VeriFact AI - A Website for fake news detection

Mini project design report submitted to CUSAT in partial fulfilment of the

requirements for the award for the degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE & ENGINEERING

Submitted by,

Pratyush Kumar Thakur (20222553)

Ayush Kumar (20222526)

Ayush Anand (20222525)

Supriyam Raj (20222562)

Mohit Raj (20222545)

Under the guidance of

Dr. Preetha Mathew K
Head of the Department, Division of CSE

Mrs. Alice Joseph
Assistant Professor, Division of CSE

Miss. Anukartha V.R
Assistant Professor, Division of CSE

Division of Computer Science & Engineering

COCHIN UNIVERSITY COLLEGE OF ENGINEERING KUTTANAD,

ALAPPUZHA

APRIL 2025

Division of Computer Science & Engineering

COCHIN UNIVERSITY COLLEGE OF ENGINEERING KUTTANAD, ALAPPUZHA

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"VeriFact AI - A Website for fake news detection"** has been submitted by **Pratyush Kumar Thakur (20222553), Ayush Kumar (20222526), Ayush Anand (20222525), Supriyam Raj (20222562) and Mohit Raj (20222545)** in partial fulfilment of the requirements for the award of the Degree B.Tech in COMPUTER SCIENCE AND ENGINEERING of COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

…….…… …...…….

Miss. Anukartha V.R
Project Coordinator

Mrs. Alice Joseph
Project Coordinator

……..…….

Dr. Preetha Mathew K

Head of the Dept.

Division of CSE

ACKNOWLEDGEMENT

Efforts have been taken by us in this project however it would not have been possible without the kind of support and help of many individuals and organizations. Thanks to our respected principal, Dr. Asaletha R, for the facilities provided by him during the preparation of this report. We also express our gratitude towards all the staff members of the Computer Science and Engineering department and faculties of CUCEK for their guidance, constant supervision, and encouragement. We express our sincere thanks to Mrs. Alice Joseph, Assistant Professor and Miss. Anukartha V.R, Assistant Professor, Dept. of Computer Science and Engineering as well as our Head of the Department, Dr. Preetha Mathew K for giving us innovative suggestions, timely advice, correction, and suggestions during this endeavor.

Pratyush Kumar Thakur (20222553)

Ayush Kumar (20222526)

Ayush Anand (20222525)

Supriyam Raj (20222562)

Mohit Raj (20222545)

ABSTRACT

Verifact AI is a web application designed for fake news detection, utilizing advanced technologies to ensure accurate verification. The frontend is developed using React.js, offering a dynamic and user-friendly interface that enables seamless interaction with the platform. The backend is primarily built with Express.js and Node.js for efficient request handling and API integration, while Flask (Python) is incorporated to implement and serve the machine learning models responsible for analyzing and classifying news articles based on credibility.

By leveraging AI algorithms and Natural Language Processing (NLP), the system can identify patterns of misinformation and help users detect unreliable sources. It processes large volumes of data rapidly, enabling real-time verification of news content. This combination of technologies enhances the accuracy and reliability of fake news detection.

For data storage, MongoDB is used to manage both structured and unstructured data, including news articles, user interactions, and verification outcomes. The system is designed to learn continuously from new inputs, improving its prediction capabilities over time. Users can submit text or URLs, and the platform provides a credibility score along with references to trusted sources.

By integrating AI-powered analysis with a responsive web application, Verifact AI aims to actively combat the spread of misinformation. The platform serves as a valuable tool for journalists, researchers, and the general public, promoting a more informed and responsible digital media environment through accessible and intelligent news verification.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 GENERAL INTRODUCTION

In the modern digital landscape, the rapid spread of misinformation has become a growing concern, influencing public perception and decision-making. Fake news, often designed to manipulate opinions, spreads quickly through social media and online platforms, making it difficult for users to distinguish between factual and misleading information. This has led to a pressing need for reliable fact-checking solutions that can help identify false content. Verifact AI is a web-based platform developed to address this challenge by providing real-time news verification and analysis. It helps users assess the credibility of news articles, reducing the risk of misinformation.

The primary goal of Verifact AI is to provide an accessible and effective tool for fact-checking. The platform analyzes news content by examining language patterns, source credibility, and cross-referencing with verified information. By using advanced algorithms, it detects misleading narratives and helps users make informed decisions. The system provides instant feedback, allowing users to verify news quickly and easily as Shown in Figure 8.4. This ensures that false or manipulated information is identified before it spreads further. An overview of the Verifact AI homepage is shown in Figure 8.1, illustrating the platform's user-friendly interface and core features.

Misinformation can have serious consequences, including social unrest, financial losses, and political instability. The rise of deepfake technology and AI-generated content has made it even harder to differentiate between real and fake news. Verifact AI plays a crucial role in combating this issue by offering a reliable and data-driven solution. By empowering individuals with accurate information, the platform helps promote transparency and trust in digital media.

Verifact AI is designed for a wide range of users, including journalists, researchers, and the general public. It aims to create a more informed society by encouraging responsible news consumption. As misinformation continues to evolve, tools like Verifact AI become essential in maintaining the integrity of online information. By providing a user-friendly and efficient way to verify news, the platform contributes to reducing the spread of false narratives and ensuring that accurate information reaches the public.

1.2 OBJECTIVES AND ADVANTAGES

The primary objective of VeriFact AI is to develop a reliable and efficient web platform that allows users to detect and verify fake news in real time. By leveraging advanced Natural Language Processing (NLP) techniques through the BERT model, the system aims to accurately classify news content as either real or fake. The project also focuses on creating a

user-friendly interface using React, ensuring that users of all technical backgrounds can interact with the platform effortlessly. On the backend, technologies such as Flask, and Express are employed to handle user requests, manage data flow, and perform machine learning inference. Another significant objective is to store and manage news-related data efficiently using MongoDB, which supports scalability and fast access to unstructured information. Additionally, the platform is intended to serve as an educational tool to promote media literacy and awareness around the spread of misinformation.

VeriFact AI provides a practical solution to one of the major challenges of the digital age — the spread of misinformation. It enables users to quickly and easily verify the authenticity of news content, helping them make informed decisions. The platform's ability to deliver accurate and timely feedback enhances user trust and reliability. Its intuitive interface ensures that individuals with varying levels of technical expertise can use the system without difficulty. The project also supports scalability, allowing it to handle a growing number of users and data inputs effectively. Furthermore, VeriFact AI promotes critical thinking and awareness among users by encouraging them to evaluate news sources more carefully. This not only helps in reducing the impact of fake news but also contributes to a more informed and responsible digital community. The system's design supports future integration with other platforms, making it adaptable and relevant for long-term use.

1.3 PROBLEM STATEMENT

In the era of digital communication, the internet has become a primary source of information for people across the globe. However, this convenience has also led to the rapid dissemination of misinformation and fake news, which poses a significant threat to individuals, communities, and even entire nations. Fake news can influence public perception, manipulate political outcomes, incite social unrest, and damage reputations. Social media platforms, blogs, and unregulated news websites often act as breeding grounds for such misleading content, making it increasingly difficult for users to distinguish between what is real and what is not.

Despite the growing awareness of this issue, most users do not have the tools or expertise to verify the authenticity of news themselves. Existing solutions are either too complex, not user-friendly, or fail to provide accurate results in real-time. Manual fact-checking, while effective, is not scalable or practical for everyday use. Moreover, the psychological tendency of individuals to trust information that aligns with their beliefs further amplifies the problem.

This project addresses the critical need for an automated, accessible, and intelligent platform that assists users in verifying news content quickly and accurately. VeriFact AI aims to fill this gap by offering a web-based solution that utilizes natural language processing and machine learning to detect fake news. The goal is to create a tool that is not only technically robust but also easy to use, promoting digital responsibility and helping to combat the spread of false information on a large scale

CHAPTER 2

EXISTING SYSTEMS

2.1 LITERATURE SURVEY

2.1.1. Introduction

The proliferation of fake news across digital platforms has emerged as a critical societal challenge. With billions of users consuming information online, misinformation can spread rapidly, influencing public opinion and even impacting democratic processes. The growing volume of digital content necessitates intelligent systems capable of detecting and mitigating fake news. Verifact AI, a system built to classify news as fake or real and support real-time user feedback and retraining, relies on modern machine learning (ML) and natural language processing (NLP) techniques. This literature survey explores the evolution and capabilities of such AI-based fake news detection systems.

2.1.2. Machine Learning Approaches for Fake News Detection

Traditional machine learning algorithms have been pivotal in early fake news detection research. Models such as Naïve Bayes, Support Vector Machines (SVMs), and Decision Trees analyze linguistic features, sentiment, and metadata from news articles to classify them as real or fake.

K. Shu et al. [1] highlight that ML-based classifiers rely heavily on hand-crafted features extracted from text such as n-grams, POS tagging, and syntactic structures. While these models are relatively lightweight and fast, they often lack contextual understanding and fail to detect more sophisticated forms of misinformation.

Rashkin et al. [2] conducted experiments on various ML models and found that although they can detect blatantly fake content, their performance drops with more nuanced or ambiguous language. These limitations prompted the shift toward more complex, deep learning models.

2.1.3. Deep Learning and NLP-Based Techniques

Deep learning has revolutionized natural language understanding, offering new tools for fake news detection. Recurrent neural networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, capture sequential dependencies in text, improving classification performance. LSTM models outperform traditional ML in capturing semantic patterns but require substantial labeled data [3].

Recent advances include transformer-based models like BERT (Bidirectional Encoder Representations from Transformers). BERT has shown state-of-the-art performance in various NLP tasks, including fake news classification [4]. Its attention mechanism allows it to consider both the context and semantics of entire sentences simultaneously.

However, deep learning models are computationally intensive and less interpretable. They also suffer from data drift over time — as fake news evolves, older training datasets may no longer represent current misinformation patterns.

In Verifact AI, BERT-based classification is applied in the backend to analyze user-submitted news and generate a prediction with a confidence score. This real-time analysis ensures responsiveness while maintaining high accuracy.

2.1.4. Role of Fact-Checking, Knowledge Graphs, and Human Feedback

Automated fact-checking systems go beyond classification to verify claims against authoritative databases. Knowledge graphs (KGs), like Wikidata and DBpedia, help cross-verify factual assertions by mapping entities and their relationships [5].

Fact-checking systems also integrate external sources like ClaimReview, which aggregates information from verified fact-checking organizations. These hybrid systems significantly improve detection accuracy, particularly in combination with NLP techniques [6].

Verifact AI introduces a human-in-the-loop model: users can submit feedback after reviewing a prediction. This feedback loop supports ongoing learning and system adaptability, which are often missing in traditional models.

2.1.5. Real-Time and Feedback-Based AI Systems

AI-powered systems now emphasize real-time prediction and adaptability. Research by Ahmed et al. [7] proposed systems integrating live social media monitoring, NLP pipelines, and user validation. These systems continuously learn from both content and engagement patterns.

Verifact AI aligns with this direction by allowing administrators to view prediction logs from the MongoDB backend. Based on user feedback and verified external data, admins can correct predictions and mark them for retraining. These corrections are stored for later use, forming a retraining dataset.

This mechanism is consistent with active learning models, where the algorithm queries the most informative samples for retraining. Such systems improve over time with minimal manual labeling effort [8].

2.1.6. Model Retraining and Continuous Learning

Most fake news detection systems degrade over time due to changes in language use, new misinformation strategies, and topic shifts. Continuous learning, or incremental model retraining, is essential to maintain accuracy.

Pérez-Rosas et al. [9] emphasized that retraining with timely, verified samples enhances long-term reliability. In Verifact AI, the admin triggers the retraining process manually through a secure retrain interface. This approach allows controlled updates to the model using curated samples, balancing stability and adaptability.Verifact AI also ensures transparency and control, as all retraining data is manually verified before being included. This approach helps prevent the model from learning spurious patterns or biasing toward unreliable sources.

2.1.7. Summary and Relevance

The literature shows a clear evolution from basic ML models to sophisticated AI systems incorporating deep learning, knowledge graphs, and human feedback. While each method has strengths and limitations, hybrid and adaptive approaches offer the most promising results.

Verifact AI integrates:

Real-time news prediction using BERT,

Feedback-based learning for adaptability,

A retraining mechanism using validated corrections,

Admin oversight for quality control.

This system directly addresses the key limitations of existing fake news detection platforms — model aging, low context understanding, and lack of user engagement — and builds on the best practices from the current research landscape.

2.1.8. Challenges and Solutions in Verifact AI

Two major challenges were faced during the development of Verifact AI:

Choosing the Correct Model: Selecting the right model for fake news detection proved difficult. Traditional machine learning models struggled with context understanding and nuanced misinformation.

Solution: Verifact AI overcame this challenge by adopting BERT-based deep learning models, which are more adept at understanding context and capturing semantic patterns in news articles [3], [4].

Finding a Comprehensive Dataset: The lack of a comprehensive, up-to-date dataset posed a challenge for training the system effectively.

Solution: To address this, Verifact AI incorporates a feedback loop where users can submit feedback on predictions. This feedback, along with verified external data, is used for continuous retraining, ensuring that the system adapts to new misinformation patterns and remains accurate [7], [8].

2.2 EXISTING SYSTEM AND ITS DISADVANTAGES

Existing System:

Currently, the fight against fake news relies on a combination of manual and semi-automated systems. Traditional fact-checking is performed by independent fact-checking organizations and journalists who analyze news stories and publish verified reports. Additionally, some websites and social media platforms have integrated basic misinformation filters or flagging mechanisms where content can be reported by users or flagged through automated keyword-based algorithms. A few online tools also exist that use machine learning techniques for fake

news detection, but many of these tools are still in development stages or are not publicly accessible. Furthermore, some detection systems are limited to specific languages, regions, or platforms, reducing their global effectiveness.

Disadvantages of the Existing System:

Despite their efforts, existing systems for detecting fake news face several limitations:

Manual Effort and Delays: Traditional fact-checking methods are heavily manual and time-consuming. This leads to delays in identifying and debunking false information, during which the misinformation may already have gone viral.

Lack of Accessibility: Most existing tools are not designed with the average user in mind. They may require technical knowledge or access to premium platforms, making them inaccessible to the general public.

Limited Automation: Many systems rely on basic keyword detection or rule-based approaches that are not capable of understanding the context or semantics of a sentence, leading to inaccurate or inconsistent results.

Low Scalability: Manual systems and less-efficient models cannot scale to handle the large volume of data generated across the internet daily, especially on social media platforms.

Inconsistent Accuracy: Some automated tools produce false positives or negatives, as they lack advanced natural language processing capabilities needed to fully grasp the intent and meaning behind news content.

Language and Regional Barriers: A majority of fake news detection systems are developed for English-language content only, making them ineffective for multilingual or regional news verification.

Poor User Experience: Many systems fail to offer a simple, intuitive interface, making them difficult to navigate or use regularly by the general public.

Lack of Continuous Improvement: Some fake news detection tools do not evolve with emerging trends or adapt to new misinformation techniques, resulting in outdated detection methods that become less effective over time.

CHAPTER 3

SYSTEM DESIGN

3.1 USE CASE DIAGRAM

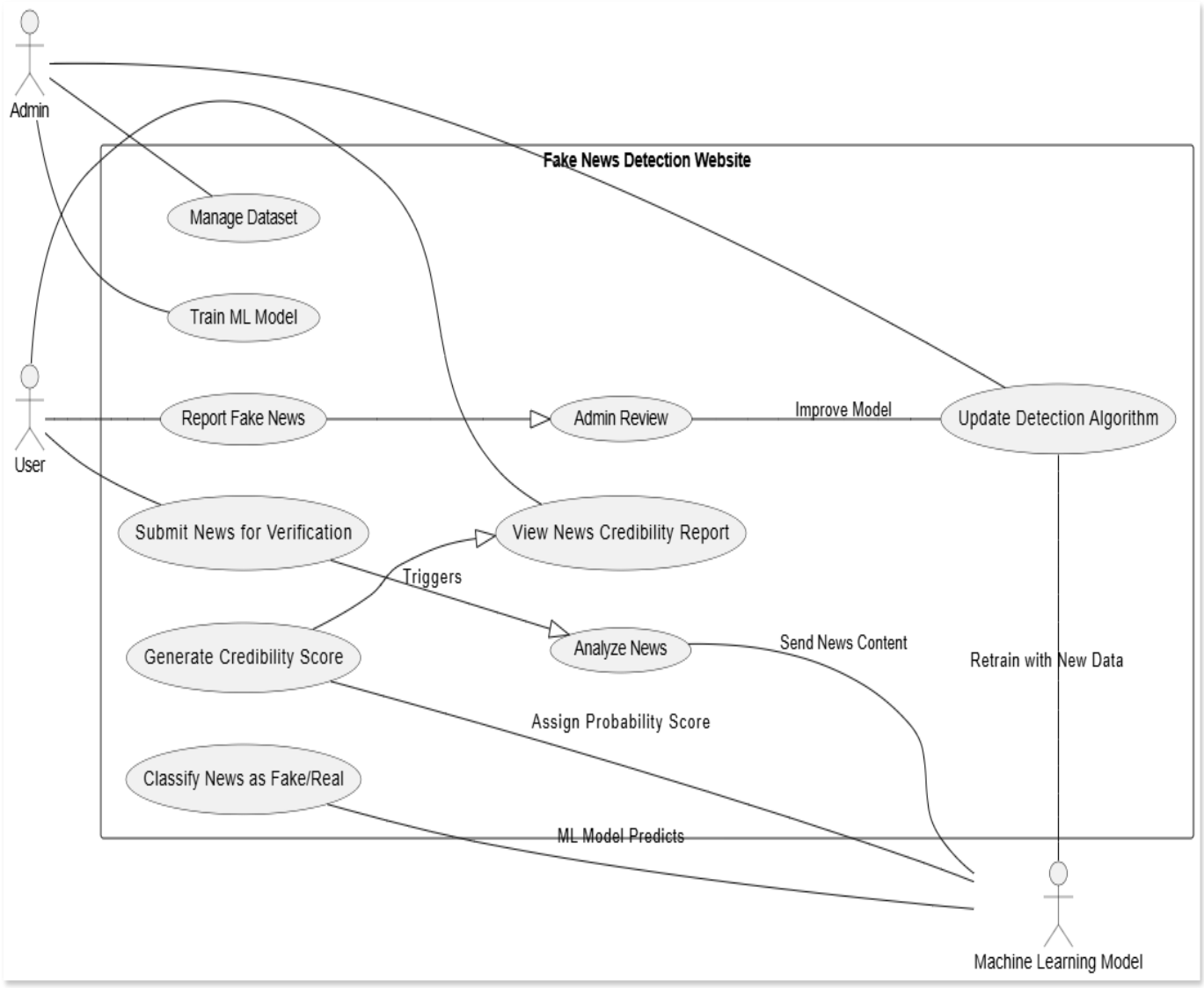

Fig. 3.1: Use Case Diagram

Admin:

These are external entities that interact with the system. In Verifact AI, the main actors include:

- User (General users submitting news for verification)
- Admin (Manages the system and database)
- AI Model (Processes news classification)

Use Cases:

These represent system functionalities that actors interact with, such as:

- Submit News Article – Users input text or URLs for verification.
- Process News – AI model analyzes the input.
- Display Results – The system presents real or fake news classification.
- User Authentication – Users register and log in.
- Manage News Records – Admin reviews stored data.

Relationships:

- Users interact with the system to submit news and view results.
- The AI model processes data and classifies the news.
- The admin monitors system performance and manages database records.

3.2 SEQUENCE DIAGRAM

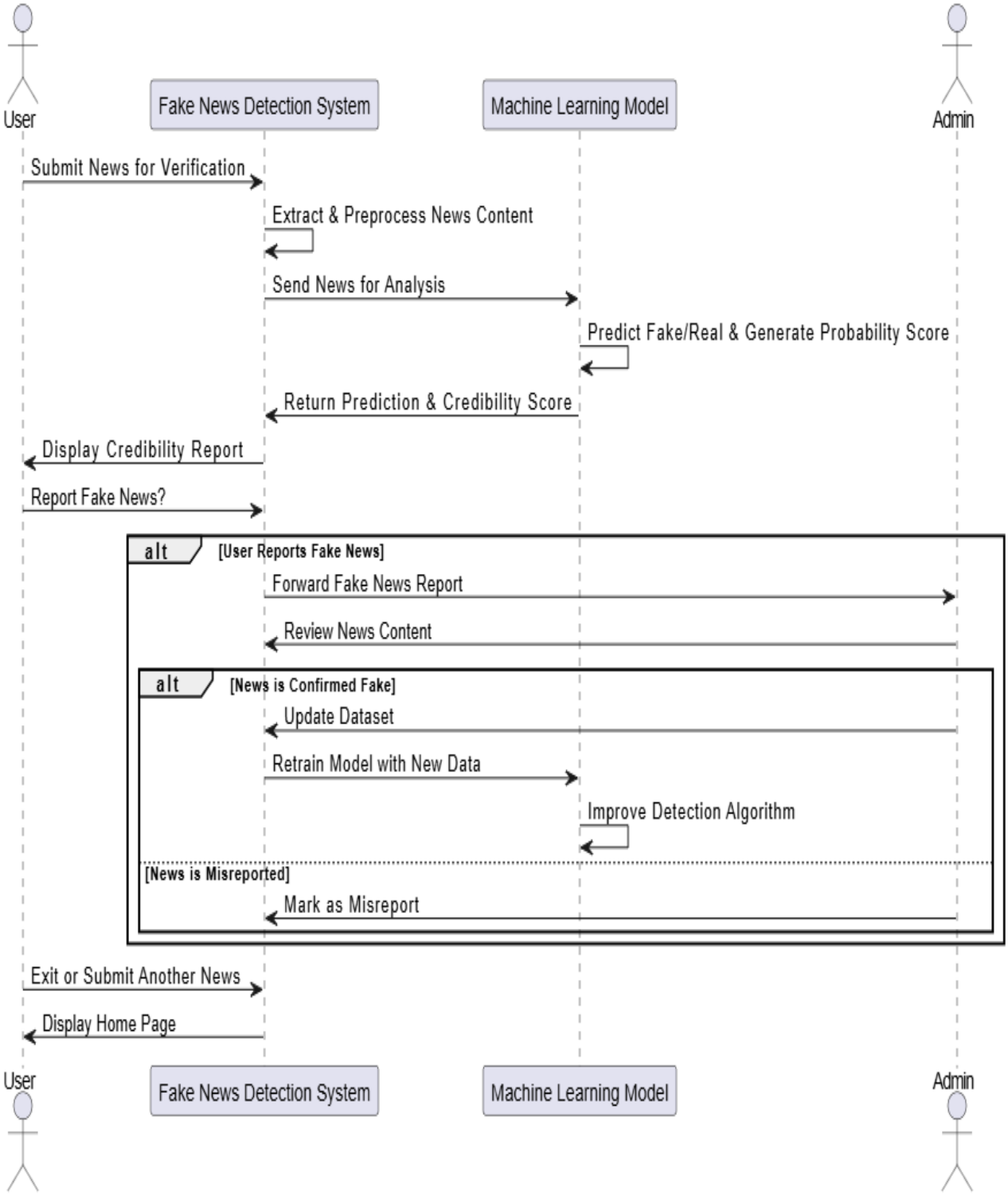

Fig. 3.2: Sequence Diagram

Admin:

- User: Submits news for verification.
- Web Application: The frontend interface where users interact.
- Backend Server: Processes requests and communicates with the AI model.
- AI Model: Analyzes the news and classifies it as real or fake.
- Database (MongoDB): Stores news analysis results and user details.

Process Flow:

- User submits news via the web application.
- Web application sends request to the backend server.
- Backend forwards data to the AI model for analysis.
- AI model processes and returns classification (real or fake).
- Backend stores results in the MongoDB database.

3.3 ACTIVITY DIAGRAM

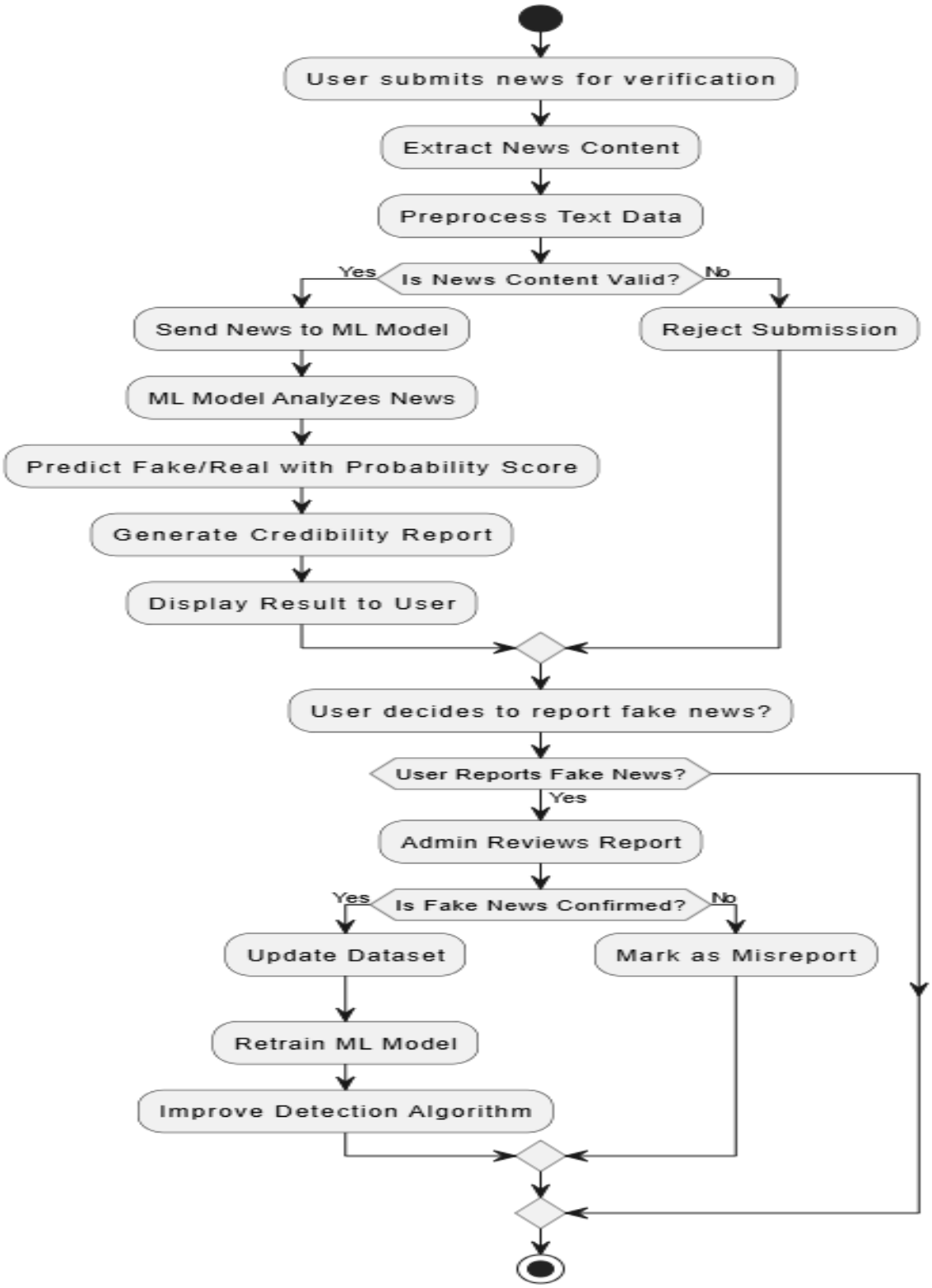

Fig. 3.3: Activity Diagram

Start Node:

- Represents the beginning of the process when a user accesses the system.

Activities:

- User Submits News: The user inputs a news article or URL for verification.
- System Preprocesses Data: The input text is cleaned and formatted.
- AI Model Analyzes News: The NLP-based AI model classifies the news as real or fake.
- Store Results in Database: The classification result is stored in MongoDB.
- Display Classification to User: The system provides the verification result to the user.

Decision Points:

- Valid Input Check: If the input is valid, the process continues; otherwise, the user is prompted to re-enter data.
- News Classification: If the news is detected as fake, the system may suggest verified sources.

Parallel Activities:

- The AI model can analyze multiple news inputs simultaneously.
- Admins can monitor the database and update flagged news.

End Node:

- Marks the completion of the verification process.

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 SYSTEM REQUIREMENTS

To successfully develop and deploy Verifact AI, both hardware and software requirements must be considered. On the hardware side, the system should ideally run on a machine with at least an Intel Core i5 processor, 8 GB of RAM (16 GB recommended for training machine learning models), and a minimum of 250 GB of storage. While integrated graphics are sufficient for deployment, a dedicated GPU can accelerate model training. A stable internet connection is essential for accessing web services and deploying the application online.

In terms of software, the application is designed to run on major operating systems such as Windows, Linux, or macOS. The frontend is developed using React.js, while the backend is powered primarily by Flask, which handles routing, machine learning model integration, and feedback endpoints. Express.js is used to manage specific API routes and acts as a bridge between the frontend and the Python-based backend. Development can be performed using code editors like Visual Studio Code or PyCharm.

The system utilizes JavaScript for frontend development and Python for backend logic and machine learning tasks. MongoDB serves as the primary database for storing user data and prediction history. Dependency management is handled through tools like npm, yarn, and pip, while Git is used for version control and collaboration.

For the machine learning component, libraries such as Hugging Face Transformers, TensorFlow, or PyTorch are employed to fine-tune and serve the BERT model [10]. Postman is used during development to test and validate API endpoints. End-users only need a modern web browser and an internet connection to access the platform, with basic interaction skills to input text and interpret results.

4.2 HARDWARE REQUIREMENTS FOR USER

VeriFact AI is designed to be accessible and lightweight for end-users, with minimal hardware requirements:

4.2.1. Device Compatibility

- Any standard desktop, laptop, tablet, or smartphone
- Device should support modern web browsing and basic multitasking

4.2.2. Minimum System Requirements

- RAM: Minimum 2 GB (Recommended: 4 GB or more for smoother experience)

- Processor: Dual-core CPU (e.g., Intel Pentium, Core i3, or ARM equivalent)

4.2.3. Browser Requirements

- Must have a modern web browser such as:
    - Google Chrome
    - Mozilla Firefox
    - Microsoft Edge

4.2.4. Internet Connectivity

- Stable internet connection (minimum 2 Mbps) for real-time API access and smooth UI interaction

4.3 SOFTWARE REQUIREMENTS

The development and deployment of VeriFact AI involve the following software tools, platforms, and libraries:

4.3.1. Operating System

- Windows 10 or 11
- macOS
- Any modern Linux distribution (e.g., Ubuntu, Fedora)

4.3.2. Frontend Technologies

- React.js – For building a dynamic and responsive user interface
- JavaScript, HTML, CSS – For designing and scripting frontend behavior
- React Router – For page navigation

4.3.3. Backend Technologies

- Flask – Used as the primary backend framework for handling machine learning model predictions, routing, and feedback endpoints [11].
- Express.js – Handles API routing and facilitates communication between the React frontend and the Python backend (Flask) [12].
- Node.js – Supports server-side logic and manages middleware communication to ensure seamless integration across components [13].

4.3.4. Code Editors / IDEs

- Visual Studio Code

4.3.5. Database

- MongoDB – For storing user data, submitted news, and prediction history [14].

4.4 HARDWARE REQUIREMENTS FOR DEVELOPMENT

The following are the essential hardware specifications for developing and running VeriFact AI efficiently:

4.4.1. Processor

- Minimum: Intel Core i5 or AMD Ryzen 5
- Recommended: Intel Core i7 or higher for faster multitasking and processing

4.4.2. RAM

- Minimum: 8 GB
- Recommended: 16 GB for smooth ML model training and multitasking

4.4.3. Storage

- Minimum: 250 GB HDD or SSD
- Recommended: SSD for faster read/write during training and deployment

4.4.4. GPU

- Integrated GPU is sufficient for basic tasks
- Dedicated NVIDIA GPU (with CUDA) recommended for ML acceleration

4.4.5. Internet

- Stable broadband connection (10 Mbps or more)
- Required for dependency installation, API testing, and deployment

CHAPTER 5

IMPLEMENTATION

5.1 DATASET USED

For the training and evaluation of VeriFact AI, a publicly available dataset was utilized that contains both fake and real news articles. The dataset was sourced from Kaggle, a popular platform for data science competitions and datasets.

5.1.1. Dataset Name

Fake and Real News Dataset

(Source: Kaggle - Fake and Real News Dataset) [15]

5.1.2. Description

This dataset includes labeled news articles categorized as either "Fake" or "Real". It is widely used for machine learning tasks involving text classification, especially in the domain of fake news detection.

5.1.3. Composition

- Fake.csv: Contains approximately 23,000+ fake news articles
- True.csv: Contains approximately 21,000+ real news articles
- Each entry includes:
    - Title
    - Text (main content)
    - Subject
    - Date

5.2 MODELS

The core of the VeriFact AI system relies on the BERT (Bidirectional Encoder Representations from Transformers) model [16] for fake news detection. BERT is a state-of-the-art Natural Language Processing (NLP) model developed by Google, known for its deep understanding of language context and semantics. Unlike traditional models that process text in a single direction, BERT reads input in both directions (left-to-right and right-to-left), allowing it to capture the full meaning of a sentence. In this project, BERT is fine-tuned on a labeled dataset consisting of real and fake news articles. The model learns to classify input text as either "real" or "fake" based on linguistic patterns, word relationships, and contextual cues.

BERT's pre-trained architecture significantly reduces training time and improves performance, as it already understands general language structure. This makes it ideal for tasks like fake news detection, where subtle differences in tone, word usage, or phrasing can indicate misinformation. Once trained, the model is integrated into the backend using Flask or Django REST APIs, where it processes user-submitted text and returns predictions in real-time. The use of BERT greatly enhances the accuracy, reliability, and intelligence of the VeriFact AI system.

5.3 TOOLS/TECHNOLOGIES USED

The development of VeriFact AI incorporates a combination of modern tools and technologies across the frontend, backend, machine learning, and database components. For the frontend, a responsive and interactive user interface is built using React.js, a popular JavaScript library known for its component-based architecture and efficient rendering. The backend is developed using Flask, a lightweight and flexible Python web framework, which handles both application routing and the integration of the trained machine learning model. Additionally, Express.js, a minimalist Node.js framework, is used to manage specific APIs and facilitate smooth communication between the client and server.

For the machine learning component, the project utilizes the BERT model, a state-of-the-art NLP model from Google's Transformers family, implemented using libraries such as Hugging Face Transformers, TensorFlow, or PyTorch. The model is fine-tuned to classify news as real or fake based on the training dataset. Database management is handled by MongoDB, a NoSQL database that stores user input, results, and interaction history in a flexible document format.

Development and testing are supported by tools like Postman for API testing, Visual Studio Code for coding, and Git/GitHub for version control and collaboration. These tools collectively ensure that the system remains modular, scalable, and user-friendly, while also supporting real-time prediction and efficient data handling.

5.4 BACKEND IMPLEMENTATION

The backend of the VeriFact AI system is built using a combination of Node.js and Python to handle data flow, machine learning tasks, and secure communication between the frontend and the model.

5.4.1. Node.js and Express.js Backend

The Node.js server, implemented using Express.js, acts as a middleware between the React frontend and the Python-based machine learning backend. It handles:

- User requests for news submission and verification
- Routing and secure API communication
- Interfacing with the MongoDB database to store user-posted news and trending data

- Forwarding requests to the Python server using Axios

Axios is used for making HTTP requests to the Flask server where the ML model is hosted. It ensures asynchronous communication and proper data transmission in JSON format, as shown in Fig 8.3.

5.4.2. Python and Flask ML Server

The Python backend, built using Flask, is responsible for machine learning operations. It includes:

- A /predict endpoint to classify news as *Fake* or *Real* using a trained model, as shown in Fig 8.4.
- A /feedback endpoint to collect labeled data from the admin for future retraining as shown in Fig 8.5.
- A /retrain endpoint which triggers model retraining using accumulated feedback, as shown in Fig 8.11.

The ML model leverages BERT (Bidirectional Encoder Representations from Transformers) for natural language understanding and fine-tuned classification. To enhance preprocessing, various NLP libraries are used for:

- Text cleaning and normalization
- Tokenization and lemmatization
- Vectorization using BERT embeddings

The model is trained using labeled datasets (fake.csv and true.csv) and saves both the model and vectorizer using joblib for efficient reuse during prediction.

5.5 FRONTEND IMPLEMENTATION

The frontend of VeriFact AI is developed using React.js, a modern JavaScript library for building responsive and component-based user interfaces. The frontend interacts with both the Node.js backend and the Python ML server to offer a seamless user experience.

5.5.1. Key Pages and Components

- Home Page: Acts as the landing page of the application, giving an overview of the platform and providing navigation links as shown in Fig 8.1.
- Check News: A user-friendly interface where users can input any news headline or paragraph and click “Analyze”. This sends a request to the backend via Axios and returns a prediction along with a confidence score and visual feedback (image, color-coded result, etc.) as shown in Fig 8.2.

- Trending News: Displays top or recent news entries fetched from MongoDB. It gives users a sense of what news is currently being posted and verified on the platform as shown in Fig 8.6.
- Submit News: Allows admin to post correct news articles along with correct prediction. These are stored in the backend folder and performs in retraining as shown in Fig 8.7.
- Admin Panel (Admin-AI): A password-protected section where the admin can view feedback-submitted news, change predictions using a dropdown (Fake/Real), and trigger retraining of the ML model. Retraining is restricted to once per day or week to avoid unnecessary load as shown in Fig 8.10.

5.5.2. Technologies and Libraries Used

- React.js for building the UI components
- Axios for making HTTP requests to the backend (Node and Flask)
- React Router for navigation between pages
- CSS and Styled Components for styling and layout
- State Management using React Hooks (useState, useEffect) for data handling and updates

5.5.3. Frontend-Backend Integration

The React components communicate with the Node.js and Flask servers using Axios. For example:

- News data is sent to the /predict endpoint for analysis as shown in Fig 8.3
- Labeled feedback is posted to the /feedback endpoint as shown in Fig 8.5.
- Admin-triggered retraining calls the /retrain endpoint securely as shown in Fig 8.11.

5.6 WORKING OF THE SYSTEM

VeriFact AI follows a structured flow for verifying news authenticity and improving prediction accuracy over time through user feedback and model retraining:

5.6.1. News Verification

- The user visits the platform and inputs a news headline or article into the Check News section.
- The frontend (React.js) sends the data to the Flask backend via a REST API call.
- The pre-trained BERT model processes the text and predicts whether the news is "Real" or "Fake".

5.6.2. Feedback Collection

- If the user disagrees with the prediction, they can submit feedback indicating the correct label as shown in Fig 8.5.
- This feedback is stored in MongoDB for review and further processing.

5.6.3. Admin Review and Retraining

- The Admin accesses the Admin AI page through a secure login.
- The admin can view feedback data and export it to a file (feedback.csv) as shown in Fig 8.6 & as shown in Fig 8.7.
- Upon review, the admin can trigger model retraining, which updates the model using both the original dataset and new feedback as shown in Fig 8.11.

CHAPTER 6

CODE

6.1 Training_model.py

```
from transformers import TrainingArguments, Trainer
# Define training arguments
training_args = TrainingArguments(
    output_dir="./bert_model",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,  # Increase for better accuracy
    logging_dir="./logs",
    logging_steps=500,
    save_total_limit=2,
)
# Define trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
# Train model
trainer.train()
```

dataset_tokenization.py

```
from transformers import BertTokenizer
# Load BERT tokenizer
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
```

```
# Tokenize function
def tokenize_function(texts):
    return tokenizer(texts, padding="max_length", truncation=True, max_length=512)
# Apply tokenization
train_encodings = tokenize_function(train_texts)
val_encodings = tokenize_function(val_texts)
```

classification_head.py

```
from transformers import BertForSequenceClassification
# Load BERT with classification head
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
```

6.2 Train_bert.py

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from torch.utils.data import Dataset
import pandas as pd
from sklearn.model_selection import train_test_split
# ✅ Load dataset
df = pd.read_csv("news_dataset.csv")
# ✅ Convert labels to integers if needed
if df["label"].dtype == object:
    df["label"] = df["label"].map({"FAKE": 0, "REAL": 1})  # Convert to 0/1
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
# ✅ Convert data to BERT input format
class FakeNewsDataset(Dataset):
    def _init_(self, texts, labels):
        self.texts = texts
        self.labels = labels

    def _len_(self):
        return len(self.texts)
```

```
    def _getitem_(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
encoding = tokenizer(text, truncation=True, padding="max_length", max_length=256,
return_tensors="pt")
          return {
           "input_ids": encoding["input_ids"].squeeze(0),
           "attention_mask": encoding["attention_mask"].squeeze(0),
           "labels": torch.tensor(label, dtype=torch.long),
        }
# ✅ Split dataset into training and validation sets
train_texts, val_texts, train_labels, val_labels = train_test_split(df["text"], df["label"], test_size=0.2,
random_state=42)
# ✅ Create Dataset objects
train_dataset = FakeNewsDataset(train_texts.tolist(), train_labels.tolist())
val_dataset = FakeNewsDataset(val_texts.tolist(), val_labels.tolist())
# ✅ Define model with correct classification head
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
# ✅ Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=500,
    save_total_limit=2,
    fp16=True,  # ✅ Enables mixed-precision for speedup on GPU
)
trainer = Trainer(
```

```
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
# ✅ Train model
trainer.train()
# ✅ Save final trained model
model.save_pretrained("bert_fake_news")
tokenizer.save_pretrained("bert_fake_news")
print("✅ Model training complete! Saved in 'bert_fake_news' folder.")
```

6.3 Training.py

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from torch.utils.data import Dataset
import pandas as pd
from sklearn.model_selection import train_test_split
# ✅ Load dataset
df = pd.read_csv("news_dataset.csv")
# ✅ Convert labels to integers if needed
if df["label"].dtype == object:
    df["label"] = df["label"].map({"FAKE": 0, "REAL": 1})  # Convert to 0/1
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
# ✅ Convert data to BERT input format
class FakeNewsDataset(Dataset):
    def _init_(self, texts, labels):
        self.texts = texts
        self.labels = labels

    def _len_(self):
        return len(self.texts)
```

```
 def _getitem_(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
 encoding = tokenizer(text, truncation=True, padding="max_length", max_length=256,
return_tensors="pt")
       return {
           "input_ids": encoding["input_ids"].squeeze(0),
           "attention_mask": encoding["attention_mask"].squeeze(0),
           "labels": torch.tensor(label, dtype=torch.long),
        }
# ✅ Split dataset into training and validation sets
train_texts, val_texts, train_labels, val_labels = train_test_split(df["text"], df["label"], test_size=0.2,
random_state=42)
# ✅ Create Dataset objects
train_dataset = FakeNewsDataset(train_texts.tolist(), train_labels.tolist())
val_dataset = FakeNewsDataset(val_texts.tolist(), val_labels.tolist())
# ✅ Define model with correct classification head
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
# ✅ Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=500,
    save_total_limit=2,
    fp16=True,  # ✅ Enables mixed-precision for speedup on GPU
)
trainer = Trainer(
```

```
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
)
# ✅ Train model
trainer.train()
# ✅ Save final trained model
model.save_pretrained("bert_fake_news")
tokenizer.save_pretrained("bert_fake_news")
print("✅ Model training complete! Saved in 'bert_fake_news' folder.")
```

dataset_prepration.py

```
import pandas as pd
# ✅ Load datasets
fake_df = pd.read_csv("fake.csv")
true_df = pd.read_csv("true.csv")
# ✅ Add labels (1 = real, 0 = fake)
fake_df["label"] = 0
true_df["label"] = 1
# ✅ Select relevant columns (assuming 'text' column contains the news article)
fake_df = fake_df[["text", "label"]]
true_df = true_df[["text", "label"]]
# ✅ Merge datasets
df = pd.concat([fake_df, true_df], ignore_index=True)
# ✅ Shuffle dataset
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
# ✅ Save the dataset
df.to_csv("news_dataset.csv", index=False)

print("✅ news_dataset.csv created successfully!")
```

6.4 Model.py

```
import joblib
import re
import string
import nltk
import spacy
from nltk.tokenize import word_tokenize
from transformers import BertTokenizer, BertForSequenceClassification
import torch
# Load NLP resources
nltk.download("punkt")
# Load spaCy's NER model
nlp = spacy.load("en_core_web_sm")
# Load saved models
vectorizer = joblib.load("vectorizer.pkl")
classifier = joblib.load("model.pkl")
bert_model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=2)
bert_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
# ✅ Function to clean text
def clean_text(text):
    text = text.lower()
    text = re.sub(r"\d+", "", text)
    text = text.translate(str.maketrans("", "", string.punctuation))
    tokens = word_tokenize(text)
    return " ".join(tokens)
# ✅ Function to check geographical facts
def check_fact(text):
    doc = nlp(text)
    locations = [ent.text for ent in doc.ents if ent.label_ == "GPE"]  # GPE = Geo-Political Entity
# Hardcoded real-world facts (extend this list)
```

```
    geo_facts = {
        "delhi": "india",
        "paris": "france",
        "new york": "usa",
        "beijing": "china",
    }
 for loc in locations:
        if loc.lower() in geo_facts and geo_facts[loc.lower()] not in text.lower():
            return False  # Fact is incorrect
return True  # Fact is correct
# ✅ Function to predict using the model
def predict_news(news_text):
    cleaned_text = clean_text(news_text)
# Traditional model prediction
    features = vectorizer.transform([cleaned_text])
    traditional_pred = classifier.predict(features)[0]  # 1 for real, 0 for fake
    traditional_label = "real" if traditional_pred == 1 else "fake"
 # BERT prediction
    inputs = bert_tokenizer(cleaned_text, return_tensors="pt", padding=True, truncation=True,
max_length=512)
    outputs = bert_model(**inputs)
    bert_pred = torch.argmax(outputs.logits).item()
    bert_label = "real" if bert_pred == 1 else "fake"
 # Fact-checking using spaCy
    fact_ok = check_fact(news_text)
# Combine results (this is an example ensemble logic)
    if not fact_ok:
        return "fake"
    if traditional_label == bert_label:
        return traditional_label
    return "uncertain"
```

6.5 App.py

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import joblib
import csv
import os
import logging
import subprocess
app = Flask(__name__)
CORS(app)
logging.basicConfig(level=logging.INFO)
model = joblib.load("model.pkl")
vectorizer = joblib.load("tfidf.pkl")
@app.route("/")
def home():
    return " ✅ Fake News Detection Backend is Running"
@app.route("/predict", methods=["POST"])
def predict():
    try:
        data = request.get_json()
        news = data.get("news", "")
        if not news:
            return jsonify({"error": "No news content provided"}), 400
 vectorized_news = vectorizer.transform([news])
        prediction = model.predict(vectorized_news)[0]
        confidence = model.predict_proba(vectorized_news).max()

        logging.info(f" 📊 Model prediction made using model id: {id(model)}")
```

```
        return jsonify({
            "prediction": "Fake" if prediction == 0 else "Real",
            "confidence": round(confidence * 100, 2)
        })

    except Exception as e:
        logging.error(f"❌ Prediction error: {e}")
        return jsonify({"error": str(e)}), 500
@app.route("/feedback", methods=["POST"])
def feedback():
    try:
        data = request.json.get("data", [])
        if not data:
            return jsonify({"error": "No data provided"}), 400
 file_exists = os.path.isfile("feedback.csv")
        with open("feedback.csv", "a", newline='', encoding='utf-8') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=["news", "label"])
            if not file_exists:
                writer.writeheader()
            for item in data:
                news = item.get("news")
                label = item.get("label")
                if news and label in ["Fake", "Real"]:
                    writer.writerow({"news": news, "label": label})

        return jsonify({"message": "Feedback stored", "count": len(data)}), 200

    except Exception as e:
        logging.error(f"❌ Feedback error: {e}")
        return jsonify({"error": "Internal server error"}), 500
@app.route("/retrain", methods=["POST"])
def retrain():
    logging.info("📨 /retrain endpoint hit")
```

```
try:
    data = request.json.get("data", [])
    logging.info(f"Received JSON: {request.json}")

    if not os.path.exists("feedback.csv"):
        logging.warning("⚠️ feedback.csv not found.")
        return jsonify({"error": "feedback.csv not found"}), 500

    logging.info("✅ Starting model retraining...")
    result = subprocess.run(
        ["python", "train_model.py"],
        check=True,
        capture_output=True,
        text=True
    )

    logging.info("✅ Retraining completed")
    logging.info(f"📤 Output from train_model.py:\n{result.stdout}")

    # ✅ Reload model and vectorizer after retraining
    try:
        global model, vectorizer
        model = joblib.load("model.pkl")
        vectorizer = joblib.load("tfidf.pkl")
        logging.info(f"🔁 Model and vectorizer reloaded successfully. Model ID: {id(model)}")
    except Exception as reload_error:
        logging.error(f"❌ Failed to reload model/vectorizer: {reload_error}")

    return jsonify({
        "message": "✅ Model retrained successfully!",
        "details": result.stdout + "\n" + result.stderr
    }), 200
```

```
    except subprocess.CalledProcessError as e:
        logging.error(f"❌ Retrain script failed: {e}")
        return jsonify({"error": "Retrain script failed"}), 500
    except Exception as e:
        logging.error(f"❌ Error: {str(e)}")
        return jsonify({"error": "Internal server error"}), 500

if __name__ == "__main__":
    app.run(debug=True)
```

6.6 index.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
require("dotenv").config();

const newsRoutes = require("./routes/newsRoutes");

const app = express();
app.use(express.json());
app.use(cors());

const MONGO_URI = "mongodb://localhost:27017/fakenews";

mongoose
  .connect(MONGO_URI)
  .then(() => console.log("✅ MongoDB connected to fakenews"))
  .catch((err) => console.error("❌ MongoDB connection error:", err));

app.use("/api/news", newsRoutes);
```

```
const PORT = 5001;
app.listen(PORT, () => console.log(`🚀 Server running on port ${PORT}`));
```

6.7 News.js

```
const mongoose = require("mongoose");
const newsSchema = new mongoose.Schema(
  {
    news: { type: String, required: true },
    prediction: { type: String, required: true },
    feedback: { type: String, default: null },
  },
  { timestamps: true }
);
console.log("✅ News Schema Loaded");
module.exports = mongoose.model("News", newsSchema);
```

6.8 NewsRoutes.js

```
const express = require("express");
const router = express.Router();
const News = require("../models/News");

// Store analyzed news
router.post("/check", async (req, res) => {
    try {
        console.log("📩 Received request to store news...");

        const { news, prediction } = req.body;

        if (!news || typeof news !== "string" || news.trim() === "") {
```

```
            return res.status(400).json({ error: "News content is required" });
        }

        if (!["Real", "Fake"].includes(prediction)) {
            return res.status(400).json({ error: "Prediction must be 'Real' or 'Fake'" });
        }

        const newNews = new News({ news, prediction });
        const savedNews = await newNews.save();

        console.log("✅ News successfully stored:", savedNews);
        res.status(201).json(savedNews);
    } catch (error) {
        console.error("❌ Error saving news:", error);
        res.status(500).json({ error: "Internal Server Error" });
    }
});

// ✅ Fixed: Return { news: [...] }
router.get("/all", async (req, res) => {
    try {
        console.log("📡 Fetching all news from MongoDB...");
        const newsList = await News.find();
        console.log(`✅ Found ${newsList.length} news items.`);
        res.json({ news: newsList });
    } catch (error) {
        console.error("❌ Error fetching news:", error);
        res.status(500).json({ error: "Internal Server Error" });
    }
});

// Store feedback
router.put("/feedback/:id", async (req, res) => {
```

```
    try {
        const { feedback } = req.body;

        if (!feedback || typeof feedback !== "string" || feedback.trim() === "") {
            return res.status(400).json({ error: "Feedback must be a valid string" });
        }

        const updatedNews = await News.findByIdAndUpdate(
            req.params.id,
            { feedback },
            { new: true }
        );

        if (!updatedNews) {
            return res.status(404).json({ error: "News not found" });
        }

        console.log("✅ Feedback updated:", updatedNews);
        res.json(updatedNews);
    } catch (error) {
        console.error("❌ Error updating feedback:", error);
        res.status(500).json({ error: "Internal Server Error" });
    }
});

module.exports = router;
```

CHAPTER 7

APPLICATIONS

VeriFact AI has wide-ranging applications across multiple domains where information accuracy and digital trust are crucial. Below are the key areas where the system proves beneficial:

7.1. Media and Journalism

- Assists editors, journalists, and content creators in verifying the credibility of news before publication.
- Ensures the accuracy of information being disseminated to the public.
- Reduces the risk of publishing false or misleading content, enhancing trust in media houses.

7.2. Social Media and Content Platforms

- Can be integrated into platforms like Facebook, Twitter, and WhatsApp as a real-time backend verification tool.
- Minimizes the spread of misinformation, especially during elections, pandemics, or social unrest.

7.3. Education and Academic Research

- Aids students, teachers, and researchers in verifying the authenticity of online sources.
- Promotes digital literacy and encourages responsible use of online information.
- Supports academic integrity by reducing reliance on unverified or biased sources.

7.4. Fact-Checking Organizations and NGOs

- Automates the initial fact-verification process, saving time and manual effort.
- Assists NGOs working to promote media transparency and awareness.

7.5. Individual Users

- Empowers users to verify the truthfulness of news they consume or share.
- Encourages critical thinking and informed decision-making.
- Fosters a society that is aware, responsible, and resistant to misinformation.

CHAPTER 8

RESULT

8.1 Home Page

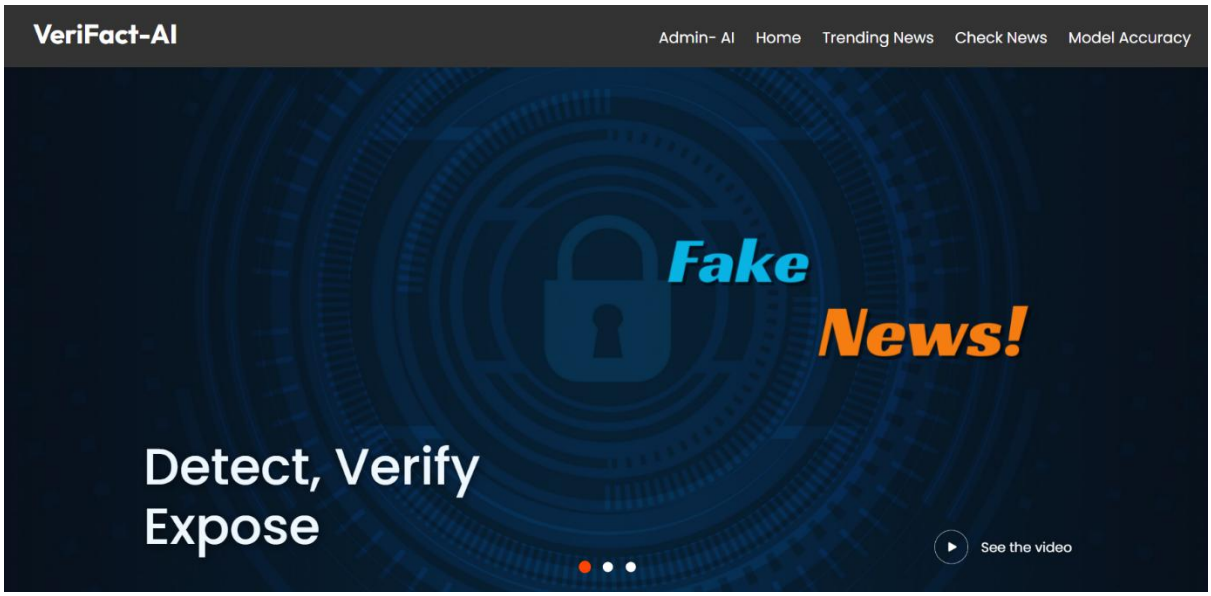

Fig. 8.1: Home Page of Verifact AI

8.2 Check News Page

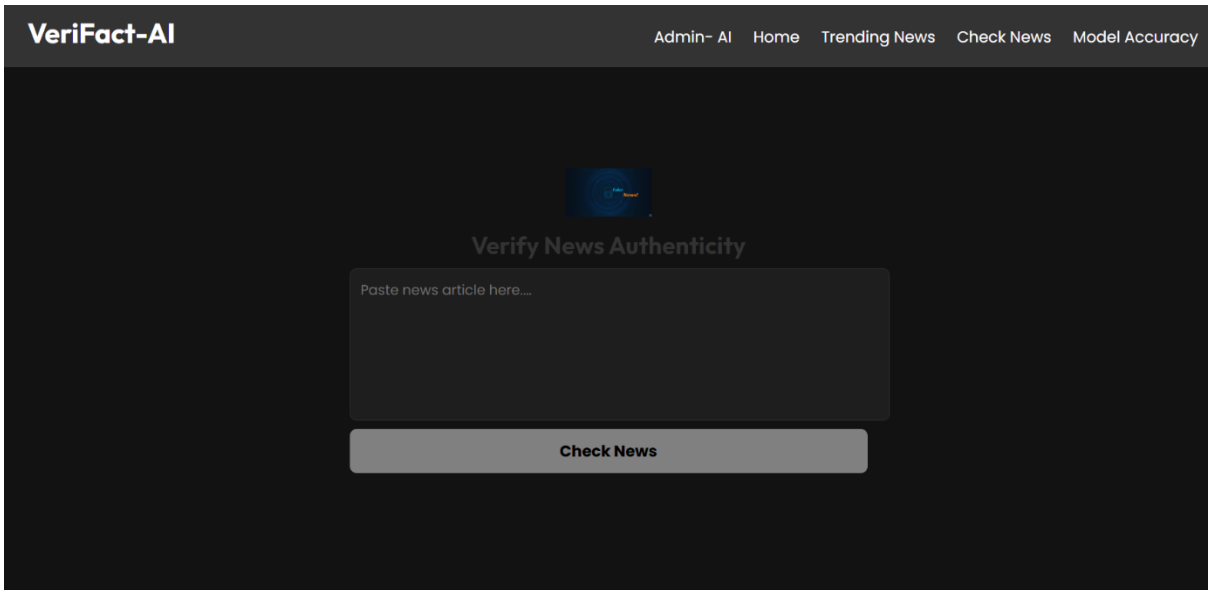

Fig. 8.2: User interface for submitting news article

8.3 Enter News Interface

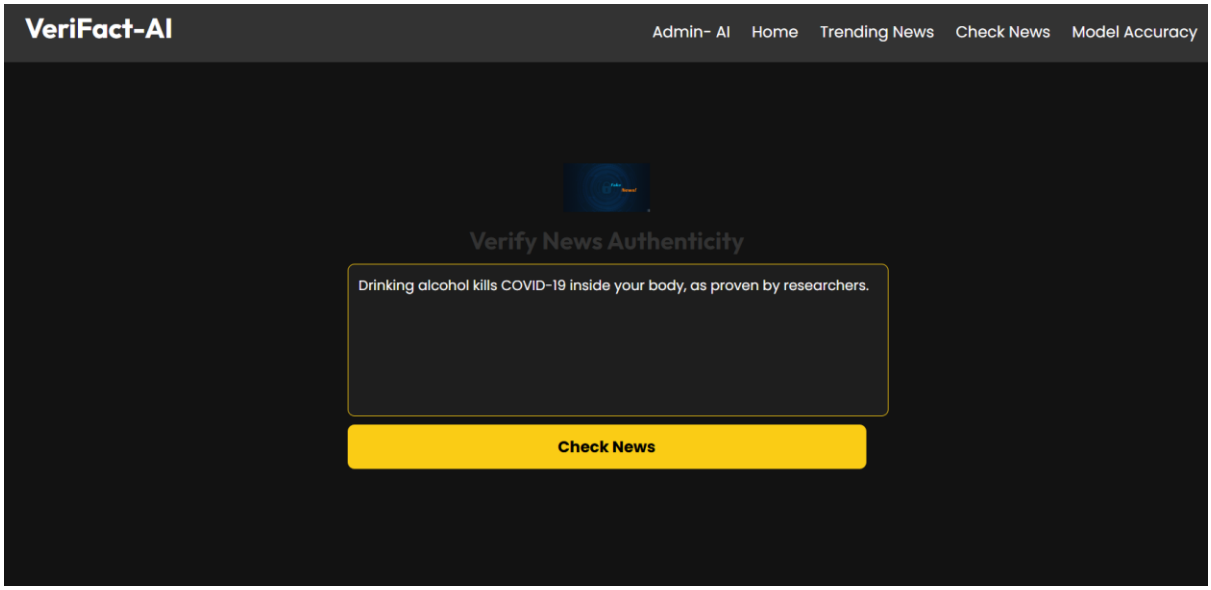

Fig. 8.3: News entered by user

8.4 Prediction Interface

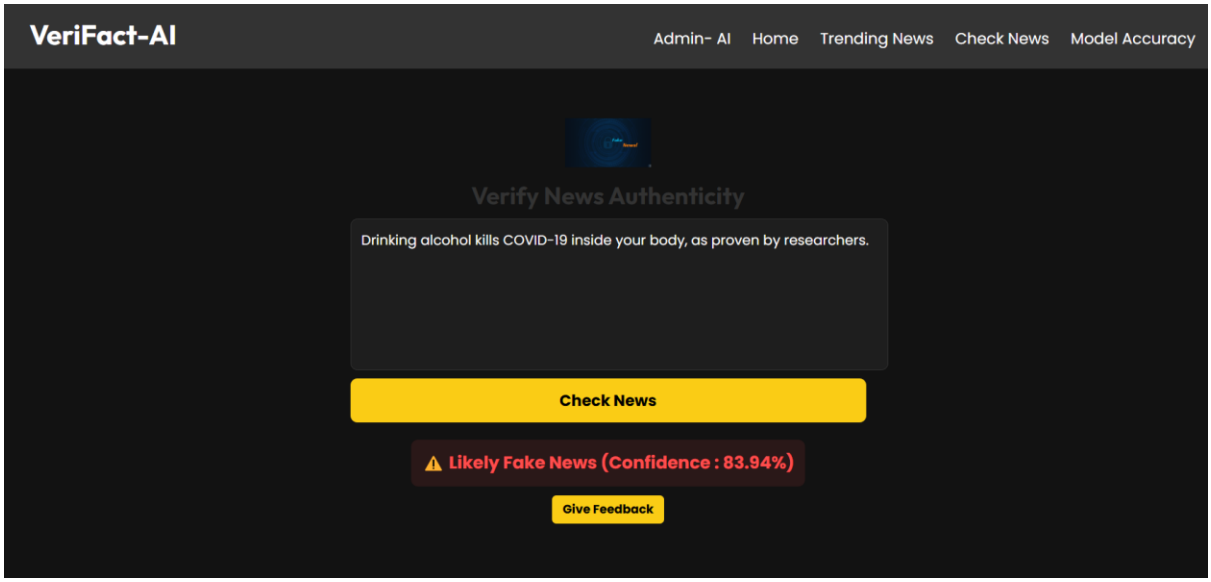

Fig. 8.4: Fake news detection result with confidence score

8.5 Feedback input Interface

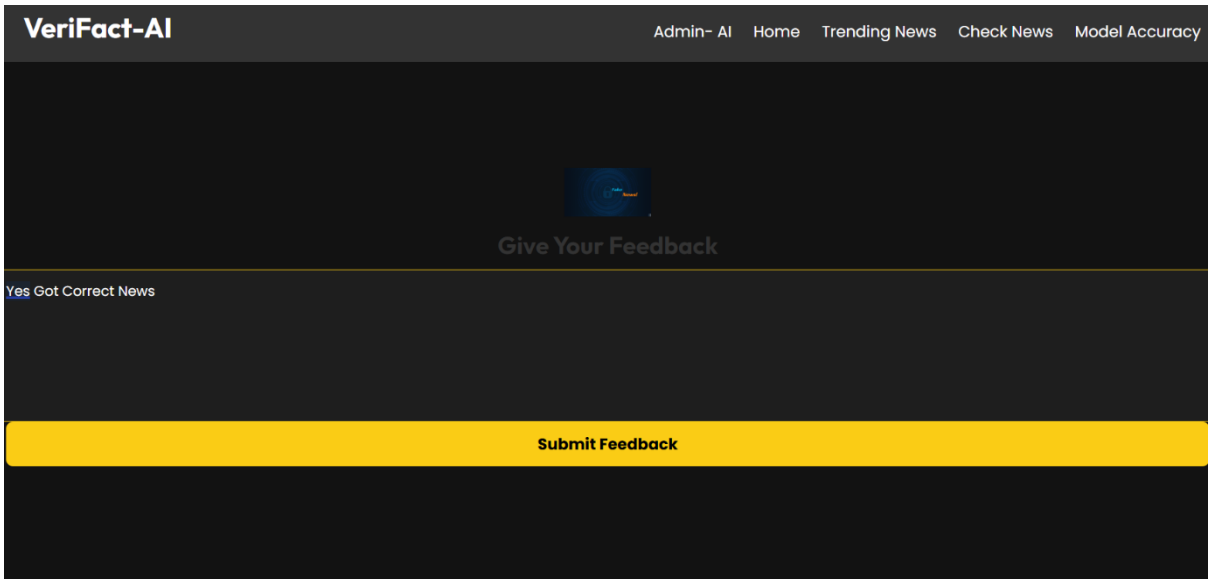

Fig. 8.5: User feedback interface

8.6 Trending News Page Interface

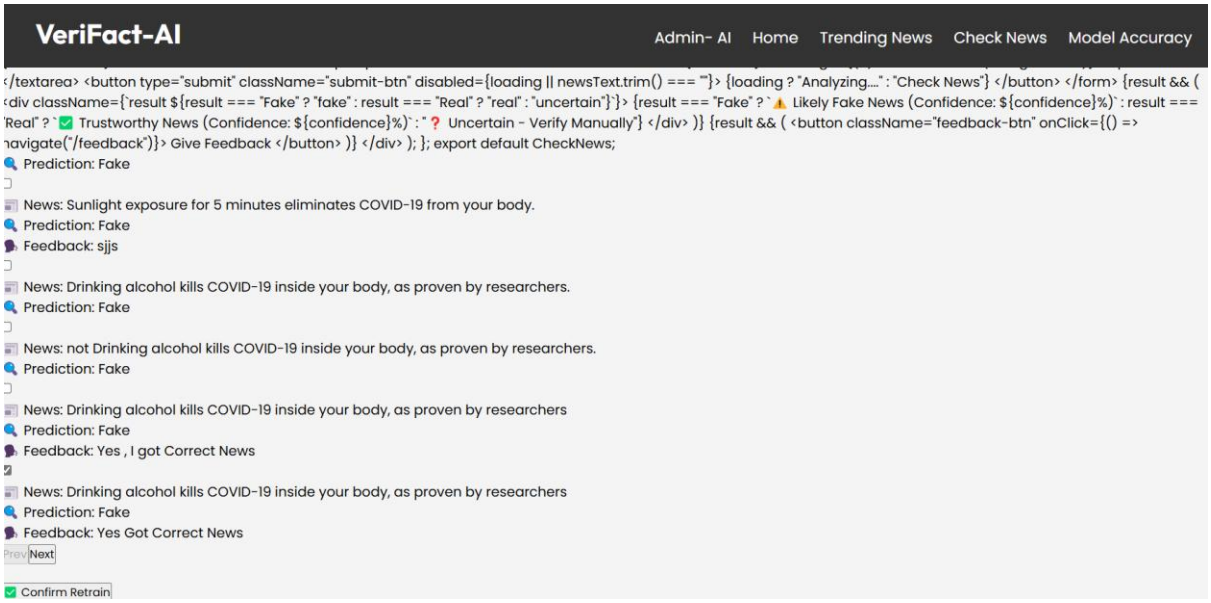

Fig. 8.6: Displays trending news

8.7 News Sent To Retrain

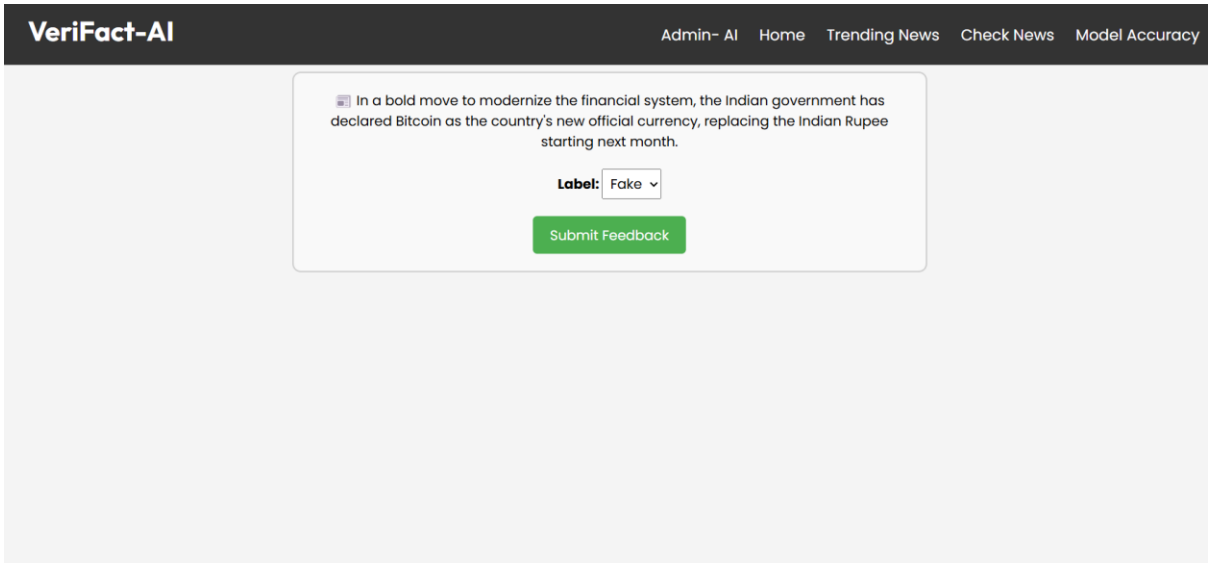

Fig. 8.7: Retraining interface

8.8 News Sent success

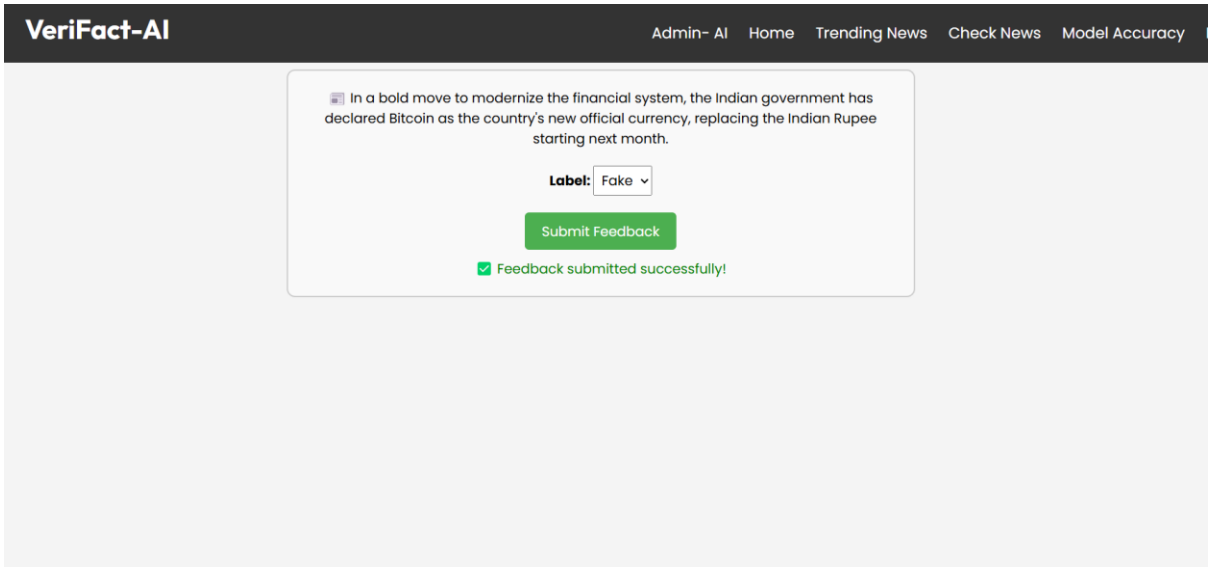

Fig. 8.8: Feedback sent successfully

8.9 News Stored to file (feedback.csv)

Fig. 8.9: Feedback folder storing

8.10 Admin page

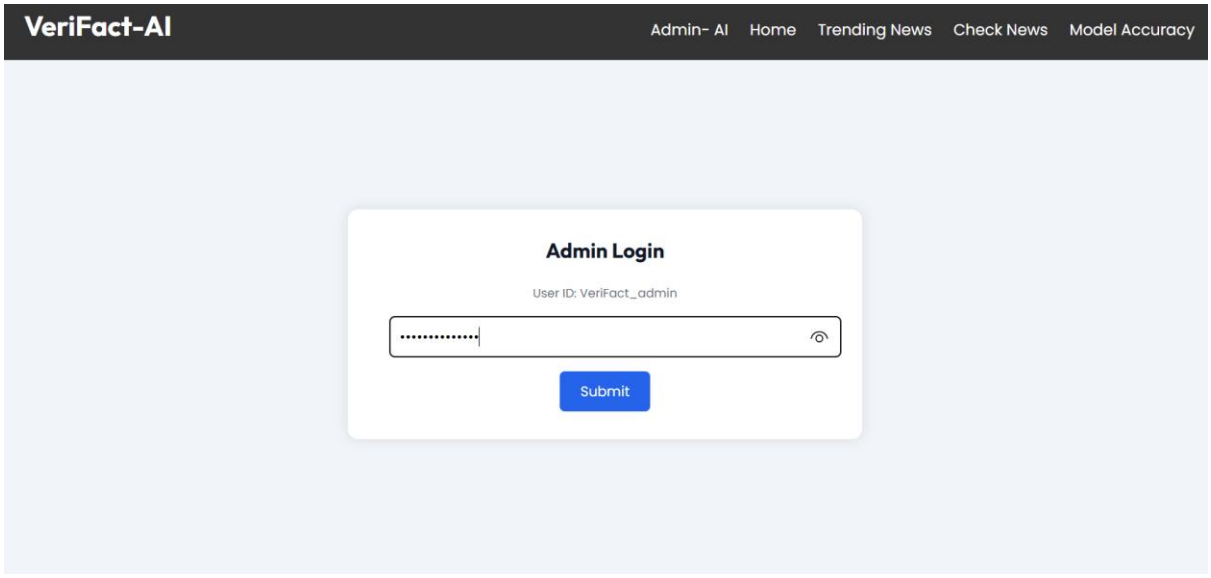

Fig. 8.10: Admin authentication

8.11 Retraining The Model

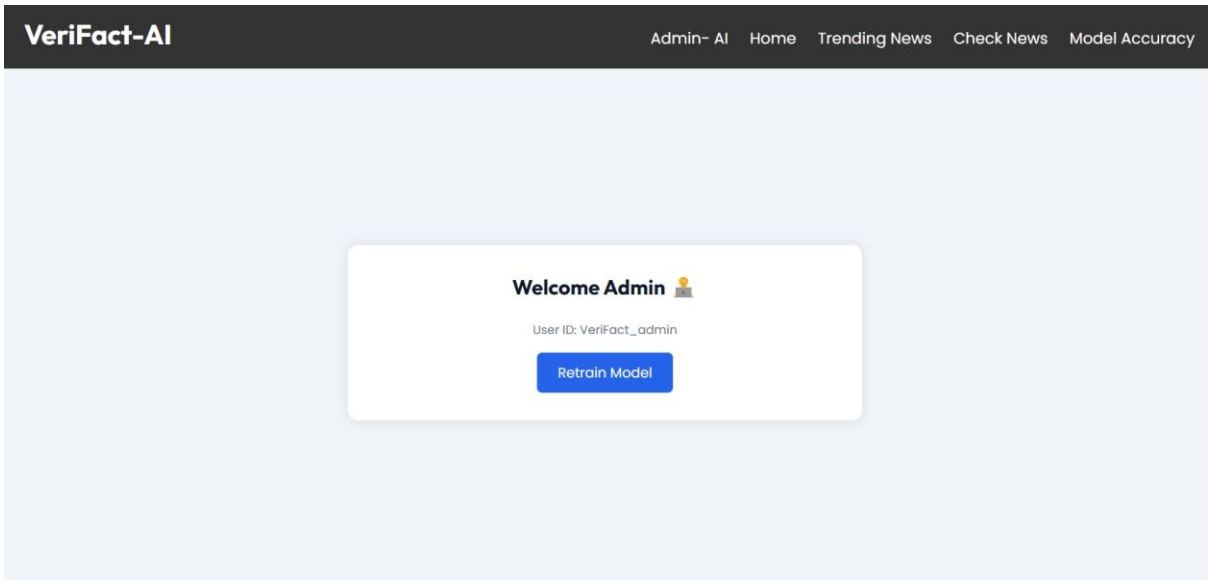

Fig. 8.11: Admin authentication successful

8.12 Retraining the Model Success

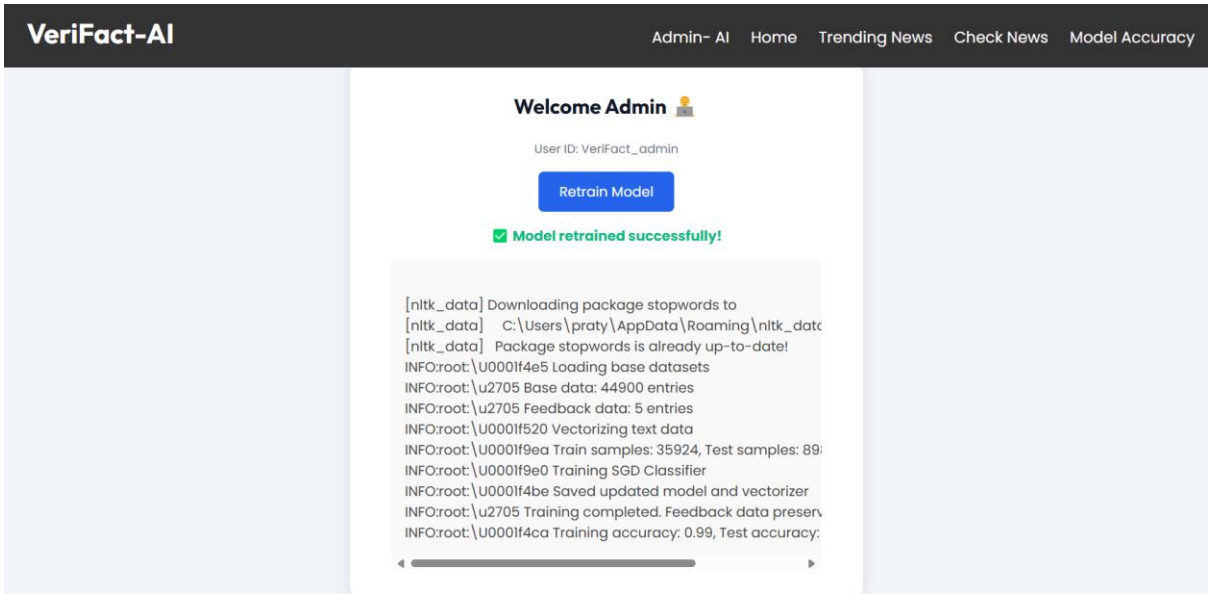

Fig. 8.12: Model retrain success

8.13 Model Accuracy page

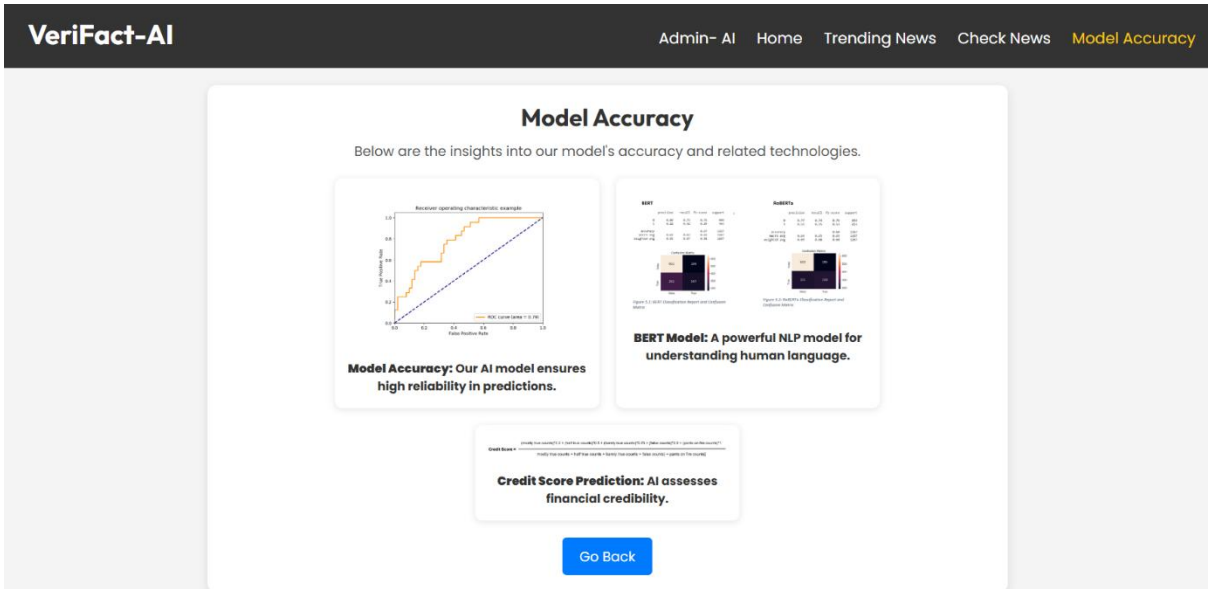

Fig. 8.13: Accuracy of model

CHAPTER 9

CONCLUSION

The Verifact AI project successfully demonstrates an advanced fake news detection system by integrating React.js for the frontend, Node.js and Express.js for backend operations, Flask (Python) for machine learning processing, and MongoDB for data storage. This system provides users with a reliable way to verify the authenticity of news articles, addressing the growing challenge of misinformation in digital media.

By leveraging Natural Language Processing (NLP) and machine learning models, the platform can analyze textual content and determine the likelihood of false information. The combination of React.js and MongoDB ensures a smooth and scalable user experience, while Flask's lightweight and flexible architecture enables efficient deployment of the trained model.

With secure authentication, real-time analysis, and cloud-based deployment, Verifact AI is a step forward in combating fake news. Future enhancements may include improving model accuracy, expanding dataset sources, and integrating real-time fact-checking APIs to further enhance credibility.

This project highlights the power of AI-driven solutions in ensuring the reliability of information and promoting responsible digital media consumption.

CHAPTER 10

FUTURE SCOPE

The future scope of VeriFact AI is broad and promising, given the rising concerns over misinformation and its impact on society. As the platform evolves, one major area of enhancement is the integration of multilingual support, allowing the system to detect fake news in various regional and international languages, thereby expanding its reach to a more diverse audience. This would be particularly useful in countries with multiple spoken languages where local misinformation can spread rapidly.

Another potential advancement is the inclusion of image and video verification, where the system could analyze visual content to detect manipulated media or deepfakes. This would make VeriFact AI a more comprehensive fake news detection platform, capable of handling different formats of misinformation beyond just text. Furthermore, real-time browser extensions or mobile apps could be developed to allow users to instantly check the authenticity of news while browsing or using social media.

The platform can also be enhanced with user feedback and learning mechanisms, where the model continuously improves its accuracy based on user interactions and reported outcomes. Incorporating blockchain technology for secure tracking and logging of news sources could further increase transparency and trust. Integration with popular news aggregators, search engines, and social media APIs could also enable automatic flagging of unverified content at the point of consumption.

From a broader perspective, VeriFact AI can be adapted for policy-making and public monitoring, where governments and organizations use the platform to track misinformation trends and respond accordingly. As artificial intelligence continues to advance, future versions of the system could incorporate contextual understanding, sarcasm detection, and sentiment analysis to further refine prediction accuracy.

In summary, VeriFact AI has the potential to evolve into a powerful, multi-functional tool for real-time misinformation detection, education, and policy support—contributing significantly to a more informed, secure, and digitally responsible society.

CHAPTER 11

REFERENCE

[1] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, Fake News Detection on Social Media: A Data Mining Perspective, 2017.

[2] H. Rashkin, E. Choi, J. Y. Jang, S. Volkova, and Y. Choi, Truth of Varying Shades: Analyzing Language in Fake News and Political Fact-Checking, 2017.

[3] M. R. Rodrigues, R. C. Barros, and A. L. Oliveira, A Deep Learning Approach for Fake News Detection in News Articles, 2019.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019.

[5] X. Zhou and R. Zafarani, Fake News Detection: A Survey, 2021.

[6] N. Hassan, G. Zhang, and C. Li, Detecting Check-worthy Factual Claims in Political Discourse, 2017.

[7] H. Ahmed, I. Traore, and S. Saad, Detection of Online Fake News Using N-Gram Analysis and Machine Learning Techniques, 2017.

[8] D. Lowd and P. Domingos, Active Learning with Support Vector Machines, 2005.

[9] V. Pérez-Rosas, B. Kleinberg, A. Lefevre, and R. Mihalcea, Automatic Detection of Fake News, 2018.

[10] J. Wolf, T. Debut, V. Sanh, et al., Transformers: State-of-the-Art Natural Language Processing, 2020.

[11] M. Grinberg, Flask Web Development: Developing Web Applications with Python (2nd ed.), 2018.

[12] T. Holowaychuk, Express - Node.js Web Application Framework, 2010.

[13] Node.js Foundation, Node.js JavaScript Runtime, 2009.

[14] MongoDB, Inc., MongoDB: The Application Data Platform, 2009.

[15] C. Ahmed, Fake and Real News Dataset, 2019.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2019.