's DataLoader, thus streamlining the training and evaluation workflows.

2. **Model configuration**

```
# now that our dataframes are tokenised,
# let's load pre-trained BERT.

from transformers import AutoModelForSequenceClassification

# load our model :D
model = AutoModelForSequenceClassification.from_pretrained("google/mobilebert-uncased", num_labels=2)
```

A custom classification head with two output labels was initialised to adapt MobileBERT's architecture for binary classification. This modification would leverage MobileBERT's pre-trained language understanding capabilities while ensuring alignment with the binary classification task.

### 3. Training setup

```
# this codeblock for our dataloader! :D

# num_workers to use multiple cpu cores, pin_memory as we are training on GPU
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16)
valid_loader = DataLoader(valid_dataset, batch_size=16)

# train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, num_workers=4, pin_memory=True)
# test_loader = DataLoader(test_dataset, batch_size=16, num_workers=4, pin_memory=True)
# valid_loader = DataLoader(valid_dataset, batch_size=16, num_workers=4, pin_memory=True)

# this codeblock for optimiser!
optimizer = AdamW(model.parameters(), lr=5e-5)

# this codeblock for scheduler!
num_training_steps = len(train_loader) * 10  # 10 epochs
lr_scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=num_training_steps)

# this codeblock for device config!
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
model.to(device)
```

The AdamW optimiser is employed with a learning rate of 5e-5 to ensure stable gradient updates.

A linear learning rate scheduler with no warm-up steps adapts the learning rate dynamically throughout training.

The training process uses a batch size of 16. Shuffling was applied to the training set to prevent learning biases and to improve generalisation.

## 4. Training loop

```python
# this is our training loop.
# i will also implement early stopping here
from tqdm import tqdm

patience = 3  # stop training after 1 epoch without improvement!
best_loss = float('inf')  # track the best loss achieved so far
patience_counter = 0

model.train()
for epoch in range(10):  # max of 10 epochs, but we might stop earlier!
    epoch_loss = 0
    loop = tqdm(train_loader, leave=True)
    for batch in loop:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

        epoch_loss += loss.item()
        loop.set_description(f"Epoch {epoch}")
        loop.set_postfix(loss=loss.item())

        # check for early stopping!
    if epoch_loss < best_loss:
        best_loss = epoch_loss
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print("triggering early stopping!")
            break
```

The training process was capped to a maximum of 10 epochs. Early stopping was implemented based on validation loss to prevent overfitting.

```
Epoch 0: 100%|██████████| 640/640 [01:43<00:00,  6.21it/s, loss=0.681]
Epoch 1: 100%|██████████| 640/640 [01:42<00:00,  6.23it/s, loss=0.737]
Epoch 2: 100%|██████████| 640/640 [01:43<00:00,  6.18it/s, loss=0.715]
Epoch 3: 100%|██████████| 640/640 [01:42<00:00,  6.27it/s, loss=0.545]
Epoch 4: 100%|██████████| 640/640 [01:44<00:00,  6.15it/s, loss=0.749]
Epoch 5: 100%|██████████| 640/640 [01:41<00:00,  6.33it/s, loss=0.688]
Epoch 6: 100%|██████████| 640/640 [01:42<00:00,  6.24it/s, loss=0.555]
Epoch 7: 100%|██████████| 640/640 [01:42<00:00,  6.27it/s, loss=0.685]
Epoch 8: 100%|██████████| 640/640 [01:42<00:00,  6.23it/s, loss=0.681]
Epoch 9: 100%|██████████| 640/640 [01:42<00:00,  6.24it/s, loss=0.618]
```

Loss values were monitored per epoch, and gradients were computed and optimised after each batch. Early stopping criteria is based on validation loss, and training would stop if no improvement was observed over three consecutive epochs.

## [4.23] Evaluation

```python
# evaluating the model!

# evaluate the model on test data :D
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

model.eval()
predictions, true_labels = [], []

with torch.no_grad():
    for batch in test_loader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        logits = outputs.logits
        predictions.extend(torch.argmax(logits, dim=-1).tolist())
        true_labels.extend(batch['labels'].tolist())

# metrics
accuracy = accuracy_score(true_labels, predictions)
precision = precision_score(true_labels, predictions, zero_division=0)

recall = recall_score(true_labels, predictions)
f1 = f1_score(true_labels, predictions)
roc_auc = roc_auc_score(true_labels, predictions)
```

The test set was used to assess the model's performance using standard classification metrics – accuracy, precision, recall, F1 score and ROC-AUC score.

```python
# visualise results

from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

cm = confusion_matrix(true_labels, predictions)
labels = train_df['label'].unique()
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap="Blues")
```

Additionally, a confusion matrix was generated to represent the distribution of true positives, true negatives, false positives and false negatives. The visualisation is meant to facilitate error analysis and highlight potential areas for model improvement.

# [4.24] Explainability

To enhance interpretability and build user trust, LIME (Local Interpretable Model-agnostic Explanations) was integrated into the prototype to identify and highlight the most influential input features contributing to the model's predictions.

This approach aims to introduce transparency to the "black box" transformer models tend to be, which would allow stakeholders to understand the rationale behind its classifications.

```
# define a wrapper for model predictions
class ModelWrapper:
    def __init__(self, model, tokenizer, max_length=128, device='cpu'):
        self.model = model
        self.tokenizer = tokenizer
        self.max_length = max_length
        self.device = device

    def predict_proba(self, texts):
        # tokenize the input texts
        encodings = self.tokenizer(
            texts,
            truncation=True,
            padding=True,
            max_length=self.max_length,
            return_tensors="pt"
        )
        input_ids = encodings['input_ids'].to(self.device)
        attention_mask = encodings['attention_mask'].to(self.device)

        # get model predictions
        with torch.no_grad():
            outputs = self.model(input_ids, attention_mask=attention_mask)
            logits = outputs.logits

        # convert logits to probabilities
        probs = torch.softmax(logits, dim=-1).cpu().numpy()
        return probs
```

A model wrapper class was created for LIME to interact with the MobileBERT model. It tokenises input text, and model outputs are transformed into interpretable probability scores for "fake" and "real" classes.

```
# initialize the LIME explainer
explainer = LimeTextExplainer(class_names=["Fake", "Real"])

# wrap model and tokenizer
wrapper = ModelWrapper(model, tokeniser, device=device)
```

LimeTextExplainer is initialised with class names "fake" and "real".

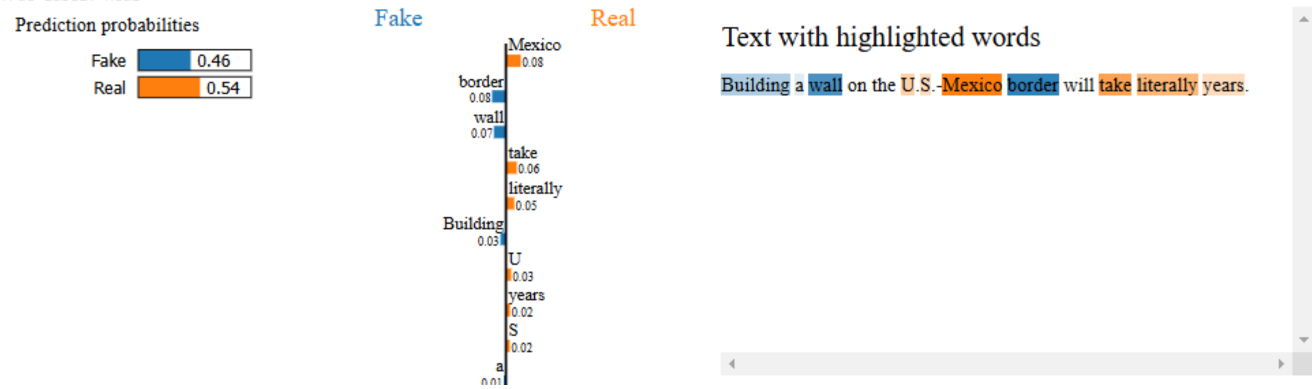
```python
# Select a sample from your test data to explain
sample_index = 0  # Example: explain the first test sample
sample_text = test_df['statement'].iloc[sample_index]
sample_label = test_df['label'].iloc[sample_index]

# generate explanation
explanation = explainer.explain_instance(
    sample_text,
    wrapper.predict_proba,
    num_features=10,  # number of features to include in explanation
    top_labels=1  # focus on the top predicted label
)
```

A test sample is selected from the dataset to generate an explanation. LIME identifies the top 10 features that most influenced the model's prediction for the sample.

```python
# display the explanation
print(f"Original Text: {sample_text}")
print(f"True Label: {'Real' if sample_label == 1 else 'Fake'}")

# show weights of top features for the predicted label
explanation.show_in_notebook(text=True)
```

LIME provides a weighted list of influential features for each explained instance.

```
Original Text: Building a wall on the U.S.-Mexico border will take literally years.
True Label: Real
```



# [4.3] Results of Initial Implementation

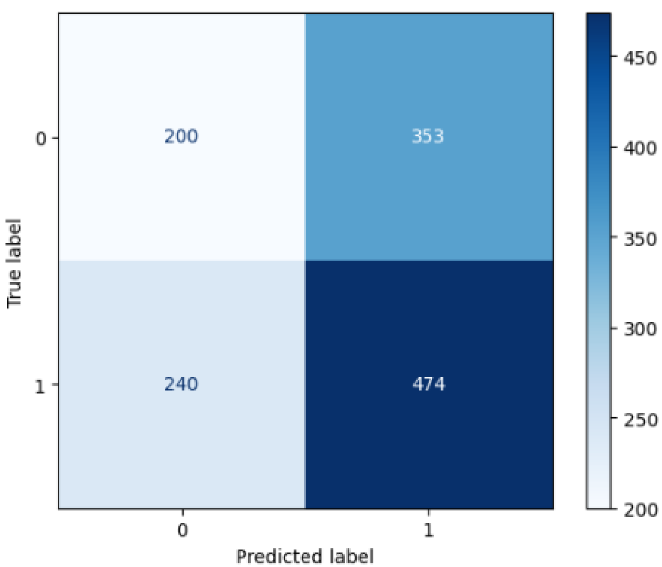
The prototype demonstrated the feasibility of fine-tuning MobileBERT for the task of fake news detection.

The key performance metrics achieved during evaluation are as follows:

```
Accuracy: 0.5320
Precision: 0.5732
Recall: 0.6639
F1 Score: 0.6152
ROC-AUC: 0.5128
```

Accuracy: 53.20%
Precision: 57.32%
Recall: 66.39%
F1 Score: 61.52%
ROC-AUC: 51.28%

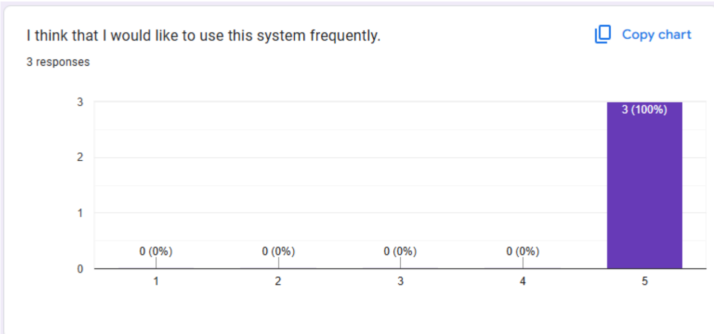
These values indicate that while the model is more accurate than randomly guessing, there is large room for improvement, especially in optimising prediction and recall.

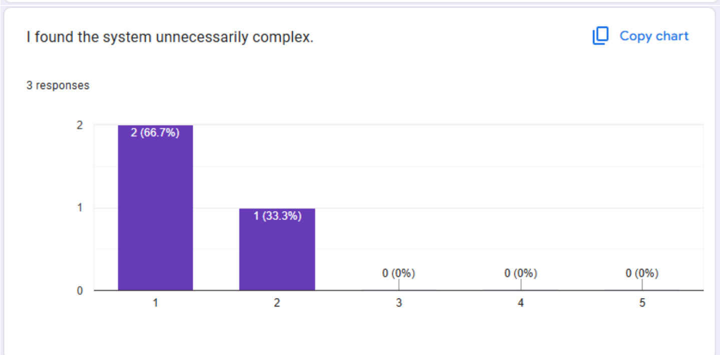


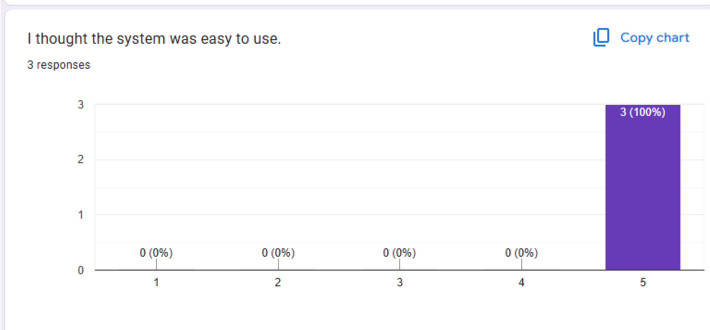

The confusion matrix reveals that the model exhibits a tendency towards higher false positives, thus significantly affecting precision. The high recall score indicates the model is capable of capturing real statements, but improving precision will be a main goal moving forward in future iterations.

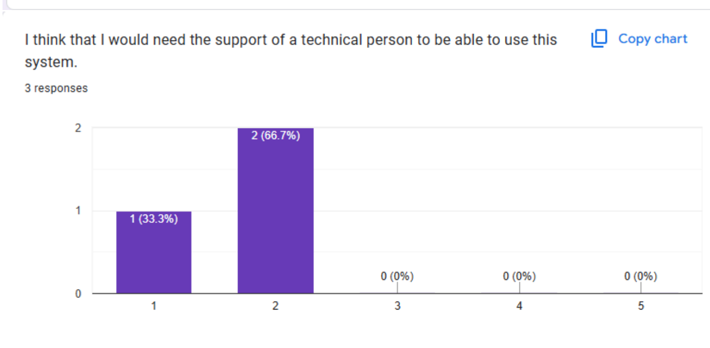
Additionally, a small group survey was conducted with a group of three student jounrliasts.

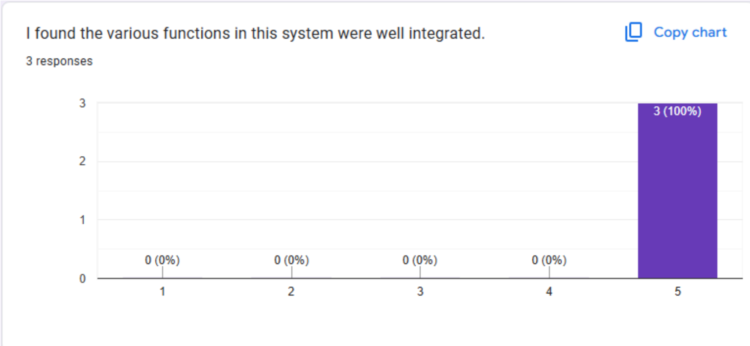
Participants filled in a System Usability Scale (SUS) questionnaire and were asked to rate how much they understood why statements were being flagged.

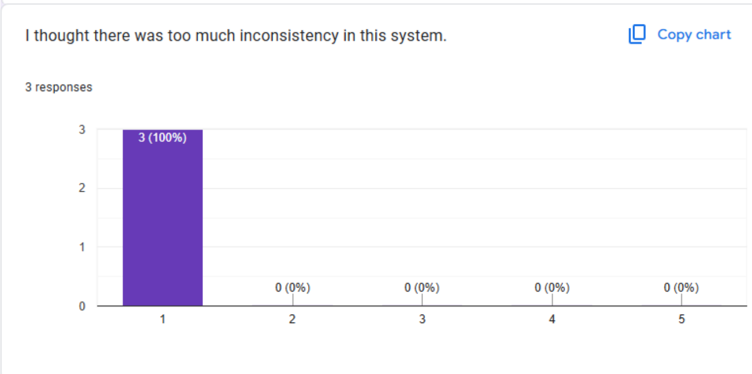


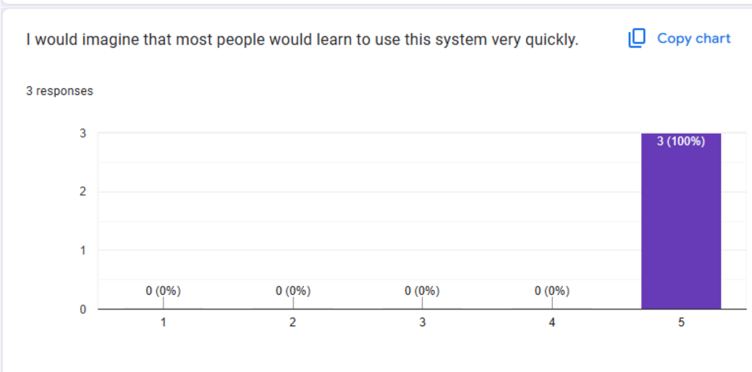



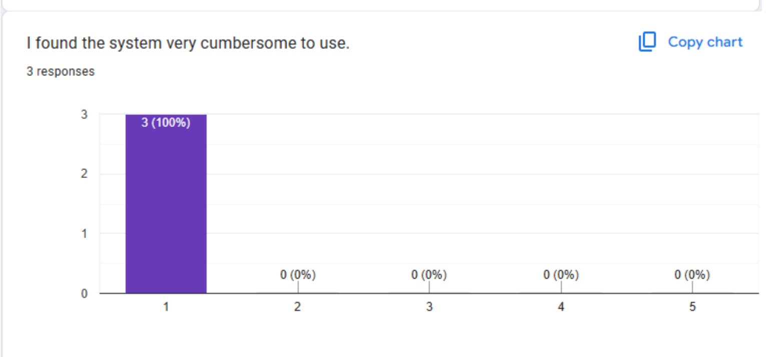



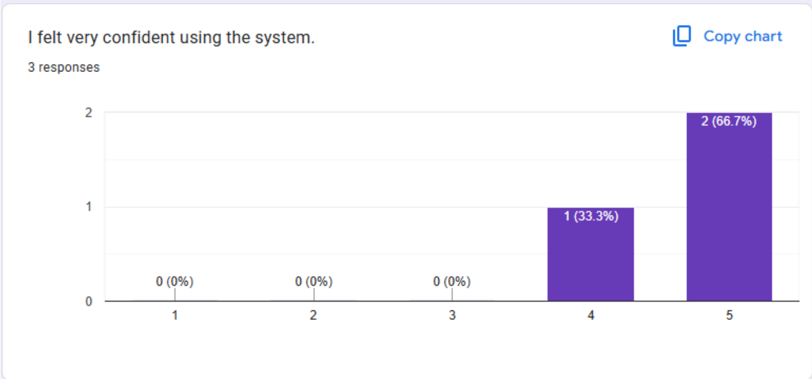



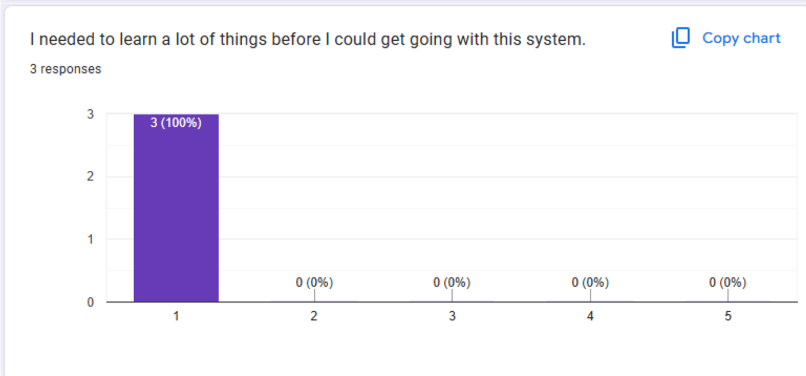

I found the various functions in this system were well integrated.
Copy chart
3 responses
3
2
1
0
0 (0%)
0 (0%)
0 (0%)
0 (0%)
3 (100%)
1
2
3
4
5

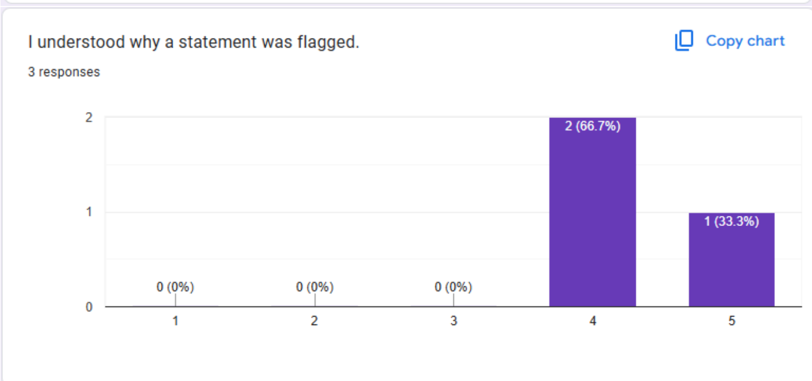

I thought there was too much inconsistency in this system.
Copy chart
3 responses
3
2
1
0
3 (100%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5


I would imagine that most people would learn to use this system very quickly.
Copy chart
3 responses
3
2
1
0
0 (0%)
0 (0%)
0 (0%)
0 (0%)
3 (100%)
1
2
3
4
5


I found the system very cumbersome to use.
Copy chart
3 responses
3
2
1
0
3 (100%)
0 (0%)
0 (0%)
0 (0%)
0 (0%)
1
2
3
4
5




From the results of the questionnaire, I can deduce that:

1. **The participants wanted more clarity as to why statements are flagged.**

   This can be done by introducing a more human explainability element, like a short statement instead of just the LIME output, to be more conducive to non-technical users.

2. **The participants felt a CLI application was too intimidating/technical.**

   Implementing a simple GUI may be a way to bridge the gap and instil confidence in non-technical users.

# [4.4] Further Iterations

As the original implementation did not perform exceptionally well according to model evaluation metrics, in subsequent iterations of the prototypes, more techniques were attempted to improve the performance of the model.

## [4.41] Mixed Precision Training

The original training loop was worked on to implement mixed precision training using PyTorch's torch.cuda.amp module to enable efficient training by performing certain operations in lower 16-bit floating point precision, while maintaining stability in others with a 32-bit floating point precision.

The potential benefit of implementing mixed precision training is to reduce computation time by utilising tensor cores on GPUs, and lower memory usage so larger batch sizes or models would be able to fit in the same memory.

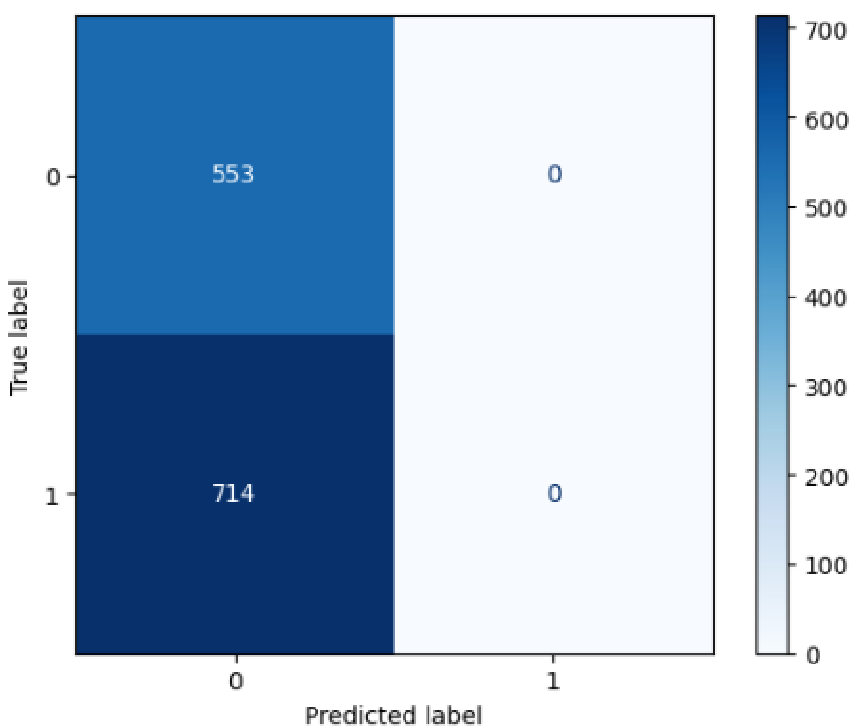
```python
# gradient scaler for mixed precision
scaler = GradScaler()

patience = 3  # stop training after 3 epochs without improvement!
best_loss = float('inf')  # track the best loss achieved so far
patience_counter = 0

model.train()
for epoch in range(10):  # max of 10 epochs, but we might stop earlier!
    epoch_loss = 0
    loop = tqdm(train_loader, leave=True)

    for batch in loop:
        # move batch to device
        batch = {k: v.to(device) for k, v in batch.items()}

        # forward pass with mixed precision
        with autocast():
            outputs = model(**batch)
            loss = outputs.loss  # calculate loss

        # backward pass with gradient scaling
        scaler.scale(loss).backward()

        # gradient clipping. i do this to avoid exploding gradients.
        clip_grad_norm_(model.parameters(), max_norm=1.0)

        # update model parameters
        scaler.step(optimizer)  # apply scaled gradients to optimizer
        scaler.update()  # update the scaler for next iteration
        optimizer.zero_grad()  # reset gradients after updating
        lr_scheduler.step()  # step the scheduler

        # accumulate epoch loss
        epoch_loss += loss.item()

        # progress bar logic
        loop.set_description(f"Epoch {epoch}")
        loop.set_postfix(loss=loss.item())

    # check for early stopping
    if epoch_loss < best_loss:
        best_loss = epoch_loss
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print("triggering early stopping!")
            break
```

To implement mixed precision training,

1. A gradient scaler ‘GradScaler()’ is initialised.
2. ‘autocast’ is used to wrap the forward pass in the training loop.
3. The scaler is used to scale the loss before calling ‘backward’, and to handle the optimizer step and update.

```
Epoch 0: 100%|██████████████████████████████████████████████████| 640/640 [01:40<00:00,  6.38it/s, loss=nan]
Epoch 1: 100%|██████████████████████████████████████████████████| 640/640 [01:38<00:00,  6.50it/s, loss=nan]
Epoch 2: 100%|██████████████████████████████████████████████████| 640/640 [01:34<00:00,  6.78it/s, loss=nan]
triggering early stopping!
```

```
Accuracy: 0.4365
Precision: 0.0000
Recall: 0.0000
F1 Score: 0.0000
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1e5ffe6b090>
```




The results of the model trained with mixed precision training indicated a serious issue in the training process, as evidenced by the NaN loss during training across all epochs. This suggests that the model encountered numerical instability during the training loop, preventing the model from learning any meaningful patterns.

Looking at the evaluation metrics, the precision, recall and F1 score are all zero, indicating that the model trained with mixed precision training did not predict any class correctly. The confusion matrix indicated that all predictions were consistently ‘0’, regardless of the true label.

Attempts were made to reduce the learning rate, monitor gradients and adjust gradient clipping. However, unfortunately, the same issues persisted and time constraints restricted any further troubleshooting and debugging to make mixed precision training work at this time.

# [4.42] Gradient Accumulation and Clipping

Gradient accumulation enables training with larger effective batch sizes without increasing memory requirements by accumulating gradients over multiple batches before performing an update, instead of updating the model parameters after each batch.

The potential improvement of implementing this is improved model stability, convergence, and numerical stability due to the prevention of exploding gradients. This also improves resource efficiency, as GPU memory usage is reduced as compared to directly increasing the batch size.

```
model.train()
for epoch in range(10):  # max of 10 epochs, but we might stop earlier!
    epoch_loss = 0
    loop = tqdm(train_loader, leave=True)
    optimizer.zero_grad()  # reset gradients at the start of the epoch

    for i, batch in enumerate(loop):
        batch = {k: v.to(device) for k, v in batch.items()}

        # forward pass
        outputs = model(**batch)
        loss = outputs.loss / accumulation_steps  # normalize loss for accumulation
        loss.backward()  # backpropagation

        # perform optimization step after accumulation_steps
        if (i + 1) % accumulation_steps == 0 or (i + 1) == len(loop):
            # gradient clipping
            torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm)

            # update model parameters
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()  # after update, reset gradients

        # accumulate epoch loss
        epoch_loss += loss.item() * accumulation_steps  # scale loss back to original value

        # progress bar update logic
        loop.set_description(f"Epoch {epoch}")
        loop.set_postfix(loss=loss.item() * accumulation_steps)

    # early stopping logic
    if epoch_loss < best_loss:
        best_loss = epoch_loss
        patience_counter = 0
    else:
        patience_counter += 1
        if patience_counter >= patience:
            print("triggering early stopping!")
            break
```

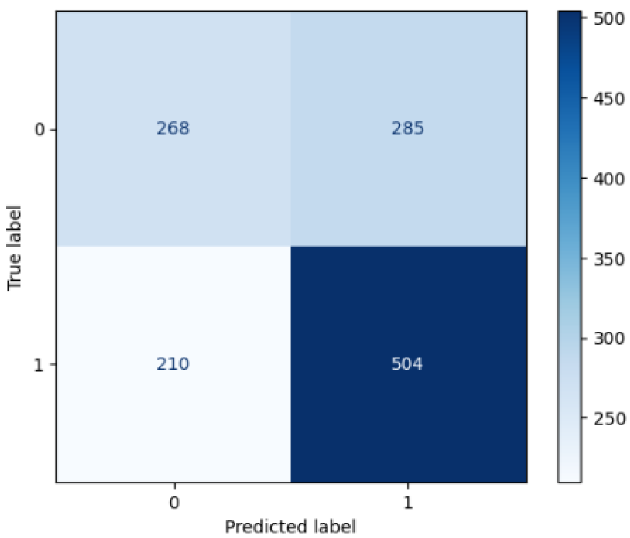
The original training loop was adapted to implement gradient accumulation and clipping, through the following key changes:

1. The loss is divided by accumulation steps to ensure that gradients are appropriately scaled during accumulation,
2. Gradients are clipped before the optimiser step to prevent exploding gradients. 'max_norm' sets the maximum gradient norm.
3. The model parameters are updated only after 'accumulation_steps' batches.
4. 'optimizer.zero_grad()' is called after the optimiser step to reset gradients.

```
Epoch 0: 100%|██████████| 640/640 [01:19<00:00,  8.04it/s, loss=0.775]
Epoch 1: 100%|██████████| 640/640 [01:20<00:00,  7.94it/s, loss=0.757]
Epoch 2: 100%|██████████| 640/640 [01:19<00:00,  8.05it/s, loss=0.548]
Epoch 3: 100%|██████████| 640/640 [01:19<00:00,  8.07it/s, loss=0.767]
Epoch 4: 100%|██████████| 640/640 [01:20<00:00,  7.94it/s, loss=0.614]
Epoch 5: 100%|██████████| 640/640 [01:22<00:00,  7.76it/s, loss=0.447]
Epoch 6: 100%|██████████| 640/640 [01:21<00:00,  7.88it/s, loss=0.436]
Epoch 7: 100%|██████████| 640/640 [01:20<00:00,  7.96it/s, loss=0.275]
Epoch 8: 100%|██████████| 640/640 [01:17<00:00,  8.30it/s, loss=0.141]
Epoch 9: 100%|██████████| 640/640 [01:16<00:00,  8.34it/s, loss=0.0166]
```

During training, the loss steadily decreases across epochs, starting from 0.775 in epoch 0 and finally converging to 0.0166, indicating that the model is learning the training data effectively.

```
Accuracy: 0.6093
Precision: 0.6388
Recall: 0.7059
F1 Score: 0.6707
ROC-AUC: 0.5953
```




The accuracy value of 60.93% has improved compared to the initial implementation (53.2%), as has the precision score (63.88%) compared to the initial implementation (57.32%). The recall and F1 score have also increased, though the current values and confusion matrix indicates the model fails to correctly classify more true instances, and suggests that while gradient accumulation and clipping have improved training stability, the balance between precision and recall is not fully optimised yet.

While the significant decrease in training loss has ranslated to better evaluation metrics, overfitting may be occurring as the loss has converged too aggressively to near-zero, suggesting that the model may have potentially memorised the training data while not generalising well to unseen data.

Attempts to tune the learning rate, increase the patience for early stopping, and using weighted loss functions were made to improve on the results, however the evaluation metrics did not see tangible improvements, with time constraints prohibiting further troubleshooting.

# [4.5] Future Improvements

While time constraints limited the amount of iterative work done on the prototype, several possible areas for improvement were identified for future work.

1. **Addressing class imbalance**

   The dataset had an uneven distribution of labels. This potentially skewed the model's performance, and can be addressed by techniques such as oversampling or applying class weights during training to mitigate bias and improve the model's robustness.

2. **Precision improvements**

   The high rate of false positives indicates a need to refine decision boundaries, possibly by adding more nuanced examples of fake statements to the dataset, or applying advanced loss functions to penalise false positives.

3. **Expanding evaluation metrics**

   Incorporating additional evaluation metrics and producing more comprehensive user evaluation metrics could provide more comprehensive insights and a better understanding of the model's reliability. This would be critical when the model is deployed for real-world applications, where confidence in predictions is crucial.

4. **More human-like explainability features**

   To instil confidence and foster user trust, explainability features need to be understood by users. Implementing short and easily-understandable textual explanations as to why content is flagged would be more useful for non-technical users who may feel alienated by LIME's charts.

# [5.0] References

[1] “Advanced Python Project - Detecting Fake News with Python,” DataFlair, Sep. 16, 2019. https://data-flair.training/blogs/advanced-python-project-detecting-fake-news/

[2] “ClaimBuster,” Uta.edu, 2024. https://idir.uta.edu/claimbuster/event/70/ (accessed Dec. 15, 2024).

[3] Snopes, “Snopes.com,” Snopes.com, 2018. https://www.snopes.com/

[4] L.-J. Chang, “Comparison of Machine Learning and Deep Learning Algorithms in Detecting Fake News,” Proceedings of the 28th World Multi-Conference on Systemics, Cybernetics and Informatics, pp. 203–209, Sep. 2024, doi: https://doi.org/10.54808/wmsci2024.01.203.

[5] K. Huh, “Surviving In a Random Forest with Imbalanced Datasets,” SFU Professional Computer Science, Feb. 13, 2021. https://medium.com/sfu-cspmp/surviving-in-a-random-forest-with-imbalanced-datasets-b98b963d52eb

[6] J. Devlin, M.-W. Chang, K. Lee, K. Google, and A. Language, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” 2019. Available: https://aclanthology.org/N19-1423.pdf

[7] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, “MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices,” arXiv:2004.02984 [cs], Apr. 2020, Available: https://arxiv.org/abs/2004.02984

[8] V. Korolev and P. Protsenko, “Accurate, interpretable predictions of materials properties within transformer language models,” Oct. 13, 2023. Accessed: Dec. 15, 2024. [Online]. Available: https://www.cell.com/patterns/pdfExtended/S2666-3899(23)00158-7

[9] “ClaimBuster,” RAND. https://www.rand.org/research/projects/truth-decay/fighting-disinformation/search/items/claimbuster.html

[10] N. Hassan et al., “ClaimBuster,” Proceedings of the VLDB Endowment, vol. 10, no. 12, pp. 1945–1948, Aug. 2017, doi: https://doi.org/10.14778/3137765.3137815.

[11] Y. Zhu, Y. Li, J. Wang, M. Gao, and J. Wei, “FaKnow: A Unified Library for Fake News Detection,” arXiv (Cornell University), 2024. https://arxiv.org/abs/2401.16441 (accessed Dec. 15, 2024).

[12] L. Graves and F. Cherubini, “The Rise of Fact-Checking Sites in Europe,” Reuters Institute, 2016. Available: https://reutersinstitute.politics.ox.ac.uk/sites/default/files/research/files/The%2520Rise%2520of%2520Fact-Checking%2520Sites%2520in%2520Europe.pdf

[13] X.-J. Liu, Q. Li, L. Wang, and M. J. Metzger, “Checking the Fact-Checkers: The Role of Source Type, Perceived Credibility, and Individual Differences in Fact-Checking Effectiveness,” Communication Research, Oct. 2023, doi: https://doi.org/10.1177/00936502231206419.

[14] J. Yang, D. A. Vega-Oliveros, T. Seibt, and A. Freitas, “Scalable Fact-checking with Human-in-the-Loop,” arXiv (Cornell University), Dec. 2021, doi: https://doi.org/10.1109/wifs53200.2021.9648388.

[15] Zi Hen Lin, Z. Wang, M. Zhao, Y. Song, and L. Lan, “An AI-based System to Assist Human Fact-Checkers for Labeling Cantonese Fake News on Social Media,” 2022 IEEE International Conference on Big Data (Big Data), Dec. 2022, doi: https://doi.org/10.1109/bigdata55660.2022.10020949.

[16] “google/mobilebert-uncased · Hugging Face,” Hugging Face. https://huggingface.co/google/mobilebert-uncased

[17] tfs4, “GitHub - tfs4/liar_dataset: dataset liar,” GitHub, 2019. https://github.com/tfs4/liar_dataset (accessed Dec. 15, 2024).