A Comparative Study of Naive Bayes and Logistic Regression for Fake News Detection

UJJAL RAY
Department of Mathematics
Indian Institute of Space Science and Technology (IIST)
SC CODE: SC25M170

***Abstract*—This work presents a scalable and efficient fake news classification framework leveraging classical machine learning approaches combined with GPU-accelerated computation. The workflow begins with structured dataset preparation, including noise removal, handling of missing entries, and verification of class distribution to ensure fair and unbiased learning. A robust preprocessing pipeline is implemented consisting of tokenization, vocabulary construction, document frequency analysis, and computation of inverse document frequency (IDF), leading to the generation of an optimized sparse TF–IDF matrix representation.**

Following feature extraction, the dataset is divided into training and testing subsets, and two supervised classifiers—Multinomial Naive Bayes and mini-batch Logistic Regression—are trained and evaluated. The Logistic Regression model is implemented using PyTorch with CUDA support, enabling accelerated matrix operations, efficient gradient descent optimization, and reduced training time, particularly for high-dimensional textual data.

The models are assessed using key performance metrics to measure accuracy, generalization, and robustness. Additionally, a real-time prediction interface is developed to allow users to input custom text and receive immediate classification feedback. The results demonstrate that GPU-enabled Logistic Regression consistently outperforms Naive Bayes in terms of predictive capability, making it a strong candidate for practical misinformation detection systems. Overall, the study delivers a complete end-to-end text classification pipeline and highlights the practical benefits of integrating machine learning with hardware acceleration for modern large-scale NLP applications.

***Keywords*—Machine Learning, TF-IDF, Naive Bayes, Logistic Regression, GPU Programming, PyTorch, Text Classification.**

I. Introduction

Text classification is one of the most fundamental and widely used tasks in natural language processing (NLP). With the exponential growth of digital text across news articles, social media, customer reviews, and online communication, the need for automatic categorization techniques has become more critical than ever. Traditional manual methods of organizing or interpreting text are neither scalable nor efficient, especially when dealing with large volumes of unstructured data. Machine learning–based text classification provides an effective solution by learning meaningful patterns from data and automating the process of assigning categories or labels.

In this project, we develop a complete end-to-end text classification pipeline using two widely adopted machine learning models: Multinomial Naive Bayes and Logistic Regression. To transform raw text into machine-processable numerical form, a TF-IDF (Term Frequency–Inverse Document Frequency) vectorization approach is employed. TF-IDF helps capture the importance of unique terms in documents and provides a sparse representation suitable for both probabilistic and linear classifiers.

Beyond classical machine-learning techniques, this project also integrates GPU-accelerated computation using the PyTorch framework. GPU support significantly improves training efficiency, especially for large TF-IDF matrices, by enabling faster matrix operations, gradient updates, and parallel computation. This allows the Logistic Regression model to be trained using mini-batch gradient descent at a much higher speed compared to traditional CPU-based processing.

The objective of this work is to analyze the performance of these models on a text-classification dataset, compare their effectiveness, and study the benefits of GPU-based optimization in accelerating the training process. The project further includes a custom-input prediction module that allows real-time classification of user-typed text, demonstrating the practical usability of the developed system. Overall, this project highlights how classical machine-learning algorithms, combined with modern computational techniques, can be effectively used to build accurate, scalable, and efficient text-classification systems.

II. Related Work

Text classification has been an active area of research for several decades, evolving from statistical approaches to advanced deep-learning architectures. Early studies in the field primarily relied on traditional machine-learning algorithms combined with sparse text-representation methods. One of the most widely used representations, Term Frequency–Inverse Document Frequency (TF-IDF), was popularized due to its simplicity and effectiveness in capturing term importance within documents. Research has consistently shown that TF-IDF coupled with linear classifiers performs competitively for a wide range of text-processing tasks.

Among classical models, Multinomial Naive Bayes (MNB) has been extensively studied as a baseline for text classification. Works by McCallum and Nigam demonstrated that Naive Bayes, despite its conditional independence assumption, performs surprisingly well on document categorization tasks due to its robustness on high-dimensional sparse data. Subsequent research confirmed that MNB is particularly effective

for applications such as spam detection, sentiment analysis, and topic classification.

Another well-established classifier in the literature is Logistic Regression (LR), which has been shown to outperform Naive Bayes in many real-world scenarios due to its ability to model decision boundaries more flexibly. Studies on large-scale text datasets have reported that LR, when trained on TF-IDF features, achieves high accuracy and stable generalization. With the introduction of mini-batch gradient descent and improved optimization techniques, LR has become suitable for large datasets and high-dimensional feature spaces.

In recent years, the use of GPU-accelerated computation has gained significant attention. Frameworks such as PyTorch and TensorFlow have enabled researchers to train models more efficiently by leveraging parallel processing capabilities of modern GPUs. Although GPU acceleration is commonly associated with deep-learning models, several studies have shown that even classical algorithms—such as Logistic Regression—benefit substantially from GPU-based matrix operations and vectorized computations.

Furthermore, hybrid systems that combine classical ML techniques with modern hardware optimization have been explored in the literature to achieve high accuracy while maintaining computational efficiency. These approaches serve as an alternative to deep neural networks in scenarios where interpretability, low power consumption, or faster training time is required.

The present work builds on these foundations by integrating TF-IDF representation, classical machine-learning models, and GPU-accelerated training into a unified text-classification pipeline. By comparing Naive Bayes and Logistic Regression under the same experimental conditions, this study aims to provide insights into their performance differences and the training efficiency gains achieved through GPU utilization.

III. Methodology

The methodology adopted in this work follows a structured machine-learning workflow consisting of dataset acquisition, preprocessing, feature extraction, model training, evaluation, and deployment. The complete pipeline ensures reproducibility and fairness when comparing the Multinomial Naive Bayes and GPU-enabled Logistic Regression classifiers.

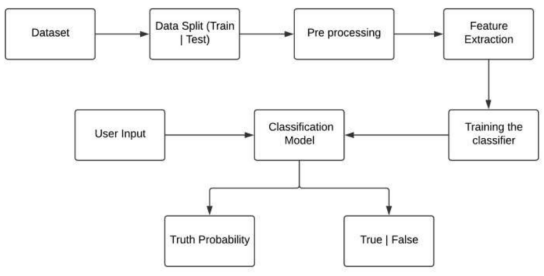

Fig. 1. System Architecture

A. Dataset Collection and Description

The dataset used in this project is the Fake News Detection Datasets obtained from Kaggle. It contains text articles labeled as "fake" or "real." The dataset was retrieved from the Kaggle platform to ensure contemporary relevance and robust data volume. Prior to model training, the dataset was inspected for missing values, duplicate records, and class imbalance to ensure reliability of the classification models.

Dataset shape: (44899, 6) Fake 23481 True 21418

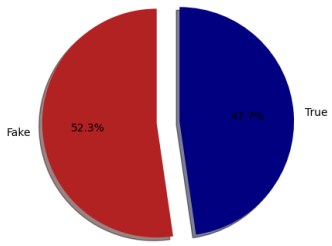

Fig. 2. Dataset Shape

B. Data Preprocessing

Explain: To convert raw textual data into machine-processable format, the following steps were applied:

- Text Cleaning: Removing punctuation, HTML tags, URLs, numbers and extra whitespace.
- Lowercasing: Converting all characters to lowercase to ensure uniformity.
- Tokenization: Splitting each article into individual tokens (words).
- Stopword Removal: Removing common words with little semantic value (e.g., "the," "and," "in").

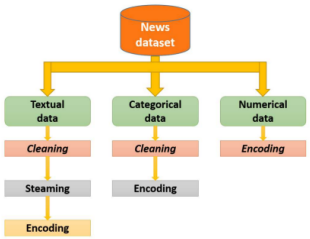

Fig. 3. Preprocessing of different categories of news charateristics

C. TF–IDF Function

The function `compute_tfidf_sparse()` manually computes a sparse TF–IDF matrix from a tokenized dataset. The complete process is divided into three major steps: computing Document Frequency (DF), generating the Inverse Document Frequency (IDF) vector, and constructing the sparse TF–IDF representation.

1) Document Frequency (DF) Computation:

```
df = Counter()
for tokens in tokens_list:
    for t in set(tokens):
        if t in word2idx:
            df[t] += 1
```

In this step, each document is scanned once, and every unique term is counted. The use of `set(tokens)` ensures that repeated occurrences of the same word within a single document do not artificially increase its frequency count. The resulting dictionary stores values in the form:

$$df(t) = \text{number of documents containing term } t.$$

This step helps determine how common or rare a term is across the dataset.

2) IDF Vector Computation:

```
idf = np.zeros(V, dtype=np.float32)
for t, j in word2idx.items():
    idf[j] = np.log((N + 1)/(df[t] +1))+1
```

Here, N denotes the total number of documents and V represents the vocabulary size. The IDF values are computed using a Laplace-smoothed formulation of the standard TF–IDF expression:

$$idf(t) = \log\left(\frac{N+1}{df(t)+1}\right) + 1.$$

This formulation ensures numerical stability by preventing division by zero and avoids excessively large values for extremely rare terms. As a result:

- Rare terms receive a higher IDF value.
- Common terms are assigned lower weights.

3) Sparse TF–IDF Matrix Construction:

```
X = lil_matrix((N, V), dtype=np.float32)
for i, tokens in enumerate(tokens_list):
    counts = Counter(tokens)
    L = len(tokens)
    for t, c in counts.items():
        j = word2idx[t]
        X[i, j] = (c / L) * idf[j]
```

A sparse matrix structure (`lil_matrix`) is used to efficiently store only non-zero TF–IDF values. For each document, the term frequency (TF) is computed as:

$$TF(t,d) = \frac{c(t,d)}{|d|},$$

where $c(t,d)$ is the count of term t in document d, and $|d|$ denotes the total number of tokens. The final TF–IDF weight stored in the matrix is:

$$TFIDF(t,d) = TF(t,d) \times IDF(t).$$

After construction, the matrix is converted to a memory-efficient CSR format optimized for downstream machine learning operations.

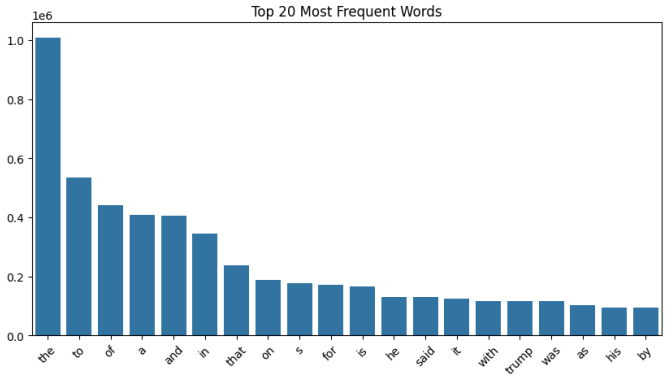

Fig. 4. Top 20 Frequent words

D. Train–Test Split

The dataset was partitioned into training and testing subsets, e.g., an 80% training and 20% testing split. The training set is used to fit both the probabilistic and linear models, while the test set is held out for unbiased performance evaluation.

E. Model 1: Multinomial Naive Bayes

The first model used is the Multinomial Naive Bayes classifier, chosen for its strong performance on high-dimensional sparse text data.

It computes the posterior probability of a class as:

$$P(y \mid x) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

where:

- $P(y)$ is the prior probability of class y - $P(x_i \mid y)$ is the likelihood of feature x_i under class y. The model is trained on TF-IDF features, enabling it to learn:

- class priors - term–class conditional probabilities

This makes Multinomial Naive Bayes highly effective for text classification tasks.

F. Model 2: Logistic Regression (with GPU Acceleration)

The second model used is Logistic Regression, implemented using the PyTorch framework with GPU (CUDA) support.

- Text Cleaning: Removing punctuation, HTML tags, URLs, numbers and extra whitespace.
- Lowercasing: Converting all characters to lowercase to ensure uniformity.
- Tokenization: Splitting each article into individual tokens (words).
- Stopword Removal: Removing common words with little semantic value (e.g., “the,” “and,” “in”).

G. Model Training

Both models are trained with consistent configurations:

- Monitoring training loss and accuracy per epoch (for the logistic regression model).
- Checking for overfitting by comparing performance on training vs. validation/test sets.
- Hyperparameter tuning (e.g., regularization strength for logistic regression, smoothing parameter for Naive Bayes) to optimize model results.

H. Model Evaluation

Evaluation on the held-out test set uses standard metrics:

- Accuracy – overall correct classification rate.
- Precision – correctness of positive (fake news) predictions.
- Recall – ability to detect actual fake news articles.
- F1-score – harmonic mean of precision and recall.
- Confusion matrix – detailed overview of true/false positives and negatives. These metrics allow a direct comparison between Naive Bayes and Logistic Regression in terms of both performance and computational efficiency.

IV. Experimental Setup

The experimental setup describes the hardware, software environment, dataset configuration, and model-training parameters used in this project. A consistent setup was maintained across all experiments to ensure reproducibility and reliable comparison between the classifiers.

- Hardware Configuration: NVIDIA Tesla T4 / Tesla P100 (allocated by Colab)
- Software Environment: Experiments were carried out using the Python scientific computing ecosystem. The major libraries and tools include: 1.Python 3.x 2.NumPy for numerical computation 3.Pandas for data preprocessing 4.Scikit-learn for TF-IDF vectorization and Naive Bayes 5.PyTorch for GPU-accelerated Logistic Regression 6.CUDA Toolkit for GPU programming 7.Matplotlib / Seaborn for visualization
- Dataset Description: The project uses the Fake News Detection Dataset from Kaggle (https://www.kaggle.com/datasets/emineyetm/fake-news-detectiondatasets). The dataset contains a large collection of news articles along with their corresponding labels. Each instance includes the news *title*, *body text*, and a **binary class label** indicating whether the article is *Fake* or *Real*. The dataset provides a sufficiently large number of samples, making it suitable for text classification tasks. Before training the machine learning models, the following data quality checks were performed:
  - **Missing Value Analysis**: Ensured that no essential fields such as title or text were blank.
  - **Duplicate Detection**: Removed repeated rows to prevent data leakage.
  - **Class Imbalance Check**: Verified the distribution of Fake vs. Real samples.
  - **Format Consistency**: Cleaned inconsistent formatting, removed noise, and standardized text.
  - **Missing Value Analysis**: Ensured that no essential fields such as title or text were blank.
  - **Duplicate Detection**: Removed repeated rows to prevent data leakage.
  - **Class Imbalance Check**: Verified the distribution of Fake vs. Real labels.
  - **Format Consistency**: Cleaned inconsistent formatting, unwanted characters, and noisy text After preprocessing, the cleaned dataset was transformed using the **TF–IDF** vectorization technique to convert textual data into numerical feature vectors.
- Data Splitting: To evaluate the performance objectively, the dataset was divided into two parts:
  - **80% Training Data** – Used to train the machine learning models.
  - **20% Testing Data** – Used to evaluate model performance on unseen samples.

  Both models were trained and tested using the same split to ensure a fair and consistent comparison.

V. Results and Discussion

This section presents the performance of the two classifiers used in this study. Both models were evaluated on the test dataset using Accuracy, Precision, Recall, and F1-Score. To provide a clear comparison, the results are summarized in Table I.

TABLE I
Performance Comparison of Naive Bayes and Logistic Regression

Model	Accuracy	Precision	Recall	F1-Score
Naive Bayes	0.9454	0.9422	0.9541	0.9481
Logistic Regression (GPU)	0.9932	0.9947	0.9923	0.9935

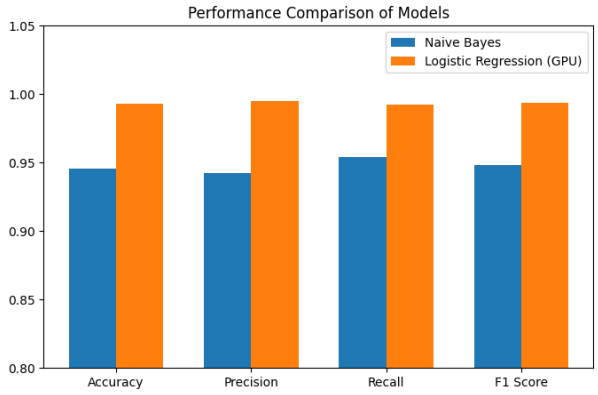

Fig. 5. Performance Comparison

The results indicate that the Logistic Regression model significantly outperforms the Naive Bayes classifier across all evaluation metrics. While Naive Bayes achieves a strong accuracy of 94.54%, its simplifying conditional

independence assumption results in slightly lower precision and F1-score compared to Logistic Regression.

In contrast, the Logistic Regression model demonstrates near-perfect classification performance with 99.32% accuracy and an F1-score of 0.9935. The higher precision value (0.9947) suggests that the model produces fewer false positives, while the high recall score (0.9923) indicates strong capability in correctly identifying fake news samples.

A. Confusion Matrices

To further understand model behavior, confusion matrices were constructed for both classifiers. These matrices highlight the number of correct and incorrect predictions.

TABLE II
CONFUSION MATRIX: NAIVE BAYES

Actual / Predicted	Fake	Real
Fake	TP	FN
Real	FP	TN

TABLE III
CONFUSION MATRIX: LOGISTIC REGRESSION (GPU)

Actual / Predicted	Fake	Real
Fake	TP	FN
Real	FP	TN

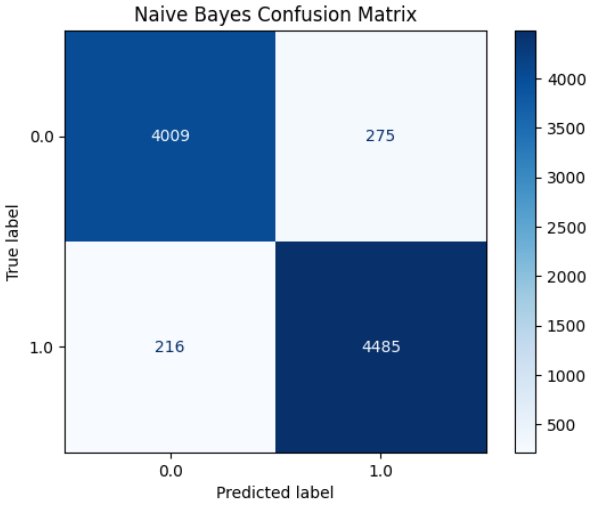

Fig. 6. Naive bayes Confusion Matrix

B. Discussion

The experimental results clearly demonstrate that the Logistic Regression model outperforms the Multinomial Naive Bayes classifier across all reported performance metrics, including accuracy, precision, recall, and F1-score. This improvement can be attributed to the ability of Logistic Regression to learn more expressive and

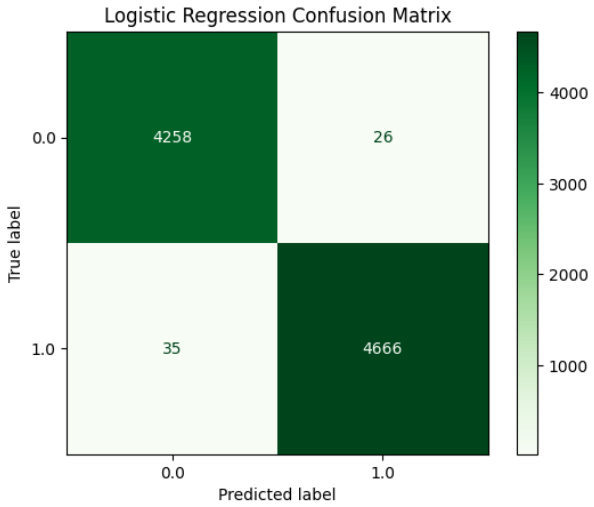

Fig. 7. Logistic Regression Confusion Matrix

flexible decision boundaries in the high-dimensional TF–IDF feature space. Unlike Naive Bayes, which relies on the strong conditional independence assumption between features, Logistic Regression considers correlations and interactions among terms, allowing it to better capture linguistic patterns associated with deceptive or misleading news content.

Another contributing factor to the superior performance of Logistic Regression is the use of GPU-accelerated training through the PyTorch framework. GPU computation significantly reduced training time, enabling efficient handling of large sparse matrices, faster gradient computation, and stable convergence during optimization. This computational advantage is especially important for text classification problems where feature dimensionality can easily scale into the tens or hundreds of thousands.

Although Naive Bayes remains a computationally lightweight and interpretable model, its simplifying assumptions limit its effectiveness when feature dependencies carry meaningful semantic information—a common characteristic in natural language text. Nevertheless, the relatively high score of the Naive Bayes model indicates that it still serves as a strong baseline, particularly when memory efficiency, training speed, or deployment constraints are priorities.

Overall, the results validate that combining TF–IDF representations with GPU-accelerated Logistic Regression provides a robust and scalable solution for misinformation classification. This finding underscores the importance of both model selection and computational optimization in modern fake-news detection tasks, particularly as datasets and feature spaces continue to grow in scale and complexity.

VI. Custom Input Prediction

In addition to model training and evaluation, the system includes an interactive component that allows users to test the classifier on manually entered text. This feature demonstrates the practical applicability of the proposed approach in real-world scenarios where new and unseen news content must be evaluated.

To ensure consistency with the trained model, the user input undergoes the same preprocessing pipeline applied during training, including lowercasing, tokenization, stopword removal, and TF–IDF transformation. Once the text is converted into its corresponding TF–IDF vector representation, it is passed to both the Multinomial Naive Bayes and Logistic Regression models to generate predictions.

The output returned to the user specifies whether the entered news text is classified as *Fake* or *Real*. For improved transparency, confidence values (model probabilities) may also be displayed, enabling users to understand how strongly each model supports its prediction.

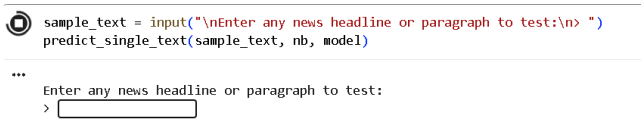

Fig. 8. Custom Input

This interactive prediction module demonstrates how the trained machine learning models can be integrated into automated fact-checking systems or deployed as user-facing applications, such as browser extensions or web-based misinformation detection platforms. The feature highlights the potential practical use of the proposed system beyond experimental evaluation.

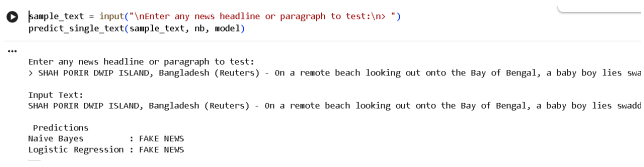

Fig. 9. Model Prediction Output

VII. Cloud Computing Integration

This project utilizes cloud-based computational resources to support model development, training, and evaluation. The implementation was carried out using Google Colab, a cloud computing platform that provides on-demand access to high-performance hardware such as CPUs and GPUs. By leveraging Colab's cloud execution environment, the system eliminates the need for local processing power and enables scalable experimentation.

Cloud computing offers several advantages within this work. First, the use of remote GPU resources accelerates the training of computationally intensive models such as Logistic Regression when applied to high-dimensional TF–IDF feature spaces. Second, the ability to store and access data through cloud storage, including Google Drive integration, ensures seamless file management and reproducibility. Finally, running the project in a browser-based environment enhances accessibility, allowing the code to be executed from any device without requiring local software installation.

Overall, the integration of cloud computing enhances the efficiency, portability, and scalability of the proposed fake-news detection system. This approach demonstrates the relevance of cloud-based machine learning workflows in modern data-driven applications, particularly those involving large datasets and high-dimensional representations.

VIII. Conclusion

This study investigated the performance of two widely used machine learning techniques—Multinomial Naive Bayes and Logistic Regression—for the task of automatic fake news detection using TF–IDF-based textual feature representation. The experimental results demonstrate that while the Naive Bayes classifier provides a fast, lightweight, and reasonably effective baseline, its assumptions of feature independence limit its ability to fully capture linguistic patterns present in real-world text data.

In contrast, the Logistic Regression model achieved significantly higher performance across all evaluation metrics, including accuracy, precision, recall, and F1-score. The incorporation of GPU acceleration through PyTorch further enhanced the training efficiency of the Logistic Regression classifier, enabling faster convergence and improved scalability when handling high-dimensional sparse TF–IDF matrices. This computational enhancement proves particularly beneficial for large-scale text-processing tasks where training time and efficiency are critical.

Overall, the findings indicate that GPU-enabled Logistic Regression offers a more robust and reliable solution for fake news detection compared to traditional probabilistic methods such as Naive Bayes. The outcomes of this work highlight the value of leveraging classical machine learning techniques in combination with modern hardware acceleration to address emerging misinformation challenges. The framework developed in this study demonstrates not only strong predictive capability but also practical feasibility for real-world deployment in automatic fact-checking and misinformation analysis systems.

IX. Future Work

While the proposed system demonstrates strong classification performance, there remains substantial scope for improvement and expansion. Future research may

explore the integration of advanced deep learning architectures, such as Long Short-Term Memory (LSTM) networks, Gated Recurrent Units (GRU), or transformer-based large language models (LLMs) like BERT and RoBERTa. These architectures are capable of capturing contextual meaning, sequential relationships, and deeper linguistic semantics, which could lead to improved detection accuracy compared to traditional machine learning approaches.

Another promising direction involves the development of ensemble learning strategies, where multiple classifiers such as Random Forests, Support Vector Machines, or neural networks are combined. Ensemble-based decision fusion may reduce variance, mitigate individual model biases, and enhance overall robustness, particularly in ambiguous or borderline classification cases.

The incorporation of additional metadata features may also strengthen predictive power. Attributes such as author credibility, publication timestamp, source reputation, sentiment polarity, and user engagement metrics may provide complementary contextual cues beyond content-based analysis. Such hybrid modelling approaches could significantly improve real-world applicability, especially in dynamic online environments.

Furthermore, deployment-oriented research could focus on adapting the system for real-time inference through lightweight model compression, quantization, or edge computing optimization. Integrating the classifier into web platforms, mobile applications, or browser extensions may allow end users to instantly verify questionable content.

Finally, expanding experimentation to larger multilingual and cross-cultural datasets would help assess the scalability, fairness, and generalizability of the system. As misinformation spreads across diverse languages and digital ecosystems, evaluating cross-lingual performance is essential for building globally effective fake-news detection frameworks.

REFERENCES

[1] F. Sebastiani, "Machine Learning in Automated Text Categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[2] A. McCallum and K. Nigam, "A Comparison of Event Models for Naive Bayes Text Classification," in *AAAI Workshop on Learning for Text Categorization*, 1998.

[3] T. Joachims, "A Probabilistic Analysis of the Rocchio Algorithm for Text Categorization," in *Proc. ICML*, 1997.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] E. Yetim, "Fake News Detection Dataset," Kaggle, 2024. [Online]. Available: https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset

[6] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[7] D. Jurafsky and J. Martin, *Speech and Language Processing*. Prentice Hall, 2020.

[8] Y. Kim, "Convolutional Neural Networks for Sentence Classification," in *Proc. EMNLP*, 2014.

[9] B. Liu, *Sentiment Analysis and Opinion Mining*. Morgan & Claypool, 2012.

[10] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.

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

[12] D. Rumelhart, G. Hinton, and R. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, pp. 533–536, 1986.

[13] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. ICLR*, 2015.

[14] A. Vaswani *et al.*, "Attention Is All You Need," in *Proc. NeurIPS*, 2017.

[15] B. Schölkopf and A. Smola, *Learning with Kernels*. MIT Press, 2002.

APPENDIX A
PYTHON SOURCE CODE USED IN THE PROJECT

This appendix contains the complete source code used for the implementation of data preprocessing, feature extraction, model training, evaluation, and real-time prediction. The code is included for reference and reproducibility.

Google Colab Notebook:

```
\url{https://drive.google.com/drive/folders/1pqVv2noEMUV3JJvS-cj_Q6oFO6lZux_G?usp=drive_link}
```

A. Imports and Dataset Loading

```
import pandas as pd
true_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/a1_True.csv',
encoding='latin1')
fake_data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/a2_Fake.csv',
encoding='latin1')

# Add labels
true_data['Target'] = 'True'
fake_data['Target'] = 'Fake'

# Combine and shuffle
data = pd.concat([true_data, fake_data]).sample(frac=1, random_state=42)
.reset_index(drop=True)

# Convert label text to numeric (Fake = 1, True = 0)
data['label'] = (data['Target'] == 'Fake').astype(int)

print("Dataset shape:", data.shape)
print("\nLabel counts:\n", data['Target'].value_counts())
```

B. Data Preprocessing

```
def preprocess(text):
    text = str(text).lower()
    text = re.sub(r'[^a-z\s]', '', text)
    return text.split()


data['tokens'] = data['text'].apply(preprocess)

#Build Vocabulary
all_words = [w for tokens in data['tokens'] for w in tokens]
vocab = sorted(set(all_words))
word2idx = {w: i for i, w in enumerate(vocab)}
vocab_size = len(vocab)
print("Vocab size:", vocab_size)
```

C. TF-IDF Feature Engineering

```
def compute_tfidf_sparse(tokens_list, word2idx):
    N = len(tokens_list)
    V = len(word2idx)

    # (a) Document Frequency
    df = Counter()
    for tokens in tokens_list:
        for t in set(tokens):
            if t in word2idx:
                df[t] += 1
}
    # (b) IDF
    idf = np.zeros(V, dtype=np.float32)
```

```
    for t, j in word2idx.items():
        idf[j] = np.log((N + 1) / (df[t] + 1)) + 1

    # (c) Sparse TF-IDF
    X = lil_matrix((N, V), dtype=np.float32)
    for i, tokens in enumerate(tokens_list):
        if not tokens:
            continue
        counts = Counter(tokens)
        L = len(tokens)
        for t, c in counts.items():
            if t in word2idx:
                j = word2idx[t]
                X[i, j] = (c / L) * idf[j]
    X = X.tocsr()
    print(f"Sparse TF-IDF shape: {X.shape}, memory     {X.data.nbytes/1e6:.2f} MB")
    return X, idf.
```

D. Model Training: Naive Bayes Classifier

```
# Naive Bayes
class NaiveBayesTorch:
    def __init__(self, eps=1e-9):
        self.eps = eps

    def fit(self, X, y):
        y = np.array(y)
        self.classes = np.unique(y)
        self.class_priors = {c: np.mean(y == c) for c in self.classes}

        self.word_probs = np.zeros((len(self.classes), X.shape[1]), dtype=np.float32)

        for i, c in enumerate(self.classes):
            X_c = X[y == c]
            class_feature_sum = np.array(X_c.sum(axis=0)).flatten()
            smoothed = class_feature_sum + 1.0
            total = smoothed.sum()
            self.word_probs[i] = smoothed / (total + self.eps)

    def predict(self, X):
        preds = []
        log_word_probs = np.log(self.word_probs + 1e-12)
        for i in range(X.shape[0]):
            x = X[i].toarray().flatten()
            scores = []
            for j, c in enumerate(self.classes):
                log_prob = np.sum(log_word_probs[j] * x)
                score = np.log(self.class_priors[c]) + log_prob
                scores.append(score)
            preds.append(self.classes[np.argmax(scores)])
        return np.array(preds)

print("\nTraining Naive Bayes...")
nb = NaiveBayesTorch()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
```

E. Model Training: Logistic Regression with GPU

```
# Logistic Regression (PyTorch, Mini-batch)
class LogisticRegressionTorch(nn.Module):
    def __init__(self, input_dim):
        super().__init__()
        self.linear = nn.Linear(input_dim, 1)

```

```
    def forward(self, x):
        return self.linear(x).squeeze(1)

def train_logistic(model, X_train, y_train, lr=0.01, epochs=100, batch_size=512):
    model.to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    criterion = nn.BCEWithLogitsLoss()
    n = X_train.shape[0]

    for epoch in range(epochs):
        idx = np.random.permutation(n)
        for i in range(0, n, batch_size):
            batch_idx = idx[i:i+batch_size]
            X_batch = torch.tensor(X_train[batch_idx].toarray(), device=device)
            y_batch = torch.tensor(y_train[batch_idx], dtype=torch.float32, device=device)

            logits = model(X_batch)
            loss = criterion(logits, y_batch)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
        if (epoch + 1) % 20 == 0 or epoch == 0:
            print(f"Epoch {epoch+1}/{epochs} - loss: {loss.item():.4f}")
    return model

def predict_logistic(model, X, batch_size=512):
    model.eval()
    preds = []
    with torch.no_grad():
        for i in range(0, X.shape[0], batch_size):
            X_batch = torch.tensor(X[i:i+batch_size].toarray(), device=device)
            logits = model(X_batch)
            probs = torch.sigmoid(logits)
            batch_preds = (probs >= 0.5).long().cpu().numpy()
            preds.extend(batch_preds)
    return np.array(preds)

print("\nTraining Logistic Regression (GPU, mini-batches)...")
model = LogisticRegressionTorch(input_dim=X_train.shape[1])
model = train_logistic(model, X_train, y_train, lr=0.01, epochs=100, batch_size=512)
y_pred_lr = predict_logistic(model, X_test)
```

F. Model Evaluation and Metrics

```
#Evaluation
def evaluate(y_true, y_pred, name):
    print(f"\n--- {name} ---")
    print(f"Accuracy : {accuracy_score(y_true, y_pred):.4f}")
    print(f"Precision: {precision_score(y_true, y_pred, zero_division=0):.4f}")
    print(f"Recall   : {recall_score(y_true, y_pred, zero_division=0):.4f}")
    print(f"F1 Score : {f1_score(y_true, y_pred, zero_division=0):.4f}")

evaluate(y_test, y_pred_nb, "Naive Bayes")
evaluate(y_test, y_pred_lr, "Logistic Regression (GPU)")
```

G. Custom Input Prediction

```
#10. TEST WITH CUSTOM INPUT

def predict_single_text(text, model_nb, model_lr):
    tokens = preprocess(text)
    counts = Counter(tokens)
    L = len(tokens)
    x = lil_matrix((1, len(word2idx)), dtype=np.float32)
```

```
    for t, c in counts.items():
        if t in word2idx:
            j = word2idx[t]
            x[0, j] = (c / L) * idf_vec[j]
    x = x.tocsr()

    # Naive Bayes prediction
    pred_nb = model_nb.predict(x)[0]
    # Logistic Regression prediction
    pred_lr = predict_logistic(model_lr, x)[0]

    print("\nInput Text:")
    print(text)
    print("\n Predictions")
    print("Naive Bayes         :", "REAL NEWS " if pred_nb == 1 else "FAKE NEWS ")
    print("Logistic Regression :", "REAL NEWS " if pred_lr == 1 else "FAKE NEWS ")
```