UNIVERSITY OF PORTSMOUTH

# Big Data and Machine Learning Applications

**Assessment Item number: 2 (coursework)**

**Assessment Title: Coursework**

**Module Code: M33146**

Module Coordinator Other lecturers

Dr Hamidreza khaleghzadeh

Student id:

UP2303248

# Real-time Fake News Detection using Big Data and Machine Learning

## Introduction

In the digital era, the spread of misinformation has emerged as a serious societal challenge. With news consumption happening primarily through online channels and social media platforms, false information often circulates unchecked, swaying public opinion, misleading users, and even impacting political and economic decisions. Combating this issue requires intelligent systems that can quickly and accurately determine the authenticity of information.

This project, titled **"Real-time Fake News Detection using Big Data and Machine Learning,"** addresses the problem by building a sophisticated fake news detection system. The system utilizes natural language processing (NLP), classical machine learning (ML) algorithms, and modern deep learning architectures to classify news articles as either **real** or **fake**. We worked with a sizable dataset containing nearly 45,000 full-length news articles, framing it as a real-world big data challenge. The goal was to process, model, and evaluate textual data efficiently, ultimately developing a solution ready for real-time implementation.

## Objective

The key objective of this project was to develop a robust and scalable system for identifying fake news articles by analysing their textual content. This was achieved through:

- Preprocessing and cleaning a large corpus of unstructured text data.
- Implementing a range of machine learning and deep learning models to classify news articles.
- Comparing the performance of traditional ML models such as Logistic Regression and Support Vector Machines (SVM) against deep learning models like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU).
- Tuning deep learning hyperparameters using **KerasTuner** to enhance model efficiency.
- Evaluating each model using key metrics such as accuracy, precision, recall, F1-score, and confusion matrices.

This end-to-end process allowed us to gain deeper insights into the strengths and limitations of various modeling techniques for fake news detection and determine which approach is best suited for real-time applications.

# Dataset Overview

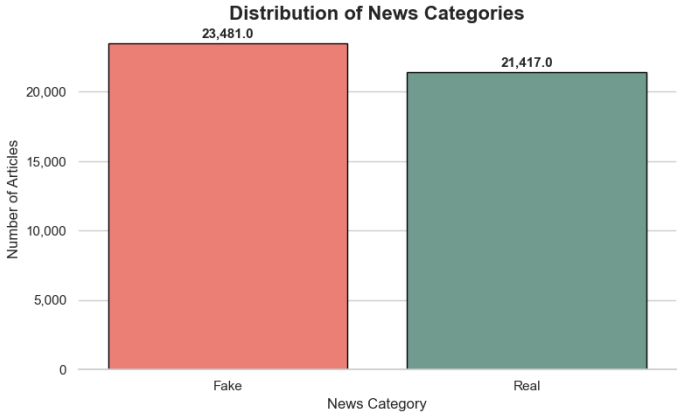
To build a robust fake news detection system, we used two datasets:

- **Fake.csv**: Contains 23,481 entries of fake news articles.
- **True.csv**: Contains 21,417 entries of real news articles.

A total of **44,898 articles** were processed. Each entry contains an entire news article, often consisting of thousands of words. Given the size and textual nature of the dataset, it qualifies as a big data problem.

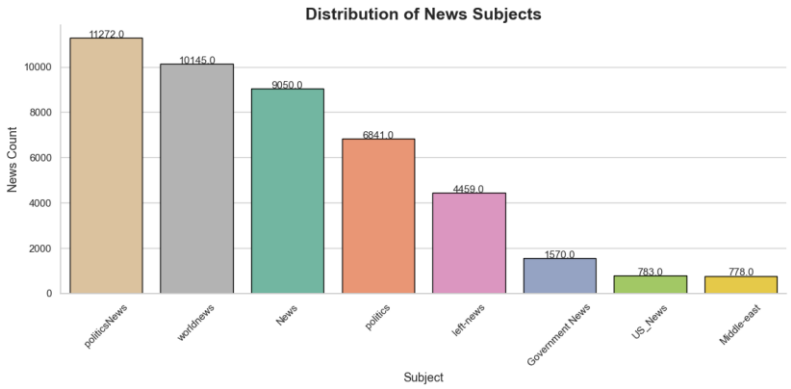
We added a new column called **label**:

- 0 for fake news.
- 1 for real news.

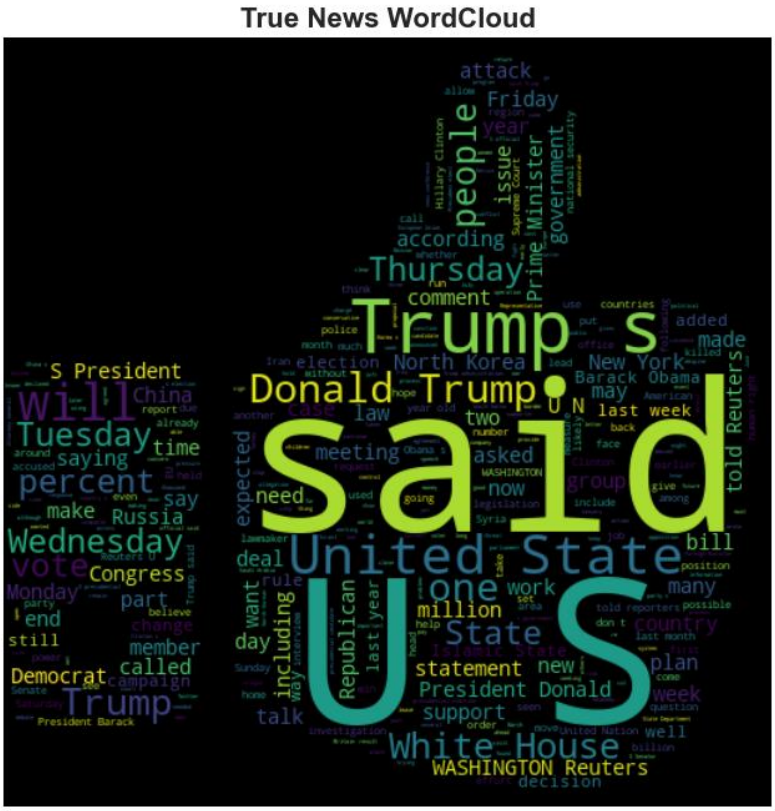



# Exploratory Data Analysis

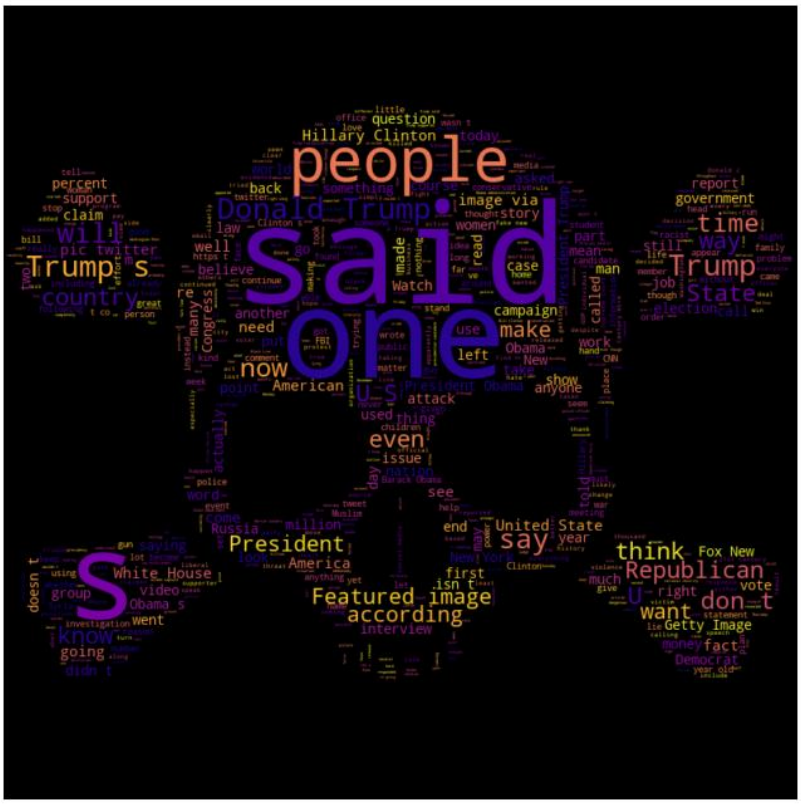
Visualized the most common words using WordCloud from the **PIL** and **matplotlib** libraries to understand the patterns in real and fake news. Word frequency differences provided useful insight for feature engineering.



## WordCloud for True news:




## WordCloud for Fake news:



# Text Preprocessing

Text pre-processing is a critical step in natural language processing that prepares raw data for modeling by cleaning, normalizing, and structuring it. Given the dataset comprises **44,898 news articles** (23,481 fake and 21,417 real), each containing hundreds to thousands of words, pre-processing was essential for handling the sheer volume and complexity of textual data.

## 1. Removing Empty or Whitespace-Only Texts

Text entries that were either completely empty or contained only whitespace characters were removed from the dataset. These entries do not contribute any meaningful information and may interfere with model training or evaluation.

**Example:**

• Original: `"  "`

• Processed: *(Removed from dataset)*

**Observation:** Eliminating such entries helped clean the dataset and ensured all samples contained actual content for analysis.

## 2. Removing Duplicate Text

Duplicate or near-duplicate articles were removed to prevent data leakage between training and testing sets and to ensure that each data point contributed unique information. This step enhanced model generalization and reduced overfitting.

**Example**:

- Original: `"Government announces new law. Government announces new law."`
- Processed: `"Government announces new law"`

**Observation**: Removing redundancy helped in achieving a more representative and diverse dataset for model training.

## 3. Converting to Lowercase

Text was converted to lowercase to ensure uniformity across tokens. This prevented the same word in different cases (e.g., “News” vs. “news”) from being treated as separate entities

**Example**:

- Original: `"Fake News Spreads Quickly"`
- Processed: `"fake news spreads quickly"`

**Observation**: Uniform text representation ensures consistent word embeddings and reduces vocabulary size.

## 4. Remove Special Characters, Numbers, and Punctuation

Special characters (e.g., @, #, %, &, *), numerical digits (e.g., 123, 2020, 3.5), and punctuation symbols (e.g., ., ,, !, ?, :) were removed from the text. These elements typically do not add meaningful semantic value in text classification tasks and can introduce noise during tokenization and model training.

**Example:**

• Original: `"Alert! 5,000 new COVID-19 cases reported @ 10:00 a.m."`

• Processed: `"Alert new COVID cases reported am"`

**Observation:** The removal of special characters, numbers, and punctuation cleaned the input text, improved consistency, and enhanced the quality of the tokenized data for downstream analysis.

## 5. Removing Stop Words

Stop words common words such as "the", "is", "in", "and", "for" were removed because they appear frequently across texts but carry minimal semantic value. Removing them allows models to focus on more informative content.

**Example**:

- Original: `"The government is working for the people."`
- Processed: `"government working people"`

**Observation**: This step enhanced model focus on key informative terms such as "government" and "working", improving classification accuracy.

## 6. Lemmatization

Lemmatization was used to convert words to their root form using grammatical context. Unlike stemming, lemmatization uses morphological analysis to ensure that words are converted to dictionary-valid forms.

**Example**:

- Original: `"Authorities were investigating"`

- Processed: `"authority be investigate"`

**Observation**: Lemmatization helped normalize verb tenses and plural forms, resulting in improved feature consistency and model interpretability.

# Illustrative Summary of Text Preprocessing Steps

| Step | Original | Processed |
|---|---|---|
| Removing Empty or Whitespace-Only Texts | " " | *(Removed from dataset)* |
| Removing Duplicate Text | "Government announces new law. Government announces new law." | "Government announces new law" |
| Converting to Lowercase | "Fake News Spreads Quickly" | "fake news spreads quickly" |
| Removing Special Characters, Numbers, and Punctuation | "Alert! 5,000 new COVID-19 cases reported @ 10:00 a.m." | "Alert new COVID cases reported am" |
| Removing Stop Words | "The government is working for the people." | "government working people" |
| Lemmatization | "Authorities were investigating" | "authority be investigate" |

These preprocessing steps dramatically improved the quality and usability of the data, making it suitable for input into machine learning and deep learning models. They reduced vocabulary size, removed noise, and enhanced token consistency—helping classifiers better distinguish between **fake** and **real** news content.

# Model Implementation and Evaluation

To ensure a comprehensive analysis, used both classical and deep learning approaches.

### Feature Extraction

**Vectorization**:

- For traditional ML models: Used **TF-IDF** (Term Frequency-Inverse Document Frequency) to convert text into numerical features.

- For deep learning: Used tokenized sequences and word embeddings.

**Train-Test Split**:

- Dataset was split into 80% training and 20% testing sets.

# Logistic Regression (Baseline Model)

**Description**: Logistic Regression is a linear model for binary classification that applies the sigmoid function to predict probabilities.

**Why it was chosen**: It's simple, interpretable, and serves as a great baseline for binary classification problems.

**Performance**:

- **Accuracy**: 98.58%
- **Precision**: 0.99
- **Recall**: 0.98
- **F1-Score**: 0.99

| Label | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.99 | 0.98 | 0.99 | 4733 |
| 1 | 0.98 | 0.99 | 0.99 | 4247 |

**Analysis**: This model performed surprisingly well given its simplicity. However, it is limited in capturing complex semantic patterns.

```
Logistic Regression Model Evaluation:
Accuracy: 0.9858574610244989
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      4733
           1       0.98      0.99      0.99      4247

    accuracy                           0.99      8980
   macro avg       0.99      0.99      0.99      8980
weighted avg       0.99      0.99      0.99      8980
```

# Support Vector Machine (SVM)

**Description**: SVM finds an optimal hyperplane to separate classes by maximizing the margin between them. We used a linear kernel.

**Why it was chosen**: SVMs are powerful classifiers, especially effective with high-dimensional spaces like TF-IDF matrices.

**Performance**:

- **Accuracy**: 99.29%
- **Precision**: 0.99
- **Recall**: 0.99
- **F1-Score**: 0.99

| Label | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 | 4733 |
| 1 | 0.99 | 0.99 | 0.99 | 4247 |

**Analysis**: SVM outperformed logistic regression and showed impressive accuracy even without deep semantic understanding. It proves classical models remain strong contenders in NLP tasks.

```
SVM Model Evaluation:
Accuracy: 0.9929844097995546
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      4733
           1       0.99      0.99      0.99      4247

    accuracy                           0.99      8980
   macro avg       0.99      0.99      0.99      8980
weighted avg       0.99      0.99      0.99      8980
```

# LSTM (Long Short-Term Memory)

**Description**: LSTM is a type of recurrent neural network (RNN) capable of learning long-term dependencies. It uses memory cells with input, output, and forget gates.

**Why it was chosen**: Ideal for sequential data like text, where context and order matter.

**Performance**:

• Accuracy: 98.50%

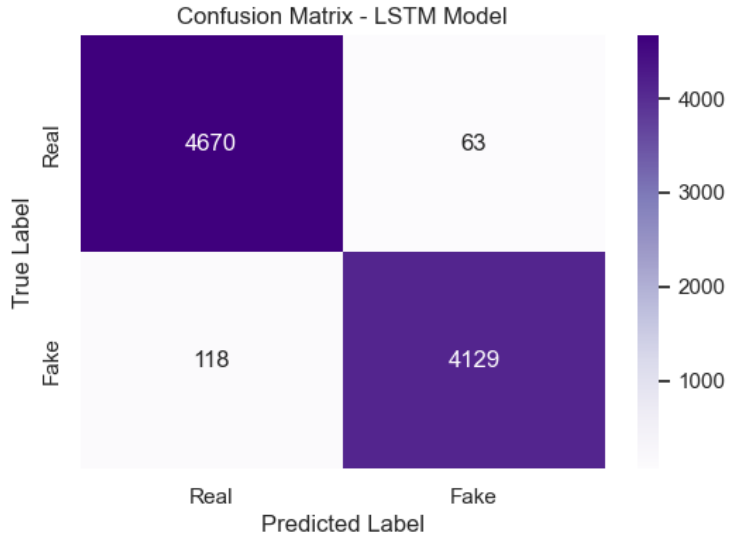
• Precision: 0.98

• Recall: 0.98

• F1-Score: 0.98

| Label | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.98 | 0.98 | 0.98 | 4733 |
| 1 | 0.98 | 0.98 | 0.98 | 4247 |

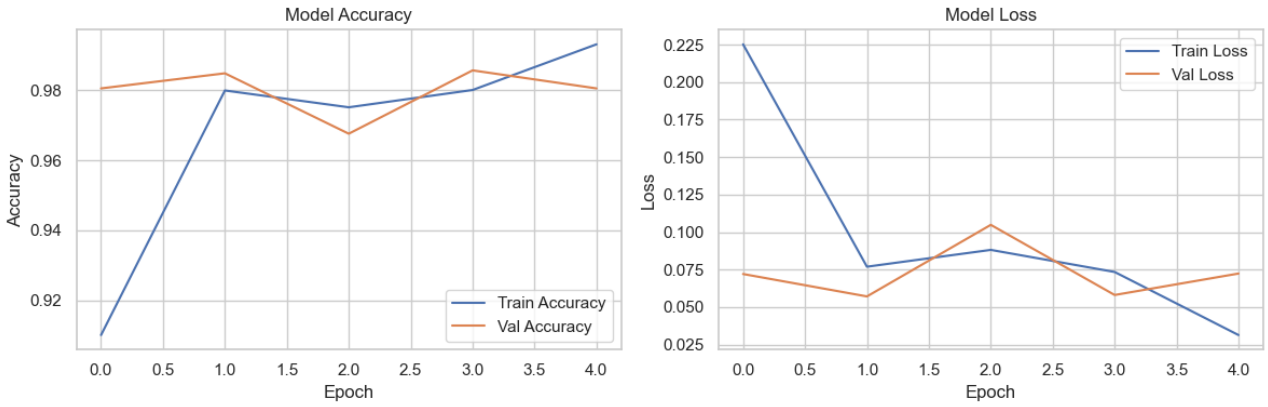
### Confusion Matrix:




**Analysis**: Although LSTM provided excellent results, it was computationally expensive and slower to train. Performance was slightly lower than SVM, but it captured textual patterns better.

```
281/281 ━━━━━━━━━━━━━━━━━━━━ 11s 40ms/step
LSTM Model Evaluation
Accuracy: 0.9798440979955456

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      4733
           1       0.98      0.97      0.98      4247

    accuracy                           0.98      8980
   macro avg       0.98      0.98      0.98      8980
weighted avg       0.98      0.98      0.98      8980


Confusion Matrix:
 [[4670   63]
 [ 118 4129]]
```

## Training history of LSTM model (Long Short-Term Memory):



# GRU (Gated Recurrent Unit)

**Description**: GRU is a variation of LSTM with fewer parameters and simplified architecture. It retains similar effectiveness while being computationally efficient.

**Why it was chosen**: Efficient for large datasets and suitable for real-time applications.

**Performance**:

• Accuracy: 99.02%

• Precision: 0.99

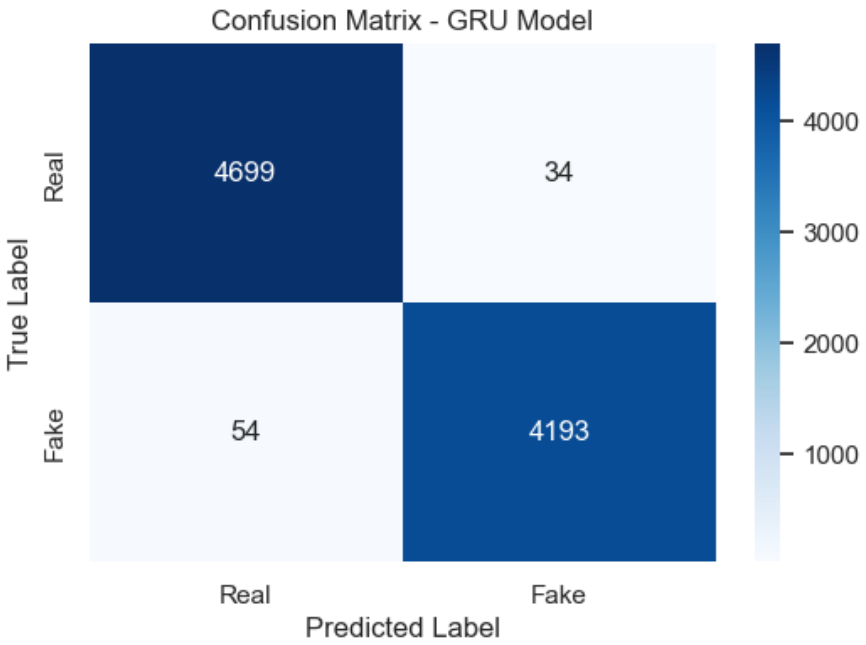
• Recall: 0.99

• F1-Score: 0.99

| Label | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 | 4733 |
| 1 | 0.99 | 0.99 | 0.99 | 4247 |

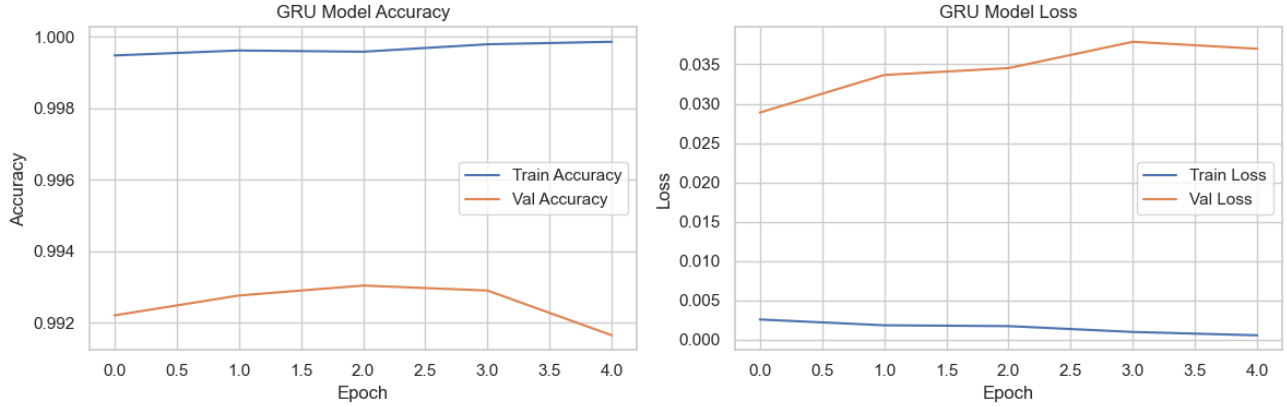
```
281/281 ━━━━━━━━━━━━━━━━━━━━ 7s 24ms/step
GRU Model Evaluation
Accuracy: 0.9902004454342984

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      4733
           1       0.99      0.99      0.99      4247

    accuracy                           0.99      8980
   macro avg       0.99      0.99      0.99      8980
weighted avg       0.99      0.99      0.99      8980


Confusion Matrix:
 [[4699   34]
 [  54 4193]]
```

### Confusion Matrix:




**Analysis**: GRU gave better performance than LSTM with less training time. It was closer to SVM in terms of accuracy.

### Training history of GRU model (Gated Recurrent Unit):




# Hyperparameter Tuning with KerasTuner

To further enhance the GRU model's performance, **KerasTuner** was used to automate hyperparameter optimization.

**Parameters Tuned:**

**Embedding Dimension**: Space to represent text (word embeddings).

**GRU Units**: Number of neurons in the GRU layer.

**Dropout Rate**: Prevents overfitting by dropping neurons randomly during training.

**Learning Rate**: Controls the pace at which the model updates weights.

**Best Hyperparameters Found:**

- **Embedding Dimension**: 64
- **GRU Units**: 256
- **Dropout Rate**: 0.3
- **Learning Rate**: 0.001

```
Best Hyperparameters:
Embedding Dim: 64
GRU Units: 256
Dropout Rate: 0.30000000000000004
Learning Rate: 0.001
```

# Tuned GRU

**Description:**

This GRU model was optimized using hyperparameter tuning (e.g. Embedding Dimension, dropout, units, learning rate) to improve generalization and accuracy.

**Why it was chosen:**

To maximize GRU's performance without increasing complexity ideal for deployment in real-time classification systems.

**Performance after Tuning:**

• Accuracy: 99.29%

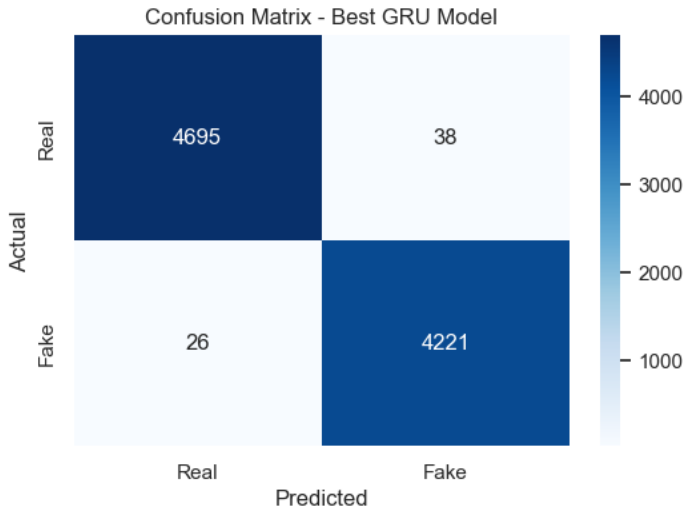
• Precision: 0.99

• Recall: 0.99

• F1-Score: 0.99

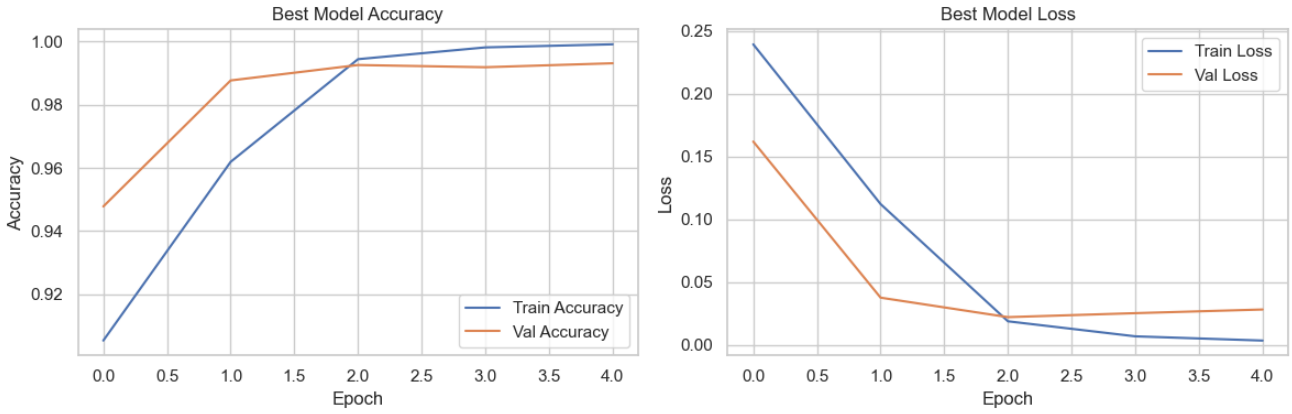
```
Best Model Evaluation
Accuracy: 0.9928730512249443

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      4733
           1       0.99      0.99      0.99      4247

    accuracy                           0.99      8980
   macro avg       0.99      0.99      0.99      8980
weighted avg       0.99      0.99      0.99      8980


Confusion Matrix:
 [[4695   38]
 [  26 4221]]
```

### Confusion Matrix:




The tuned GRU model now matched the performance of SVM while being optimized for sequence-based text analysis.

### Training history of GRU model (Gated Recurrent Unit):




# Model Comparison

| Model | Accuracy |
|---|---|
| Logistic Regression | 98.58% |
| SVM | 99.29% |
| LSTM (Before Tuning) | 97.98% |
| GRU (Before Tuning) | 99.02% |
| GRU (After Tuning) | 99.29% |

### Best Performer:

The **tuned GRU model** and **SVM** both achieved an accuracy of **99.29%**, making them the top-performing models in this study.

**SVM**: Simpler to train and very effective.

**GRU (Tuned)**: Efficient in handling long-term dependencies and scalable for real-time applications.

Given the sequential nature of news articles, GRU offers a better long-term solution for production use.

# Conclusion:

This project demonstrated the power of combining big data and machine learning to solve a highly relevant real-world problem — fake news detection. Here's what we learned:

**Dataset Scale**: The dataset contained nearly **45,000 full-length news articles**, each with thousands of words, making it a true big data problem.

### Traditional vs Deep Learning:

- **Logistic Regression** and **SVM** offered high accuracy with minimal training time.
- **Deep learning models** like **LSTM** and **GRU** provided nuanced understanding and better handling of sequential data.

### Hyperparameter Tuning Matters:

- The performance of the GRU model significantly improved after hyperparameter tuning, making it equal to SVM.

### Real-time Suitability:

- The GRU model is more suitable for real-time systems due to its speed and accuracy post-tuning.

### Interpretability vs Power:

- SVM is easier to interpret and debug.
- GRU provides better handling of complex sequential information.

This comprehensive evaluation shows that a well-tuned deep learning model like GRU can match and even outperform classical models when optimized correctly. The project is a step toward building scalable, real-time systems that can combat the spread of misinformation effectively.

# Future Work

**Deployment**: Integrate the best-performing model into a real-time API.

**Explainability**: Add explainable AI (XAI) techniques to interpret predictions.

**Multilingual Support**: Extend to non-English news sources.

**Fake News Categorization**: Classify fake news into political, financial, health, etc.

This project lays a strong foundation for a production-ready misinformation detection engine leveraging both traditional ML and deep learning techniques.

# References:

## Deep Learning Models (LSTM & GRU)


1. **Hochreiter, S., & Schmidhuber, J. (1997).**
   *Long Short-Term Memory.*
   *Neural Computation*, 9(8), 1735–1780.
   The foundational paper introducing LSTM networks.
2. **Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014).**
   *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.*
   *arXiv preprint arXiv:1406.1078*.

   Introduced GRU, a simplified and computationally efficient alternative to LSTM.


## Text Preprocessing & NLP Techniques


3. **Manning, C. D., Raghavan, P., & Schütze, H. (2008).**
   *Introduction to Information Retrieval.*
   *Cambridge University Press.*

   Provides core techniques for text processing, stop word removal, and vectorization.

4. **Bird, S., Klein, E., & Loper, E. (2009).**
   *Natural Language Processing with Python.*
   *O'Reilly Media.*

   Practical guide to text preprocessing using NLTK, tokenization, and cleaning.