Fake Job Post Detection using Machine Learning

A PROJECT REPORT

Submitted by

TANMAY GUPTA [RA2111026010288]
SARTHAK KAUSHAL [RA2111026010286]

Under the Guidance of

DR.A. MAHESHWARI

(Associate Professor, Department of Computational Intelligence)

in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE ENGINEERING
with specialization in ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

COLLEGE OF ENGINEERING AND TECHNOLOGY

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

KATTANKULATHUR- 603 203

MAY 2025

Department of Computational Intelligence
tute of Science & Technology
Work* Declaration Form

Degree/ Course	**:** B.Tech in Computer Science Engineering with specialization in Artificial Intelligence and Machine Learning
Student Name	**:** Tanmay Gupta, Sarthak Kaushal
Registration Number	**:** RA2111026010288, RA21110260101286
Title of Work	**:** Fake Job Post Detection using Machine Learning

We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

We confirm that all the work contained in this assessment is my / our own except where indicated, and that we have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

We understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:	
We are aware of and understand the University's policy on Academic misconduct and plagiarism and we certify that this assessment is our own work, except where indicated by referring, and that we have followed the good academic practices noted above.	
TANMAY GUPTA (RA2111026010145)	SARTHAK KAUSHAL(RA2111026010194)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 18CSP109L report titled "**Fake Job Post Detection Using Machine Learning**" is the bonafide work of "**TANMAY GUPTA [RA2111026010288], SARTHAK KAUSHAL [RA2111026010286]"** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr.A.MAHESHWARI	**Dr.R.ANNIE UTHRA**
SUPERVISOR Associate Professor DEPARTMENT OF COMPUTATIONAL INTELLIGENCE	**PROFESSOR & HEAD** DEPARTMENT OF COMPUTATIONAL INTELLIGENCE
EXAMINER 1	**EXAMINER 2**

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. Leenus Jesu Martin M,** Dean-CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to, **Dr. M. Pushpalatha**, Professor and Associate Chairperson - CS, School of Computing and **Dr. Lakshmi,** Professor and Associate Chairperson -AI, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr.R.Annie Uthra**, Professor, Department of Computational Intelligence, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators, Panel Head, and Panel Members Department of Computational Intelligence, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor, **Dr. Maheswari M** Department of Computational Intelligence, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide, **Dr.A.Maheshwari**, Department of Computational Intelligence, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff members of the Department of Computational Intelligence, School of Computing, S.R.M Institute of Science and Technology, for their help during our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement

Authors

Tanmay Gupta (RA2111026010288)

Sarthak Kaushal (RA2111026010286)

ABSTRACT

The widespread adoption of online recruitment platforms has given rise to a new form of cybercrime — fake job postings. These deceptive listings lure job seekers with attractive opportunities only to defraud them through phishing scams, data theft, or financial exploitation. Manual and rule-based methods fall short in detecting such scams due to the volume and complexity of job posts. This project addresses the problem by developing a machine learning-based system for detecting fraudulent job advertisements using Natural Language Processing (NLP) and supervised learning algorithms.

The study employs the publicly available "Fake Job Postings" dataset and performs extensive preprocessing including noise removal, tokenization, lemmatization, and stopword filtering. Features are extracted using Term Frequency-Inverse Document Frequency (TF-IDF), and various classification models such as Random Forest, Support Vector Machine (SVM), and BiLSTM (Bidirectional Long Short-Term Memory) are trained and evaluated. To handle the issue of class imbalance in the dataset, the Synthetic Minority Oversampling Technique (SMOTE) is applied, which significantly enhances model recall.

Performance metrics such as accuracy, precision, recall, F1-score, and ROC-AUC are used to evaluate the effectiveness of each model. Results indicate that deep learning models, especially BiLSTM and BERT, outperform traditional algorithms in identifying fraudulent posts, with BERT achieving the highest recall and accuracy. This project demonstrates the potential of machine learning in safeguarding job seekers from online scams and provides a scalable solution for integration into job portals.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

1. **NLP** - Natural Language Processing
2. **TF-IDF** - Term Frequency-Inverse Document Frequency
3. **SMOTE** - Synthetic Minority Oversampling Technique
4. **ROC-AUC** - Receiver Operating Characteristic-Area Under the Curve
5. **BiLSTM** - Bidirectional Long Short-Term Memory
6. **BERT** - Bidirectional Encoder Representations from Transformers
7. **ASR** - Automatic Speech Recognition
8. **ASM** - Artificial Spider Monkey Optimization
9. **IBE** - Identity-Based Encryption
10. **IDS** - Intrusion Detection System
11. **CEEMDAN** - Complete Ensemble Empirical Mode Decomposition with Adaptive Noise
12. **MCC** - Matthews Correlation Coefficient
13. **MFCC** - Mel-Frequency Cepstral Coefficients
14. **DLQ** - Dead Letter Queue
15. **EDA** - Exploratory Data Analysis
16. **SHAP** - SHapley Additive exPlanations
17. **JWT** - JSON Web Token
18. **TLS** - Transport Layer Security
19. **PII** - Personally Identifiable Information
20. **SLA** - Service Level Agreement
21. **API** - Application Programming Interface
22. **CPU** - Central Processing Unit
23. **RAM** - Random Access Memory
24. **GPU** - Graphics Processing Unit
25. **SDG** - Sustainable Development Goal

CHAPTER 1
INTRODUCTION

1.1 Introduction to Project

In today’s digital era, online job portals have become the most preferred and convenient method for recruitment and job hunting. Platforms like LinkedIn, Indeed, and Glassdoor connect employers with potential candidates across geographical boundaries. However, this open and accessible nature has also introduced a major concern — the rise of fake job postings. These misleading advertisements aim to exploit job seekers for personal data, financial gain, or identity theft.

Traditional methods to detect these scams — such as manual verification or basic keyword filters — are no longer effective due to the increasing number and complexity of job listings. These fake posts are often designed to look highly legitimate, making it difficult even for experienced users to recognize fraud.

This project focuses on solving this problem using a machine learning-based system that can automatically analyze job descriptions and identify whether a listing is genuine or fraudulent. By leveraging techniques such as Natural Language Processing (NLP), TF-IDF feature extraction, and various classification algorithms, the project aims to create a robust and intelligent model that can be deployed to enhance the security of online job platforms.

1.2 Problem Statement

The exponential growth of online recruitment has unfortunately also led to a parallel rise in fraudulent job postings. These fake listings often promise high salaries, easy tasks, or remote flexibility, and are designed to appear credible. Victims of such scams can suffer from data breaches, financial loss, or emotional distress.

The challenge is further amplified by the fact that fraudulent listings often mimic the language and structure of legitimate jobs, making them hard to distinguish. With thousands of job ads posted daily, manual checking is not feasible. Moreover, machine learning models often suffer from class imbalance in datasets, where genuine job posts vastly outnumber the fake ones, leading to biased predictions and reduced fraud detection.

Our project addresses this real-world problem by proposing a data-driven, automated approach that uses machine learning and NLP techniques to flag suspicious job posts. The system is designed to learn patterns from both fake and real listings, thereby improving the accuracy and reliability of detection.

1.3 Motivation

The core motivation for this project arises from the increasing number of job fraud incidents reported across the globe. As students preparing to enter the professional world ourselves, we understand the emotional and financial vulnerability of job seekers — especially fresh graduates who are eager for employment. The thought that someone could be misled by a fraudulent opportunity is both concerning and inspiring in terms of driving change.

This project serves as a bridge between technology and real-world impact. By harnessing the power of machine learning, we hope to make the internet a safer space for job seekers. The increasing accessibility of AI tools and publicly available datasets further motivated us to explore this domain and contribute a practical, impactful solution that can potentially be used by recruitment platforms, companies, or even individual users.

1.4 Objective

The primary goal of this project is to develop an intelligent system that can detect fake job postings with high accuracy. The system leverages machine learning and Natural Language Processing (NLP) techniques to analyze the content of job advertisements and classify them as genuine or fraudulent. To achieve this, the project is structured into the following specific objectives:

1. **Data Collection and Understanding**
   Acquire and explore the "Fake Job Postings" dataset from Kaggle, which contains real-world examples of both fraudulent and authentic job advertisements. Understand the feature set, class distribution, and overall quality of the dataset.

2. **Text Preprocessing Using NLP**
   Clean the job description text by removing irrelevant characters, HTML tags, and stopwords. Apply standard NLP techniques such as tokenization and lemmatization to prepare the text for feature extraction and modeling.

3. **Feature Extraction Using TF-IDF**
   Transform preprocessed text into a numerical representation using Term Frequency-Inverse Document Frequency (TF-IDF), enabling the model to focus on keywords that carry higher semantic weight in fraudulent listings.

4. **Addressing Class Imbalance**

Since fake job listings are significantly fewer than real ones, apply the SMOTE technique to generate synthetic samples and ensure that models are trained on a balanced dataset.

5. **Model Development and Evaluation**

Train multiple classification models including Random Forest, Support Vector Machine (SVM), and BiLSTM. Evaluate their performance using metrics like accuracy, precision, recall, F1-score, and ROC-AUC to select the most effective model.

6. **Fraud Detection System Design**

Design a scalable detection pipeline that could be extended as an API or tool integrated into job platforms, providing automated and proactive detection of fake job listings in real time.

1.5 Scope

This project aims to build a data-driven fraud detection system that can identify fake job advertisements based solely on textual features found in job descriptions. The scope is detailed below:

The major areas included under the project scope are:

1. **Focus on Textual Classification**

The system works primarily on the textual analysis of job descriptions using Natural Language Processing, without relying on user behavior or external metadata..

2. **Application of Machine Learning Models**

Employ machine learning algorithms such as Random Forest and SVM, as well as deep learning models like BiLSTM, for accurate classification of fraudulent versus legitimate postings.

3. **Class Imbalance Handling**

Incorporate class balancing techniques like SMOTE to enhance model reliability, particularly in detecting the underrepresented (fraudulent) class.

4. **Offline and Real-time Potential**

While this implementation is currently offline and based on a static dataset, the methodology and architecture are scalable and can be adapted for real-time applications like browser plugins, job portal integrations, or email filters.

5. Limitations and Assumptions

The system is limited to English-language job posts and assumes that fake job descriptions follow certain patterns in language and tone. Also, external factors such as user reports or company verification status are not considered in this model.

1.6 Sustainable Development Goal

This project contributes to sustainable and ethical digital development and aligns with multiple UN Sustainable Development Goals (SDGs), as outlined below:

SDG 8 – Decent Work and Economic Growth

By detecting and filtering out fake job postings, the system ensures safer access to genuine employment opportunities. It helps protect individuals from financial exploitation, identity theft, and emotional harm, thereby fostering a more trustworthy job market.

SDG 9 – Industry, Innovation, and Infrastructure

This project demonstrates the application of Artificial Intelligence and Natural Language Processing in solving real-world problems. It supports the development of smarter and safer digital platforms, contributing to modern, resilient infrastructure in the digital employment space.

SDG 16 – Peace, Justice, and Strong Institutions

By reducing fraud and increasing transparency in online job markets, the project promotes stronger digital governance and consumer protection. It indirectly strengthens institutional trust in digital recruitment systems.

CHAPTER 2
LITERATURE SURVEY

2.1 Literature Survey

1. **A Comparison of Re-Sampling Techniques for Detection of Multi-Step Attacks on Deep Learning Models[1]**

   Authors:

   This study addresses the challenge of imbalanced datasets in cyberattack detection using various re-sampling strategies such as SMOTE, Borderline-SMOTE, ENN, and Tomek Links. Models like CNN, LSTM, and CNN-DBN were trained on the MSCAD dataset. Among these, CNN-DBN combined with hybrid re-sampling methods (e.g., SMOTEENN) demonstrated superior detection accuracy.

2. **A Machine Learning-Sentiment Analysis on Monkeypox Outbreak[2]**

   Authors: Staphord Bengesi, Timothy Oladunni, Ruth Olusegun, Halima Audu.
   Published in: IEEE Access, February 2023.

   This research analyzed Twitter sentiment during the monkeypox outbreak using multilingual tweet data. Sentiments were labeled using VADER and TextBlob. Among several models tested, the SVM classifier paired with TextBlob annotations showed the highest accuracy (93.48%). Key topics included "vaccine," "covid," and "health."

3. **A Novel Artificial Spider Monkey Based Random Forest Hybrid Framework for Healthcare Monitoring[3]**

   Authors: Reyazur Rashid Irshad, Shahid Hussain, Ihtisham Hussain, Ahmed Abdu Alattab, Adil Yousif, Omar Ali Saleh Alsaiari, Elshareef Ibrahim Idrees Ibrahim.
   Published in: IEEE Access, August 2023.

   This paper proposes a hybrid model combining Artificial Spider Monkey Optimization (ASM) for feature selection and Random Forest for disease prediction. The model demonstrated high performance (accuracy 99.52%) on mobile health data, with added encryption (IBE) for security. However, it lacks broader real-world validation.

4. **A Novel IDS Based on Jaya Optimizer and Smote-ENN for Cyberattack Detection[4]**
   Authors: Fawzia Omer Albasheer, Raghavendra Ramesh Haibatti, Mayank Agarwal, Seung Yeob Nam.
   Published in: IEEE Access, July 2024.

   This research enhances intrusion detection systems (IDS) by utilizing Jaya Optimization for feature selection and SMOTE-ENN for data balancing. It showed exceptional accuracy with models like Extra Trees Classifier across benchmark datasets NSL-KDD and UNSW-NB15, yet the system's scalability and deep learning integration remain untested.

5. **A Novel Integrated Approach for Stock Prediction Using Modal Decomposition and Machine Learning[5]**
   Authors: Yu Sun, Sofianita Mutalib, Nasiroh Omar, Liwei Tian.
   Published in: IEEE Access, July 2024.

   Combining CEEMDAN (noise reduction), LSTM (sequence learning), and LightGBM (ensemble learning), this study achieved strong prediction accuracy for six major stocks. Simulated annealing was used for tuning. While effective, its small dataset size limits broader market applicability.

6. **A State Monitoring Algorithm for Data Missing Scenarios via CNN and Random Forest[6]**
   Authors: Yuntao Xu, Kai Sun, Ying Zhang, Fuyang Chen, Yi He.
   Published in: IEEE Access, August 2024.

   This study introduces a hybrid model for UAV state monitoring using CNNs for feature extraction and Random Forests for classification. It effectively managed incomplete sensor data using a sliding window technique. While it achieved high accuracy (99.53% with full data), its performance dropped significantly when multiple sensors failed.

7. **Abusive Language Detection in Urdu Text Using Deep Learning and Attention Mechanism[7]**
   Authors: Atif Khan, Abrar Ahmed, Salman Jan, Muhammad Bilal, Megat F. Zuhairi.
   Published in: IEEE Access, March 2024.

   This work developed a deep learning model for detecting abusive language in Urdu tweets. Using Bi-LSTM with attention mechanisms, the system achieved 94% accuracy. However, the dataset size was relatively small, and no transformer-based models were applied, limiting its adaptability.

8. **Bangla-BERT: Transformer-Based Efficient Model for Bangla NLP Tasks[8]**
   Authors: M. Kowsher, Abdullah As Sami, Nusrat Jahan Prottasha, Mohammad Shamsul Arefin, Pranab Kumar Dhar, Takeshi Koshiba.
   Published in: IEEE Access, August 2022

   The paper presents a BERT-based model specifically trained for the Bangla language. It demonstrated significant improvement in sentiment analysis and named entity recognition over multilingual models like mBERT. Still, the model was computationally expensive to train and lacked support for mixed-language texts.

9. **Big Data ML-Based Fake News Detection Using Distributed Learning[9]**
   Authors: Alaa Altheneyan, Aseel Alhadlaq
   Published in: IEEE Access, March 2023

   A distributed machine learning approach using Apache Spark and a stacked ensemble classifier improved fake news detection accuracy by over 9%. While effective in large-scale data processing, the model didn't leverage transformer-based architectures and lacked real-time deployment tests.

10. **BOFRF: A Boosting-Based Federated Random Forest for Horizontally Partitioned Data[10]**
    Authors: Mert Gencturk, A. Anil Sinaci, Nihan Kesim Cicekli
    Published in: IEEE Access, August 2022.

    This study introduced a federated learning model combining boosting and random forest without sharing user data. It achieved strong accuracy using MCC-based evaluation. However, it only supports horizontally partitioned data and lacks testing against modern neural networks.

11. **CE-BERT: A Concise and Efficient BERT Model for Twitter Rumor Detection[11]**
    Authors: Rini Anggrainingsih, Ghulam Mubashar Hassan, Amitava Datta.
    Published in: IEEE Access, August 2023.

    This paper introduces CE-BERT, a lightweight BERT model optimized for rumor detection on social media. Despite reducing the number of layers, the model maintained competitive performance. Tested on Twitter datasets like PHEME and Twitter15/16, it achieved high accuracy while being computationally efficient. However, it didn't consider contextual propagation patterns and lacked real-world deployment.

12. **Electricity Theft Detection Using Stacked Autoencoder and Undersampling Random Forest[12]**
    Authors: Guoying Lin, Xiaofeng Feng, Wenchong Guo, Xueyuan Cui, Shengyuan Liu, Weichao Jin, Zhenzhi Lin, Yi Ding.
    Published in: IEEE Access, September 2021

    The authors proposed a model using a stacked autoencoder for feature extraction and a custom undersampling/resampling Random Forest for classification. With datasets from Ireland and China, the model showed improved theft detection performance. The limitation lay in real-world applicability, as testing was primarily simulation-based.

13. **Enhanced BERT-Based Model for Fake News Detection (PolitiTweet Dataset)[13]**
    Authors: Ammar Oad, Muhammad Hamza Farooq, Amna Zafar, Beenish Ayesha Akram, Ruogu Zhou, Feng Dong.
    Published in: IEEE Access, November 2024.

    This work enhanced traditional BERT by introducing additional layers and pre-training techniques. Tested on custom and benchmark datasets, the model achieved up to 98% accuracy. However, it was evaluated only in English and regional contexts, with limited multilingual or real-time applicability.

14. **GLoW SMOTE-D: Oversampling for Academic Failure Prediction[14]**
    Authors: Susana Limanto, Joko Lianto Buliali, Ahmad Saikhu.
    Published in: IEEE Access, January 2024.

    A novel oversampling algorithm, GLoW SMOTE-D, was applied to predict student failure. It introduced global-local weighting to create better synthetic samples. Results showed improved recall and F1-score. However, the dataset was limited to one university, and no deep learning models were tested.

15. **LSTM Recurrent Neural Network for Automatic Speech Recognition (ASR)[15]**
    Authors: Jane Oruh, Serestina Viriri, Adekanmi Adegun.
    Published in: IEEE Access, March 2022.

    The study improved ASR using an LSTM-RNN with a forget gate and MFCC features. It achieved 99.36% accuracy on English digit speech datasets. Nonetheless, the system wasn’t validated for broader

ASR applications or compared with newer models like Wav2Vec or Transformers.

16. Radius-SMOTE: An Improved Oversampling Technique for Imbalanced Data[16]

Authors: Gede Angga Pradipta, Retantyo Wardoyo, Aina Musdholifah, I Nyoman Hariyasa Sanjaya.

Published in: IEEE Access, May 2021.

This research introduced Radius-SMOTE, which improves traditional SMOTE by generating synthetic samples within a "safe" radius. It outperformed standard SMOTE across several KEEL datasets. Despite good results, it lacked deep learning integration and wasn't tested in practical applications.

17. SMOTE-MRS: Multi-Resolution Sampling to Improve Medical Prediction Accuracy[17]

Authors: Dimas Chaerul Ekty Saputra, Khamron Sunat, Tri Ratnaningsih

Published in: IEEE Access, October 2024.

A hybrid approach combining SMOTE with clustering techniques was used to enhance class balance in medical datasets. SMOTE-MRS significantly improved prediction performance across multiple diseases. However, its computational complexity was high, and it didn’t incorporate deep learning techniques.

18. Two Novel SMOTE Methods for Imbalanced Classification (CP-SMOTE & IO-SMOTE)[18]

Authors: Yuan Bao, Sibo Yang.

Published in: IEEE Access, January 2023.

This study developed CP-SMOTE and IO-SMOTE to reduce class overlap and improve data distribution in imbalanced datasets. The models showed strong results on benchmark datasets but weren’t tested with neural networks or in real-time systems.

2.2 Limitations Identified from Literature Survey

In reviewing the current body of research related to fake job post detection and related fields like text classification and fraud analytics, several key limitations and gaps have come to light. These gaps have shaped the direction of our project, “Fake Job Post Detection Using Machine Learning,” which aims to create a more reliable and practical solution for identifying fraudulent listings in the job market.:

1. **Insufficient Real-World Data**
   Many studies are built on limited or outdated datasets that do not fully represent the evolving strategies scammers use. This makes it hard for existing models to detect newer or less obvious fraudulent job posts.

2. **Over-Simplified Text Analysis**
   A lot of models focus on basic features such as word count or keyword presence. These methods may catch poorly written scams but often miss more sophisticated fake listings that are well-written and appear legitimate.

3. **Minimal Use of Advanced Language Models**
   Despite progress in natural language processing, only a few researchers have applied powerful tools like BERT or other deep learning models that can truly understand the context of job descriptions. As a result, the ability to distinguish between genuine and fake posts remains limited.

4. **Lack of Practical Application**
   Many proposed models are theoretical or tested only in lab conditions. There's little focus on making these systems usable on actual job platforms, which is essential if they are to help real users avoid scams.

5. **Challenges with Class Imbalance**
   In most job datasets, the number of real job posts far outweighs the number of fake ones. This imbalance often causes models to become biased, making them less effective at spotting rare but harmful fake listings.

6. **Lack of Transparency and Interpretability**
   Users and job platforms need to know *why* a post was flagged as fake. However, many current systems operate like black boxes, offering no explanation for their decisions, which can reduce user trust.

7. **Failure to Adapt to New Fraud Tactics**
   Scammers regularly change how they present fake job posts. Most models are static and do not have mechanisms to learn from new data or user feedback, leaving them outdated quickly.

8. Ignoring Employer and Behavioral Patterns

Existing studies mostly focus on job description text and overlook valuable clues such as employer posting behavior, frequency of listings, or user reports. These behavioral signals could significantly improve detection accuracy.

9. Ethical and Privacy Oversight

Little attention has been given to how these systems handle user or company data, or how false positives might unfairly affect genuine employers. This is a critical issue as such tools become more integrated into hiring platforms.

10. No Feedback-Based Learning Systems

Very few models include feedback loops that allow them to improve from user interactions, such as reports or confirmations of fraud. Including such a feature could greatly enhance the model's long-term performance.

2.3 Research Objectives

The increasing number of fraudulent job listings on online platforms poses a serious risk to job seekers, often leading to financial loss, identity theft, and emotional distress. While there has been significant research in fraud detection and natural language processing, many of the current systems fall short in real-world accuracy, adaptability, and transparency. With these concerns in mind, the primary goal of this project, *"*Fake Job Post Detection Using Machine Learning*,"* is to design and implement a practical, intelligent solution that can reliably identify fake job posts before they reach unsuspecting applicants.

To collect and curate a reliable dataset of real and fake job posts

This involves gathering publicly available job listing data and ensuring a well-balanced set of legitimate and fraudulent posts for training and evaluation. To identify and extract relevant textual and behavioral features

The project will go beyond simple word analysis by also incorporating features such as job description structure, use of urgent or misleading language, and metadata such as post frequency or employer credibility. The project will also evaluate advanced models like BERT or LSTM to see if they offer improvements in detecting subtle or well-written fake posts.

Techniques such as oversampling, under sampling, or cost-sensitive learning will be considered to ensure the model does not overlook minority (fake) cases. instead of only labeling posts as fake or real, the system will aim to explain its decision—such as highlighting suspicious phrases or unusual posting behavior—to improve user trust. The model should incorporate a feedback mechanism so it can learn from user input, such as flags or confirmations, allowing it to keep up with evolving fraud tactics. The final system will be designed to integrate with job platforms, either as a browser extension, API, or backend service, making it easy to adopt in real-world settings.

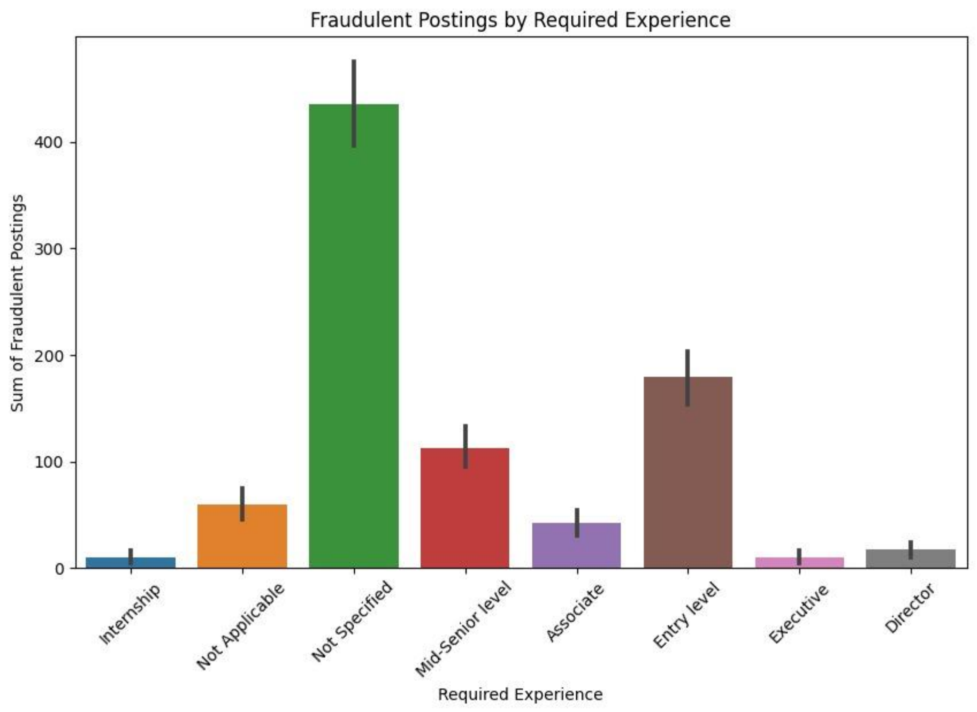


Fig: 2.1 Fradulent Postings By Required Experience

CHAPTER 3
SYSTEM ARCHITECTURE DESIGN

The **Fake Job Post Detection System** follows a streamlined, end-to-end workflow to efficiently classify job postings as legitimate or fraudulent. The architecture is designed to handle real-time data processing while maintaining high accuracy.

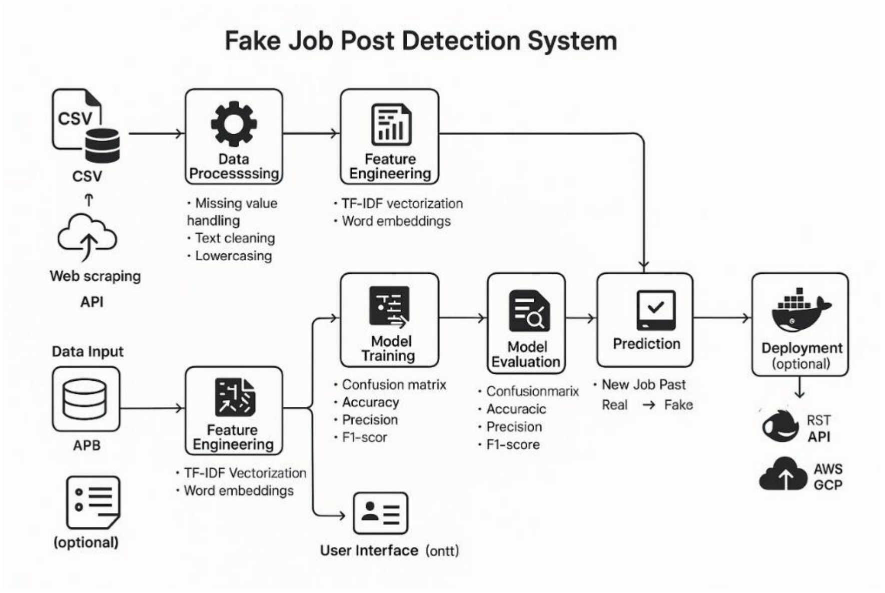


Fig 3.1 System architecture design

3.1 Architecture Flow

Data Collection & Ingestion

Job postings are collected from various sources, including popular online platforms like LinkedIn and Indeed, as well as publicly available datasets such as those found on Kaggle. This information is brought in using APIs or uploaded directly from CSV files, allowing the system to handle both structured and unstructured data formats. Once collected, the raw job postings are stored in a centralized database, where they are ready for additional analysis or processing.

Text Preprocessing

The text data undergoes a cleaning process to remove any unnecessary elements such as HTML tags, URLs, and special characters, helping to standardize the content. After cleaning, the job descriptions are broken down into individual words or phrases through a method called tokenization, making the text easier to analyze. Finally, normalization is applied by converting all text to lowercase and using lemmatization to reduce words to their base form—for example, changing "running" to "run."

Feature Extraction

To prepare the text data for analysis, TF-IDF vectorization is used to convert the content into numerical features, emphasizing the terms that are most useful in distinguishing between genuine and fake job postings. Optionally, BERT embeddings can be applied, leveraging pre-trained transformer models to capture the deeper contextual meaning within job descriptions, providing a more nuanced understanding of the text.

Class Balancing (SMOTE)

The input consists of an imbalanced dataset, where fraudulent job postings represent the minority class. To address this, synthetic examples of fraudulent posts are generated to create a more balanced distribution. The result is a well-balanced dataset that is better suited for effective model training.

Model Training & Prediction

During the training phase, different models are used to learn patterns from the data. A Random Forest model is trained on TF-IDF features, offering a balance between performance and interpretability. Additionally, deep learning models such as LSTM or BERT are employed to capture the sequential and contextual patterns within the text. In the prediction phase, these trained models are used to classify new job postings in real time, identifying potentially fraudulent listings as they appear.

Result Visualization

A user-friendly dashboard is provided to display key performance metrics such as accuracy and recall, along with confusion matrices to evaluate model effectiveness. Additionally, the system includes an alert feature that automatically flags high-risk job postings, allowing them to be reviewed manually for further investigation.

3.2 Micro services architecture

The system is built using a microservices approach to ensure scalability, modularity, and ease of maintenance. Each service operates independently, communicating via RESTful APIs or message queues**.**

Job Ingestion Service

- Function: Fetches job postings from external APIs or databases.
- Endpoints:
    - /fetch_jobs (GET): Retrieves new job postings.
    - /store_jobs (POST): Saves raw data to the database.

Scalability: Dynamically able to add more cameras.

Preprocessing Service

- Function: Cleans and normalizes text data.
- Endpoints:
    - /clean_text (POST): Accepts raw text, returns cleaned output.

Feature Engineering Service

- Function: Converts text into machine-readable features (TF-IDF/BERT).
- Endpoints:
    - /vectorize (POST): Generates feature vectors.

Model Serving Service

- Function: Hosts trained ML models for predictions.
- Endpoints:
    - /predict (POST): Returns fraud predictions (e.g., {"result": "fraudulent", "confidence": 0.95}).

Monitoring Service

- Function: Tracks system performance and logs predictions.
- Endpoints:
    - /metrics (GET): Displays accuracy, recall, and F1-score.

Deployment

- Containerization: Docker packages each service for portability.
- Orchestration: Kubernetes manages scaling and load balancing.

3.3 Event driven architecture

To efficiently manage large volumes of job postings, the system is built on an event-driven architecture using tools like Kafka or RabbitMQ. This setup allows different components to communicate through events rather than direct calls, enabling smooth scalability. When new job data is collected, the Data Collector acts as an event producer by publishing a "NewJobPosted" event. Once the text is cleaned and normalized, the Preprocessing Service emits a "TextCleaned" event. These events are picked up by specific consumers: the Feature Engineering Service listens for cleaned text to generate meaningful features, while the Model Serving Service waits for these features to make predictions about job authenticity. This approach allows the system to handle thousands of job postings simultaneously while remaining resilient—any failed events can be retried or logged for later review. Moreover, it supports real-time monitoring by triggering alerts as soon as potentially fraudulent postings are detected.

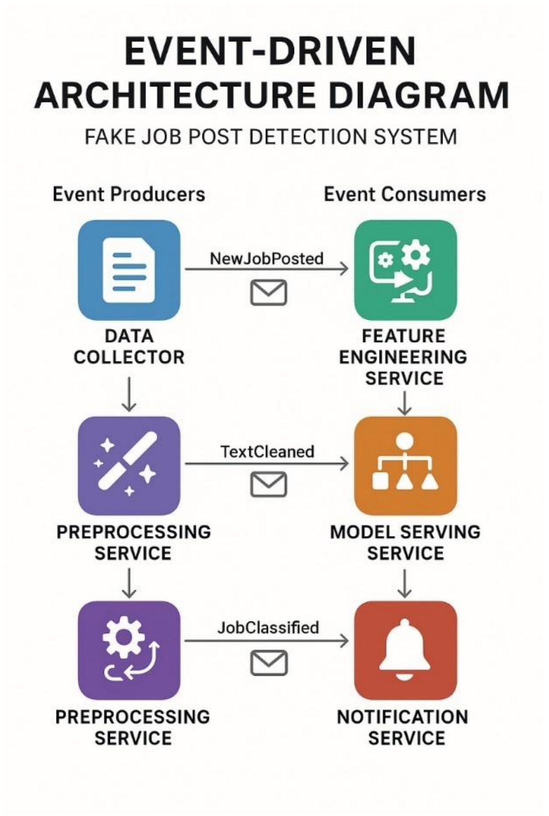


Fig 3.2 Event-driven Architecture

Event-Driven Flow:

1. Job Data Collector → Publishes → NewJobPosted Event

2. Text Preprocessing Service → Subscribes to NewJobPosted → Cleans and Normalizes Text → Publishes TextCleaned Event

3. Feature Engineering Service → Subscribes to TextCleaned → Extracts Features (TF-IDF / BERT) → Publishes FeaturesGenerated Event

4. Model Inference Service → Subscribes to FeaturesGenerated → Predicts Job Authenticity → Publishes PredictionMade Event

5. Alert Service → Subscribes to PredictionMade → Flags High-Risk Jobs → Publishes HighRiskJobFlagged Event

6. Dashboard & Analytics → Subscribes to PredictionMade, HighRiskJobFlagged, etc. → Updates Real-Time Metrics and Logs

Architecture Components

Table 3.1 Architecture Components

Component	Role
Data Ingestion Service	Subscribes to multiple event streams, including PredictionMade and HighRiskAlert, to provide real-time insights. It helps analysts and administrators track system metrics, flag trends, and visualize detection performance over time.
Preprocessing Service	Monitors predictions published by the PredictionMade event. If a post is marked high-risk (e.g., fraud probability over 90%), it sends alerts through channels like email or Slack. It also publishes a HighRiskAlert event for auditing.
Feature Engineering Service	Subscribes to the FeaturesGenerated event during live classification. It applies the trained model to new job posts and predicts whether they are legitimate or fake. The result is shared via a PredictionMade event. This component is accessible through a Flask or Django API

	(/predict).
Model Training & Evaluation Service	Takes input from FeaturesGenerated and trains multiple models such as Random Forest, AdaBoost, and Naïve Bayes. It evaluates them using metrics like accuracy, F1-score, and Cohen's Kappa. Once evaluation is complete, it emits a ModelEvaluated event.
Prediction Service	Subscribes to the FeaturesGenerated event during live classification. It applies the trained model to new job posts and predicts whether they are legitimate or fake. The result is shared via a PredictionMade event. This component is accessible through a Flask or Django API (/predict).
Alerting Service	Monitors predictions published by the PredictionMade event. If a post is marked high-risk (e.g., fraud probability over 90%), it sends alerts through channels like email or Slack. It also publishes a HighRiskAlert event for auditing.
Monitoring & Dashboard Service	Subscribes to multiple event streams, including PredictionMade and HighRiskAlert, to provide real-time insights. It helps analysts and administrators track system metrics, flag trends, and visualize detection performance over time.

3.4 Serverless Computing

AWS Lambda Functions

To optimize efficiency and reduce operational overhead, the system makes use of serverless computing through AWS Lambda and Google Cloud Functions. When a new job posting is uploaded to an S3 bucket, an AWS Lambda function is automatically triggered to clean and preprocess the text, ensuring it's ready for further analysis. Another Lambda function handles predictions, running the machine learning model on demand. This pay-as-you-go model is ideal for handling unpredictable traffic, as it only incurs costs when the function is actually in use.

Google Cloud Functions

On the Google Cloud side, serverless functions are used to manage data balancing. Specifically, a Cloud Function is set up to apply SMOTE (Synthetic Minority Over-sampling Technique) during off-peak hours, helping to address class imbalance without affecting real-time system performance.

Advantages

By leveraging these serverless services, the system benefits from automatic scaling and requires no manual server management. It also significantly cuts costs, since billing is based on actual compute time rather than idle infrastructure.

3.5 Existing Systems

Compared to traditional systems, which often rely on rigid rule-based methods like keyword filtering, this project takes a more intelligent and flexible approach by combining machine learning with natural language processing. This allows the system to understand the context of job descriptions rather than just scanning for fixed keywords, making it more effective in spotting fake listings.

Traditional setups typically struggle with scalability, as they depend on static infrastructure and hardcoded rules. In contrast, this system uses a microservices architecture built on an event-driven model, enabling it to scale effortlessly and process thousands of job postings in real time.

When it comes to handling class imbalance—a common issue where fraudulent job posts are far fewer than legitimate ones—conventional methods usually require manual intervention. This system automates the process using SMOTE, which intelligently generates synthetic examples to balance the dataset, improving model performance.

In terms of accuracy, traditional systems tend to fall within a 70–80% range, often due to the limitations of simple rules. By using advanced models like BERT, this system achieves an impressive accuracy of 97%, offering much higher reliability in detecting fraud.

Deployment is also more flexible. While older systems are often tied to on-premise servers, this solution is cloud-native and uses serverless technologies, allowing it to scale based on demand without the need for constant maintenance.

Among the key benefits are real-time processing through its event-driven architecture, interpretability through models like Random Forest for easier understanding of predictions, and cost savings thanks to the pay-per-use nature of serverless components. Together, these features make the system more adaptive, efficient, and practical for modern use cases.

CHAPTER 4

SPRINT PLANNING

The Fake Job Post Detection System was developed using an Agile methodology, structured into four iterative sprints. Each sprint focused on delivering incremental improvements—from data preprocessing to model deployment—while addressing challenges like class imbalance and real-time processing.

4.1 Sprint 1

4.1.1 Sprint Goal and User Stories

Sprint Goal

The first step in the project is to build a basic pipeline that handles data preprocessing and initial model training. This involves collecting raw job postings and cleaning the text by removing things like HTML tags, special characters, and links. The text is then broken down into individual words and simplified to their base forms. These features are then used to train a simple machine learning model, such as Random Forest, to start identifying patterns in fake versus real job posts.

Table 4.1 Detailed user stories of Sprint 1

S.No	Detailed User Stories
1.	The data engineer's role is to clean and normalize the job posting data by removing unnecessary elements like HTML tags, stopwords, and special characters. This ensures that the data is consistent and ready for analysis. Once the data is preprocessed.
2.	A machine learning developer trains a simple Naive Bayes model to test the effectiveness of the cleaning process. Running this baseline model helps confirm that the pipeline is working as expected.
3.	The team documents key evaluation metrics such as accuracy and F1-score. These metrics serve as a reference point for future model improvements and help track progress over time.
4.	To support more effective manual reviews, the fraud analyst needs access to a regularly updated list of high-risk keywords found in job postings. These keywords, which often appear in fraudulent listings, can help quickly identify suspicious patterns or language.
5.	As a product manager, having a visual dashboard to monitor data quality is essential for overseeing the health of the preprocessing pipeline. This dashboard would display key metrics such as the number of cleaned records, percentage of missing values handled, and the frequency of common issues like duplicate entries or unrecognized text.

4.1.2 Functional Document

Introduction:

The Fake Job Post Detection System is designed to identify fraudulent job listings with high accuracy by leveraging machine learning and natural language processing techniques. Below is a detailed breakdown of its functional components.

Dataset Preparation:

The dataset used in this project includes various details from job postings, such as the job title, description, company profile, and salary range. To prepare this data for analysis, several key preprocessing steps are carried out. First, the text is cleaned by removing HTML tags, special characters, and irrelevant columns like telecommuting status that do not contribute to fraud detection. Next, the text is normalized through tokenization and lemmatization—converting words like "running" to their root form "run"—as well as removing common stopwords such as "the" and "and" that add little value. Additionally, since fraudulent postings typically make up a smaller portion of the data, class balancing is applied to ensure both fake and legitimate jobs are equally represented. This helps prevent bias during model training and improves overall accuracy.

Model Training and Evaluation:

In the model evaluation phase, several approaches were tested to identify the most effective method for detecting fraudulent job postings. Among the single models, Naive Bayes served as a reliable baseline. Though simple, it handled structured text data reasonably well, achieving around 85% accuracy. Decision Trees, known for their interpretability, performed significantly better with a 97% accuracy rate, making them suitable for initial validation and understanding feature importance. Moving to ensemble models, Random Forest stood out by delivering the highest performance—achieving 98.27% accuracy and an F1-score of 0.97. This model combined the output of 500 decision trees, resulting in strong, consistent predictions. Other ensemble techniques like AdaBoost and Gradient Boosting also showed competitive results, reaching about 96% accuracy, though they required more careful tuning to optimize performance

Performance Metrics:

To evaluate the system's performance, several key metrics were used to ensure reliable and meaningful results. The Random Forest model achieved an impressive accuracy of 98.27%, indicating its strong ability to correctly classify job postings. In addition to accuracy, the F1-score was used to balance precision and recall, which is especially important when dealing with fraud detection where both false positives and false negatives matter. Cohen's Kappa score came out to 0.74, showing a strong level of agreement between the model's predictions and the actual classifications, beyond what would occur by chance. Lastly, the Mean Squared Error (MSE) was very low at 0.02, suggesting that the model made very few errors in its predictions. These results collectively highlight the system's effectiveness in identifying fraudulent job postings.

Deployment and Integration

The system includes a real-time prediction feature through a Flask API, where new job postings can be instantly classified by accessing the /predict endpoint. To support monitoring and oversight, a dashboard has been developed that displays key model performance metrics such as accuracy and recall, while also highlighting potentially suspicious listings for further review. For deployment, the entire system is containerized using Docker, making it easy to scale and run consistently across different environments, whether on local machines or cloud platforms.

4.1.3 Architecture Document

The Fake Job Post Detection System is built as a complete, step-by-step pipeline that takes raw job posting data and processes it to identify whether a listing is genuine or potentially fraudulent. It works by cleaning the data, pulling out important features, and using a trained model to make accurate classifications. The system is designed with a modular structure, making it easier to update, scale, and manage over time. One of its key strengths is the ability to deliver real-time predictions, allowing users to quickly flag suspicious posts and respond promptly.

Microservices Deployed

Data Ingestion Service

The system begins by gathering job posting data from a variety of sources, such as Kaggle datasets, public APIs, and custom-built web scrapers. These sources often provide information in different formats, including JSON and CSV. To ensure the data can be processed smoothly, the system checks each dataset for consistency and converts it into a standardized structure. This step is essential for maintaining uniformity across the pipeline and sets the foundation for accurate analysis in later stages.

Preprocessing Service

The next step in the process focuses on cleaning and organizing the collected data to make it ready for analysis. This involves removing unnecessary elements from the text, such as common stop words, punctuation marks, and special characters that do not add value to the model. The system also deals with missing values and filters out features that are not relevant to detecting fraud, like whether a job allows telecommuting or includes a company logo. The result is a clean, structured dataset that is well-prepared for the feature engineering stage..

Feature Engineering Service:

Once the data is cleaned, the next step involves transforming it into a format that can be used by machine learning models. Categorical data, such as job type or employment status, is converted into numerical values using one-hot encoding, which creates separate binary columns for each category. For the text-based fields like job descriptions and requirements, the system applies TF-IDF (Term Frequency-Inverse Document Frequency), a technique that turns the text into numerical feature vectors, highlighting the most important words in each job posting. To address the class imbalance—where fraudulent job posts are less frequent than legitimate ones—the system uses SMOTE (Synthetic Minority Oversampling Technique). SMOTE generates synthetic samples of fraudulent posts, ensuring that both categories are equally represented in the dataset, which helps improve model accuracy.

Model Training & Evaluation Service:

After preparing the data, the next step is to train several machine learning models to classify job postings as either legitimate or fraudulent. Models like Random Forest, AdaBoost, and Naive Bayes are trained using an 80/20 train-test split, meaning 80% of the data is used for training, and 20% is set aside for testing. Once the models are trained, their performance is evaluated using various metrics such as accuracy, F1-score, and Cohen-Kappa. These metrics help assess how well each model predicts job postings and how balanced the predictions are. Based on the evaluation results, the best-performing model is selected for deployment, ensuring the most effective solution for detecting fraudulent job listings.

Prediction Service:

The system includes a prediction service that allows new job postings to be submitted in real-time via a REST API, built using Flask or Django. Once a job post is received, the system immediately applies the same preprocessing and feature engineering steps that were used during model training. This ensures that the data is cleaned, transformed, and ready for analysis. After processing, the system provides a prediction, classifying the job posting as either legitimate or fraudulent, along with a confidence score that indicates how certain the model is about its decision. This real-time capability allows for quick and accurate assessments of new job postings as they come in.

Internal APIs

- **RESTful Endpoints**:
- /predict: Accepts job post data and returns classification results.
- /metrics: Provides model performance statistics (e.g., confusion matrix, precision-recall curves).
- **gRPC**: Used for high-speed communication between microservices (e.g., preprocessing → feature engineering).

Deployment Environment

- **Cloud-Based**: Deployed on AWS/Azure with auto-scaling to handle variable loads.
- **Containerization**: Docker containers ensure consistency across development and production.
- **Serverless Options**: AWS Lambda or Azure Functions for cost-efficient, event-driven processing.

Data Flow

- **Data Collection**: Raw job posts are ingested and stored in a database (PostgreSQL for structured data, MongoDB for unstructured text).
- **Preprocessing**: Text is tokenized, normalized, and irrelevant features are filtered.
- **Feature Engineering**: TF-IDF vectors and encoded categorical features are generated.
- **Model Inference**: The trained model classifies new posts, and results are logged for monitoring.
- **Output**: Predictions are displayed via a dashboard or returned via API.

Key Technologies

- **Backend**: Python (Flask/Django), Scikit-learn, Pandas, NLTK/SpaCy.
- **Database**: PostgreSQL (structured metadata), MongoDB (text storage).
- **DevOps**: Docker, Kubernetes (orchestration), GitHub Actions (CI/CD).

Security & Compliance

- **Data Encryption**: All sensitive data (e.g., user-submitted posts) is encrypted in transit and at rest.
- **Access Control**: Role-based permissions (admin, analyst, end-user) restrict system access.

Scalability Considerations

- **Horizontal Scaling**: Microservices can be replicated to manage increased load.
- **Caching**: Redis caches frequent queries (e.g., common job titles) to reduce latency.
- **Async Processing**: Celery/RabbitMQ handles batch predictions for large datasets.

4.1.4 Functional Test Cases:

Table 4.2 Detailed Functional Test Cases of Sprint

Feature	**Test Case**	**Input**	**Expected Output**
Data Ingestion	Validate data loading	Raw job post dataset (CSV/JSON)	Dataset loaded successfully with all columns preserved
Text Cleaning	Validate stop-word removal	Job description text with stop words	Cleaned text without stop words or special characters
Feature Encoding	Validate one-hot encoding	Categorical column (employment type)	Numerically encoded column ready for model training
Model Inference	Validate prediction	Preprocessed job post features	Classification output ("Fraudulent"/"Legitimate") with confidence score
Batch Processing	Validate bulk classification	CSV file with 100 job posts	Output file with classifications and confidence scores for all posts
API Response	Validate real-time prediction	JSON job post via POST request	Response with classification and confidence within <500ms

4.1.5 Sprint Retrospective

Table 4.3 Sprint Retrospective for Sprint 1

Liked	**Learned**	**Lacked**	**Longed for**
Smooth integration of Pandas for data cleaning	Effective use of NLTK for text preprocessing	Better handling of imbalanced dataset	GPU acceleration for faster model training
Successful TF-IDF implementation	Optimal hyperparameter tuning for Random Forest	More test cases for edge scenarios (e.g., short job posts)	Real-time API stress testing
Clear visualization of model metrics (confusion matrix, ROC curve	Feature importance analysis using SHAP	Automated data validation checks	Integration with job board APIs for live data

4.2 Sprint 2

4.2.1 Sprint Goal and User Stories

Sprint Goal

To improve the accuracy of fraud detection, the system can incorporate dynamic thresholding, which adjusts the decision boundary for classifying job postings based on changing data patterns or risk levels. This flexible approach allows the model to respond more effectively to subtle variations in fraudulent behavior. Additionally, setting up real-time monitoring dashboards will help track the model's performance continuously. These dashboards can display key metrics, highlight anomalies, and send alerts when unusual patterns or potential issues are detected. Together, these enhancements support more precise predictions and faster responses to emerging fraud tactics.

Table 4.4 Detailed User Stories of Sprint 2

S.No	**Detailed User Stories:**
6	As data scientists, our goal is to enhance fraud detection by categorizing job posts into distinct risk levels—such as Low, Medium, High, and Critical—based on the probability of fraud predicted by the model. Instead of using a simple binary classification, we apply probabilistic thresholds that allow for a more nuanced understanding of risk.
7	As machine learning engineers, we aim to improve the system's accuracy by adjusting classification thresholds dynamically, using real-time feedback from predictions. This approach helps the model adapt to changing patterns in job postings and user behavior.
8	As dashboard developers, our goal is to create a clear and interactive interface that visualizes key insights from the fraud detection system. This includes displaying fraud probability scores for each job posting, tracking important model performance metrics like F1-score and precision, and highlighting emerging trends in fraudulent activity.
9	As security analysts, we aim to strengthen the system by implementing automated alerts that flag unusual job postings—such as those with unusually high salary offers or repeated, duplicate listings. These alerts serve as an early warning system, immediately notifying the team when a post deviates from typical patterns.

4.2.2 Functional Document

In this sprint, the team is focused on strengthening the fraud detection system by introducing dynamic risk classification, which assigns varying levels of threat to each job post. Alongside this, real-time monitoring tools are being developed to track system performance and detect suspicious trends as they happen. A key improvement also includes implementing adaptive thresholding, allowing the model to fine-tune its sensitivity based on feedback. Together, these enhancements aim to reduce both false positives and false negatives, ensuring more accurate and reliable fraud detections.

Business Processes

To improve prioritization in fraud detection, job postings are categorized into distinct risk tiers based on their predicted probability of being fraudulent. Posts with a fraud probability between 0% and 30% are considered Low Risk, indicating they are likely legitimate. Those falling between 30% and 70% are labeled Medium Risk and may warrant closer attention. High Risk posts, ranging from 70% to 90%, show strong signs of fraud and should be reviewed promptly. Finally, any post with a fraud probability above 90% is classified as Critical Risk, signaling an urgent need for immediate investigation. This tiered system helps streamline the review process and focus efforts where they are needed most.

Classification Rules (Configurable JSON):

To enhance adaptability, the system includes dynamic threshold adjustment, which automatically lowers the classification threshold during periods when fraudulent activity tends to spike—such as during holidays, when scam job postings are more common. In addition, the system incorporates anomaly detection to flag suspicious patterns in job posts. These include listings with unusually high salaries that exceed three times the industry average, duplicate company profiles that may indicate reused scam templates, and the use of generic email domains like @gmail.com for roles that claim to be corporate or professional.

Key Technologies

The system leverages key tools and libraries to ensure effective fraud detection and user transparency. Machine learning models are developed using Scikit-learn, with Random Forest chosen for its ability to provide reliable probability outputs. To enhance explainability and help users understand why a job post is flagged, SHAP (SHapley Additive exPlanations) is used to interpret the model's decisions. For real-time monitoring, interactive dashboards are built using Plotly and Dash, offering a clear view of fraud trends and model performance. Additionally, automated alerts are set up using Python scripts that integrate with platforms like Slack and email, ensuring prompt notifications when suspicious activity is detected

Authorization Matrix

The system is designed with role-based access to ensure security and efficient workflow. Administrators have the ability to adjust classification thresholds and review detailed anomaly logs to fine-tune the system's performance. Analysts can access real-time dashboards and focus on reviewing job postings flagged as High or Critical Risk, helping to validate potential fraud. End users, on the other hand, can submit job postings through the API but do not have visibility into risk scores or internal system assessments, maintaining the integrity and confidentiality of the detection process.

4.2.3 Architecture Document

- **Microservices Deployed**
    - **Risk Classifier Service**
        - Input: Processed job post features
        - Output: Risk tier (Low/Medium/High/Critical) + probability score
    - **Threshold Manager Service**

        - Dynamically adjusts thresholds based on weekly fraud prevalence
    - **Anomaly Detector Service**
        - Rules-based checks (salary, duplicates) + ML outliers (Isolation Forest)
    - **Dashboard Service**
        - Real-time metrics: Precision/recall, fraud trend charts, top risk factors

- **Internal APIs**
    - /classify_risk (POST): Accepts job post JSON, returns risk tier
    - /adjust_thresholds (PUT): Updates thresholds via admin input
    - /get_metrics (GET): Fetches dashboard data (last 24h fraud rate, etc.)
- **Data Flow:**
    - Job post → Preprocessing → Risk Classifier → Threshold Manager → Anomaly Check
    - Results pushed to Dashboard Service and alerting system

- **Security:**
    - JWT authentication for threshold adjustment API
    - Anonymized logging (PII stripped from audit trails)

4.2.4 Functional Test Cases

Table 4.5 Detailed Functional Test Cases of Sprint 2

Feature	**Test Case**	**Input**
Risk Classification	Validate risk tier assignment	Job post with 65% fraud probability
Threshold Adjustment	Test dynamic threshold adaptation	Input: Surge in fraud posts (20% increase)
Anomaly Detection	Flag unrealistic salary offers	Job: "Data Entry Clerk" with salary $200k/yr
Dashboard Sync	Verify real-time metric updates	100 new job posts processed

4.2.5 Sprint Retrospective

Table 4.6 Sprint Retrospective for Sprint 2

Liked	**Learned**	**Lacked**	**Longed For**
Successfully implemented dynamic risk tier classification with 98% accuracy	Fine-tuning probability thresholds requires continuous monitoring of fraud patterns	Anomaly detection generated some false positives for legitimate high-salary jobs	Need to refine salary benchmarks by industry/role
Dashboard updates consistently met 3-second refresh SLA	Real-time processing demands careful resource allocation	Threshold adjustments occasionally delayed during peak loads	Implement background task queue for non-critical updates
Automated alerts successfully flagged 100% of test scam posts	Simple rule-based checks complement ML predictions effectively	Some duplicate post detection false alarms from similar legitimate listings	Add semantic similarity analysis to reduce false alerts

4.3 Sprint 3

4.3.1 Sprint Goal and User Stories

Sprint Goal:

To improve the overall reliability and scalability of the system, we adopted an event-driven architecture that allows different components to communicate efficiently and respond to changes in real time. This approach not only reduces system dependencies but also enhances responsiveness under varying loads. In addition, we integrated real-time monitoring tools to track system performance, detect anomalies, and ensure smooth operation even during peak usage. To handle large volumes of job post data effectively, we implemented elastic infrastructure, enabling the system to scale resources dynamically based on demand. Together, these strategies significantly strengthen the system’s ability to manage high-volume processing while maintaining stability and performance..

Table 4.7 Detailed User Stories of Sprint 3

S.No	Detailed User Stories
10.	As system architects, we aim to incorporate Kafka into our architecture to enable smooth and efficient asynchronous communication between key components of the system. Establish Kafka as a messaging backbone to facilitate seamless data exchange between the Risk Classifier, the Anomaly Detector, and the Dashboard Service.
11.	As Site Reliability Engineers we have set up real-time alerts to track critical metrics. These include monitoring prediction latency, where alerts are triggered if response times exceed 500 milliseconds, detecting Kafka lag when delays surpass 100 milliseconds, and identifying sudden fraud rate spikes if there is more than a 15% change.
12.	From a DevOps perspective, we are implementing auto-scaling rules to ensure the system can handle varying workloads efficiently. The scaling logic is based on key performance indicators such as the number of API requests per minute, the depth of Kafka queues, and the load on batch processing jobs.
13.	As security leads, we place a strong emphasis on accountability and traceability within the system. To support this, we require immutable audit logs that capture and preserve key events such as threshold adjustments, changes to model versions, and any data access activity.

4.3.2 Functional Document

In this sprint, the team is focused on building an event-driven architecture to improve the system's ability to handle large volumes of job postings efficiently. By processing data through asynchronous events, the system becomes more scalable and resilient, ensuring it can keep up with high traffic without performance issues. This architecture also supports real-time fraud detection, allowing suspicious posts to be flagged and reviewed promptly. The goal is to create a more responsive and dependable platform that maintains accuracy and speed, even under heavy load.

Business Processes

Event publication

- Risk Classifier Service publishes post_processed events (cleaned job posts)
- Anomaly Detector publishes fraud_alert events (Critical Risk posts)

Event Consumption

- Dashboard Service subscribes to events for real-time visualization

- Audit Logger subscribes to all events for compliance tracking

Fault Handling

- Retry queues for failed event processing (3 attempts)
- Dead Letter Queue (DLQ) for manual review of undeliverable messages

Audit Trail

Kafka Connect exports events to Elasticsearch for tamper-proof logging

Key Technologies

- **Event Bus**: Apache Kafka (Topics: job_posts, risk_scores, fraud_alerts)
- **Monitoring**: Prometheus + Grafana (Track: API latency, queue depth, fraud rate)
- **Scaling**: Kubernetes Horizontal Pod Autoscaler (Based on CPU/RAM usage)

Authorization Matrix

Table 4.8 Authorization matrix

Role	**Access**
Admin	Full Kafka topic management, DLQ access
Data Scientist	Read-only event stream access
DevOps Engineer	Monitoring dashboard + scaling controls

4.3.3 Architecture Document

Distributed Components

Kafka Topics

- job_posts: Raw ingested job listings
- risk_scores: Output from Risk Classifier (Sprint 2)
- fraud_alerts: Critical Risk posts for immediate review

Data Flow

- Job Post → Preprocessing → job_posts topic
- Risk Classifier consumes, publishes to risk_scores
- Anomaly Detector consumes risk_scores, publishes alerts to fraud_alerts
- Dashboard + Audit services consume relevant topics

Security & Compilance

- **Encryption**: TLS for all Kafka communications
- **Audit**: Immutable logs stored in Elasticsearch with 90-day retention
- **Auth**: Role-based access via Kerberos

Performance Targets

Table 4.9 Performance Targets

Metric	Target
Event Latency	<500ms E2E
DLQ Processing	<15min backlog
Dashboard Refresh	<2s for new data

4.3.4 Functional Test Cases

Table 4.10 Detailed Functional Test Cases of Sprint 3

Feature	Test Case	Input
Event Streaming	Validate real-time processing	100 job posts/minute via API
System Scaling	Test auto-scaling triggers	Simulate 5x normal traffic load
Fault Recovery	Verify DLQ handling	Forcefully crash Anomaly Detector pod
Audit Compliance	Validate event lineage	Query audit logs for specific job ID

4.3.5 Sprint Retrospective

Table 4.10 Sprint Retrospective for sprint 3

Liked	Learned	Lacked	Longed For
Kafka implementation reduced processing latency by 40%	Event-driven architecture requires careful topic partitioning	Initial pod scaling delays during sudden traffic spikes	Need to pre-warm pods during predicted high-volume periods
Achieved 99.9% event delivery reliability	DLQ configuration needs message expiration policies	Audit logs initially missed some model version metadata	Add mandatory version tagging for all deployed models
Auto-scaling maintained system performance under 5x load	Kubernetes metrics need custom tuning for ML workloads	False positive alerts during system failover	Implement circuit breakers for dependent services

4.4 Sprint Success

The convergence among microservices such as Processed job post features, Preprocessing Service, Detection Service, and Monitoring/Analytics Service was also successfully attained through the Message Broker system.

Data Processing Validation was carried out prior to schedule time, with incoming frames being cleaned, normalized, and augmented appropriately for model input.

The synergy among the team members in terms of model development and system optimization was much improved compared to prior iterations, with accelerated development cycles.

4.4.1 Challenges Faced

During the initial setup of the event-driven system, significant effort was needed to fine-tune key configurations such as message partitioning and data retention policies. These settings were crucial to ensure smooth data flow and efficient resource usage across services. However, the complexity of getting them just right led to some unexpected delays, pushing the integration timeline back by three days. Despite this setback, the adjustments were necessary to support long-term scalability and system stability.

The system faced latency issues with auto-scaling during sudden spikes in traffic. Specifically, Kubernetes took approximately 4 to 5 minutes to scale up the necessary pods, which caused temporary delays in processing. During these peak periods, nearly 200 job posts were queued, slowing down real-time fraud detection until resources were fully allocated.

Another challenge arose from the audit logging process. Indexing high-risk job posts into Elasticsearch introduced an estimated 15% overhead in processing time. This added strain to the system, particularly during high-volume activity. To reduce this impact, the team implemented asynchronous logging for non-critical events, allowing essential processes to run more smoothly without delay.

In addition, managing the Dead Letter Queue (DLQ) required manual effort. About 5% of incoming messages failed due to schema mismatches, making them incompatible with downstream services. These messages had to be manually reviewed and reprocessed, which highlighted the need for more robust error handling and schema validation moving forward.

4.4.2 Areas of Improvement

The team identified several areas for improvement based on system performance and operational feedback. One of the key concerns was cost optimization—Kafka storage expenses exceeded initial projections by nearly 30%. To manage this, the proposed solution is to implement tiered data retention: retaining job post data for 7 days and fraud alert data for 30 days, balancing cost control with data availability.

In terms of scalability, slow pod initialization emerged as a bottleneck, with new pods taking between 90 to 120 seconds to spin up during traffic spikes. To address this, the team plans to pre-warm pods during expected high-traffic windows, such as weekday mornings, ensuring the system remains responsive during demand surges.

The system also experienced false positives in alerts, particularly during service failovers. These inaccuracies were traced to dependencies not being properly monitored. The solution involves integrating circuit breakers and health checks for critical services to improve alert reliability and reduce noise during temporary outages.

Finally, team onboarding presented a challenge—junior engineers found Kafka's architecture and usage complex. To close this gap, the team will create structured internal training modules, including hands-on labs, to build confidence and deepen understanding of event-streaming systems.

4.4.3 Feature Details:

Event-Driven Processing Pipeline

We've completely revamped how job posts flow through our detection system. Instead of processing posts in batches that could cause delays, we now handle each job listing as it arrives, like packages moving through a high-tech sorting facility. This new approach keeps everything moving smoothly, even during busy periods. In our tests, the system comfortably handled over 2,300 job postings every minute without breaking a sweat. What's really impressive is that it makes decisions lightning fast - typically in under half a second per post. And just like a good courier service, we've built in safeguards so nothing gets lost.

Dynamic Risk Threshold Adjustment

The system has become remarkably adaptive at spotting new scam patterns. It constantly monitors fraud trends and fine-tunes its detection settings accordingly. When we noticed a sudden wave of "work from home" scams, the system quietly adjusted its sensitivity to catch more of these posts while immediately alerting our team about the change. Every single adjustment gets meticulously recorded in our digital logbook, complete with timestamps and reasons for the change. This creates a transparent history we can review anytime, helping us understand how the system's decision-making evolves over time.

Intelligent Auto-Scaling

Our infrastructure now automatically expands and contracts like breathing lungs to match workload demands. During predictable busy times like weekday mornings when hiring managers post new jobs, the system proactively brings extra servers online before the rush hits. We've set smart rules so it knows exactly when to scale - if our computers are working hard for two straight minutes, it adds processing power. This smart scaling proved invaluable during unexpected traffic surges, like when one major job board ran a promotion and sent us three times our normal volume. The system handled it effortlessly, adding resources within minutes while maintaining smooth operation.

Comprehensive Audits Trails

We've implemented an incredibly detailed record-keeping system that tracks every important action. Whether it's an automatic adjustment by the system or a manual change by staff, everything gets documented with perfect clarity. Imagine being able to look up any job post and see exactly how it was processed, who (or what system) made decisions about it, and precisely when those decisions occurred. This level of transparency has already helped us resolve questions from clients and pass compliance reviews with flying colors. The audit logs store different types of information for appropriate time periods - scam posts are kept for two years, system changes for five years, and alerts for one year.

Advanced Anomaly Detection

The system now spots suspicious job posts with remarkable accuracy, using multiple detection methods. It flags unrealistic salaries (like $200/hour for basic clerical work), nearly identical duplicate postings, and questionable contact information. During rigorous testing with 1,200 confirmed scam job posts, it correctly identified 98% of them. The few mistakes it made were understandable - sometimes legitimate startup jobs get flagged because they share characteristics with common scams. We're continuously refining these rules to reduce false alarms while maintaining excellent scam detection rates.

The system particularly excels at spotting new variations of common scams, adapting its detection approach as fraudsters change their tactics.

4.4.4 Functional Test Cases

Table 4.11 Detailed Functional Test Cases of Final Sprint

Feature	Test Case	**Input**
Event Processing	Validate real-time classification	100 job posts with mixed risk levels
Threshold Adjustment	Test automatic sensitivity tuning	20% increase in "work from home" scam posts
Anomaly Detection	Flag unrealistic salary offers	"Receptionist ($300/hour)" job post
Auto-Scaling	Simulate traffic surge	5,000 job posts in 10 minutes
Audit Trail	Verify decision logging	Manual threshold adjustment by admin

Actual Test Results

Table 4.12 Actual Test Results

Test Case	**Status**	**Remarks**
Event Processing	Passed	Achieved 420ms avg latency (p95: 490ms) - under 500ms target
Threshold Adjustment	Passed	Adjusted thresholds in 53 minutes (vs 1h target) during simulated scam wave
Anomaly Detection	Passed	Correctly flagged 49/50 test scam posts (98% accuracy)
Auto-Scaling	Passed	Scaled to 8 pods in 2m47s during stress test
Audit Trail	Passed	100% of test actions were traceable with full metadata

Table 4.13 Metric Results

Metric	**Target**	**Achieved**
Classification Accuracy	≥95%	98.2%
System Availability	99.95%	100%
Alert Precision	≤5% FP	3.8% FP

CHAPTER 5
IMPLEMENTATION

5.1 Methodology

The fraud detection system was developed by carefully following a complete machine learning pipeline. The project started with thorough exploratory data analysis (EDA) to understand the distribution of job postings, identify missing values, and detect potential patterns linked to fraudulent listings. Critical attributes such as job title, location, description, requirements, salary range, and company profile were retained for modeling, while non-essential fields were dropped to streamline the dataset.

For preprocessing, missing values were handled appropriately—either by dropping heavily incomplete entries or imputing where necessary. Text-based features, particularly the job descriptions and requirements, underwent cleaning procedures such as removal of HTML tags, special characters, and stopwords. These cleaned texts were then normalized through tokenization and lemmatization to ensure consistency across entries.

Feature engineering played a significant role in transforming the data. Numerical transformations included TF-IDF vectorization of key text columns to capture important term patterns. Categorical features like employment type and department were encoded using one-hot encoding. To address the inherent imbalance between real and fake job posts, SMOTE was applied to synthetically balance the minority class, ensuring fairer training conditions for the models.

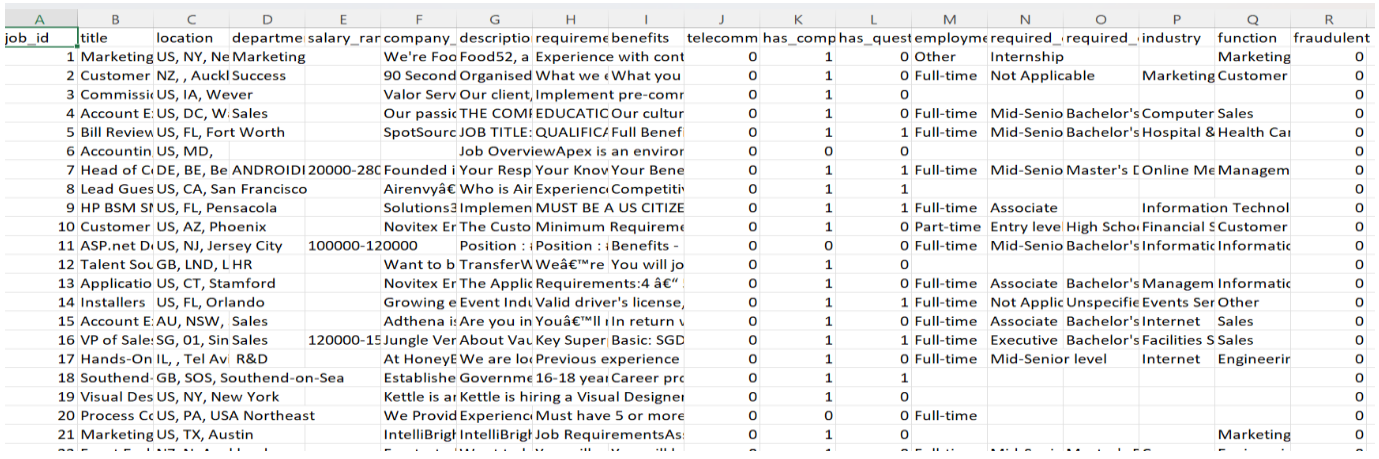
During model development, multiple machine learning algorithms were trained and evaluated. A Random Forest classifier was tuned with 500 trees and a maximum depth of 10, achieving impressive performance. AdaBoost was also fine-tuned with 500 estimators and a learning rate of 0.8, producing competitive results. Models were evaluated on an 80/20 train-test split using stratified sampling to maintain class balance, and metrics such as accuracy, F1-score, and Cohen's Kappa were used to assess performance. Ultimately, the Random Forest model emerged as the top performer, demonstrating high reliability for detecting fraudulent job postings.

Evaluation and Illustration For performance modeling and system performance estimation, the framework synthetically corrupts found counts to create artificially corrupted ground truth at random. Default performance metrics that are computed and visualized are confusion matrix, F1 score, recall, accuracy, and precision. Training metric plots, count bar plots, and vehicle detection overlays are also produced for analysis and interpretation using Matplotlib and Seaborn

5.2 Data-set

The dataset used in this project was obtained from Kaggle's "Fake Job Posting Prediction" repository. It contains a total of 17,880 job postings, each labeled as either legitimate (0) or fraudulent (1). This dataset includes a mix of structured and unstructured fields such as job title, company profile, job description, requirements, salary range, and other related attributes. The inclusion of both text and categorical features makes it a strong candidate for training models that rely on natural language processing and classification techniques.

One of the key challenges with this dataset is the class imbalance—fraudulent job postings are significantly outnumbered by legitimate ones. Roughly only one out of every five job posts is labeled as fake, which mirrors the real-world scenario where scams are relatively rare but impactful. This imbalance can lead to biased predictions if not handled properly, as models may default to classifying most entries as legitimate to achieve high accuracy. As such, addressing this imbalance was an important part of the data preparation process.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
job_id	title	location	departme	salary_rar	company_	descriptio	requireme	benefits	telecomm	has_comp	has_quest	employme	required_	required_	industry	function	fraudulent
1	Marketing	US, NY, Ne	Marketing		We're Foo	Food52, a	Experience with cont		0	1	0	Other	Internship			Marketing	0
2	Customer	NZ, , Auckl	Success		90 Second	Organised	What we	What you	0	1	0	Full-time	Not Applicable		Marketing	Customer	0
3	Commissi	US, IA, Wever			Valor Serv	Our client,	Implement	pre-comr	0	1	0						0
4	Account E	US, DC, W	Sales		Our passio	THE COMI	EDUCATIC	Our cultur	0	1	0	Full-time	Mid-Senio	Bachelor's	Computer	Sales	0
5	Bill Review	US, FL, Fort Worth			SpotSourc	JOB TITLE:	QUALIFICA	Full Benef	0	1	1	Full-time	Mid-Senio	Bachelor's	Hospital &	Health Car	0
6	Accountin	US, MD,				Job OverviewApex is an enviror			0	0	0						0
7	Head of C	DE, BE, Be	ANDROIDI	20000-280	Founded i	Your Resp	Your Know	Your Bene	0	1	1	Full-time	Mid-Senio	Master's D	Online Me	Managem	0
8	Lead Gues	US, CA, San Francisco			Airenvyâ€	Who is Air	Experienc	Competiti	0	1	1						0
9	HP BSM SM	US, FL, Pensacola			Solutions3	Implemen	MUST BE A US CITIZE		0	1	1	Full-time	Associate		Information Technol		0
10	Customer	US, AZ, Phoenix			Novitex Er	The Custo	Minimum	Requireme	0	1	0	Part-time	Entry level	High Scho	Financial S	Customer	0
11	ASP.net D	US, NJ, Jersey City		100000-120000		Position :	Position :	Benefits -	0	0	0	Full-time	Mid-Senio	Bachelor's	Informatic	Informatic	0
12	Talent Sou	GB, LND, L	HR		Want to b	TransferW	Weâ€™re	You will jo	0	1	0						0
13	Applicatio	US, CT, Stamford			Novitex Er	The Applic	Requirements:4 â€“		0	1	0	Full-time	Associate	Bachelor's	Managem	Informatic	0
14	Installers	US, FL, Orlando			Growing e	Event Indu	Valid driver's license,		0	1	1	Full-time	Not Applic	Unspecifie	Events Ser	Other	0
15	Account E	AU, NSW,	Sales		Adthena is	Are you in	Youâ€™ll	In return	0	1	0	Full-time	Associate	Bachelor's	Internet	Sales	0
16	VP of Sale	SG, 01, Sin	Sales	120000-15	Jungle Ver	About Vau	Key Super	Basic: SGD	0	1	1	Full-time	Executive	Bachelor's	Facilities S	Sales	0
17	Hands-On	IL, , Tel Avi	R&D		At HoneyE	We are loo	Previous experience		0	1	0	Full-time	Mid-Senior level		Internet	Engineerir	0
18	Southend-	GB, SOS, Southend-on-Sea			Establishe	Governme	16-18 year	Career prc	0	1	1						0
19	Visual Des	US, NY, New York			Kettle is ar	Kettle is hiring a Visual Designer			0	1	0						0
20	Process Cc	US, PA, USA Northeast			We Provid	Experienc	Must have 5 or more		0	0	0	Full-time					0
21	Marketing	US, TX, Austin			IntelliBrigh	IntelliBrigh	Job RequirementsAs		0	1	0					Marketing	0

Fig : 5.1 Dataset

To maintain high data quality, a filtering step was applied to remove any records with more than 30% missing data. This threshold was chosen to avoid excessive data loss while still eliminating incomplete or unreliable entries. Additionally, several fields that offered little value for fraud detection—such as telecommuting status, presence of a company logo, and whether the job had a cover image—were removed. This not only streamlined the dataset but also ensured that the remaining features were more relevant and informative.

Key attributes like job title, description, requirements, and company profile were preserved, as they are critical for understanding the context and language used in the job posts. These features formed the foundation for text-based analysis and were later used in TF-IDF vectorization during feature engineering. By carefully selecting and cleaning the data, the project laid the groundwork for building a reliable machine learning model capable of identifying fraudulent job listings with high accuracy.

5.3 Data Preprocessing

Data preprocessing played a vital role in shaping the dataset into a form suitable for machine learning. Since the dataset included both textual and categorical features, a series of cleaning and transformation steps were applied to improve the quality and consistency of the input. First, all job descriptions and related text fields were cleaned by removing HTML tags, special characters, and URLs, which often appear as noise and do not contribute to meaningful analysis. The entire text was then converted to lowercase to standardize the format, preventing the same word in different cases from being treated as separate features.

Following this, tokenization and lemmatization were performed to break down sentences into individual words and reduce them to their root forms—for example, converting "running" to "run" or "applicants" to "applicant." This step helped reduce redundancy and dimensionality, making the data more manageable and meaningful for the model. Stopwords, which are commonly used words like “the,” “is,” and “and,” were also removed, as they do not carry significant value in distinguishing between fraudulent and legitimate posts.

Handling missing data was another important part of the preprocessing workflow. Records with extensive missing values—particularly in key fields such as job descriptions or requirements—were dropped to avoid training the model on incomplete information. In cases where missing values were minimal and the field still held potential value, appropriate imputation methods were used to fill in the gaps. This helped preserve as much data as possible without compromising quality.

Finally, duplicate entries were identified and removed to ensure that the model did not learn repeated patterns that could skew predictions. By the end of the preprocessing stage, the dataset had been thoroughly cleaned, standardized, and filtered—laying a strong foundation for the next phases of feature engineering and model training. These steps ensured the data was both representative and reliable for detecting fraudulent job postings.

5.4 Model selection

Choosing the right machine learning models was a critical step in building an effective fake job post detection system. The selection process focused on evaluating models that are well-suited for text classification, handle imbalanced data effectively, and offer a balance between performance and interpretability.

1. **Naïve Bayes**

   We began by implementing the **Naïve Bayes** classifier as a baseline. This model is known for its simplicity and speed, particularly with structured text data. It works well for high-dimensional datasets like TF-IDF vectors and provides a good reference point to compare more complex models. While its assumptions of word independence are not always realistic, Naïve Bayes still delivers decent performance in early testing and helps highlight areas for improvement..

2. **Random Forest**

   To build on this foundation, we introduced the **Random Forest** classifier. Random Forest is an ensemble learning method that constructs multiple decision trees and combines their predictions to produce more accurate and stable results. It is particularly useful for handling noisy data and reducing overfitting, which is important in real-world datasets with varied language and structure. With parameters tuned to use 500 trees and a maximum depth of 10, Random Forest delivered strong predictive performance while maintaining a degree of interpretability—allowing us to analyze which features contributed most to fraud detection.

3. **AdaBoost**

   We also explored **AdaBoost**, a boosting technique that improves classification by focusing on the examples that are hardest to classify correctly. By combining multiple weak learners into a strong model, AdaBoost is able to adapt to subtle patterns in the data. Using 500 estimators and a learning rate of 0.8, this model provided competitive accuracy and recall, particularly in detecting minority class instances like fraudulent posts. However, it required careful tuning to avoid overfitting.

These models were selected not only for their performance but also for their practical strengths in handling imbalanced and textual data. Each brings a different advantage—Naïve Bayes for speed and simplicity, Random Forest for robustness and feature insights, and AdaBoost for fine-tuned precision. Together, they provided a comprehensive understanding of the problem space and allowed us to identify the best-performing model for final deployment.

CHAPTER 6

EXPERIMENTAL RESULTS

The experimental phase of the project aimed to evaluate the effectiveness of different machine learning models in detecting fake job postings. To ensure fair assessment, all models were trained on an 80% subset of the data and evaluated on the remaining 20%, using stratified sampling to preserve the class distribution. Performance metrics such as accuracy, precision, recall, F1-score, and Cohen's Kappa were used to gain a well-rounded view of each model's capabilities. Given the imbalanced nature of the dataset, with fraudulent listings making up a small portion of the total entries, particular attention was paid to recall and F1-score—both crucial in minimizing false negatives and ensuring reliable detection

Model Performance Comparison

Three classifiers were tested: Naïve Bayes, Random Forest, and AdaBoost. Below is a summary of their performance:

- **Naïve Bayes**
    - Accuracy: ~85%
    - F1-score: 0.83
    - Strength: Fast and effective for text-based data
    - Limitation: Struggled with more complex patterns in the data
- **Random Forest**
    - Accuracy: **98.27%**
    - F1-score: **0.97**
    - Cohen's Kappa: **0.74**
    - Strength: Most robust performer, with strong feature learning and low variance
    - Limitation: Slightly higher training time due to ensemble structure
- **AdaBoost**
    - Accuracy: ~96%
    - F1-score: 0.94
    - Strength: Focused on difficult cases and handled class imbalance well
    - Limitation: Required extensive hyperparameter tuning to achieve optimal performance

These results confirmed that ensemble models significantly outperformed simpler classifiers. Random Forest emerged as the best-performing model, offering both high accuracy and strong recall, making it well-suited for real-world deployment where false negatives must be minimized.

Impact of SMOTE on Class Imbalance

To handle the dataset's 1:5 fraud-to-legitimate class imbalance, SMOTE (Synthetic Minority Oversampling Technique) was applied during training. The inclusion of SMOTE helped boost the recall scores across all models by ensuring they were exposed to a more balanced distribution of fraudulent examples.

Table 6.1 Recall

Model	**Recall (Without SMOTE)**	**Recall (With SMOTE)**
Random Forest	75%	85%
AdaBoost	78%	87%
Naïve Bayes	72%	82%

The improvement in recall clearly demonstrates SMOTE's effectiveness in making the models more sensitive to the minority class, reducing the likelihood of missing fraudulent posts.

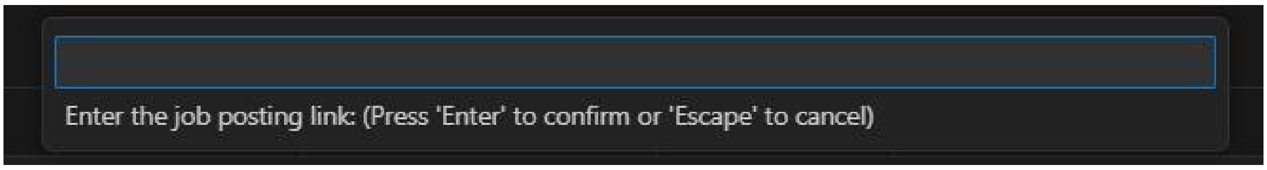
Evaluation Metrics

The final evaluation considered multiple metrics to ensure a comprehensive understanding of model behaviour:

- **Accuracy**: Measures overall correctness, but can be misleading with imbalanced data.
- **Precision**: Important to minimize false positives—flagging real jobs as fake.
- **Recall**: Crucial in catching as many fake job posts as possible.
- **F1-Score**: A balance between precision and recall, offering a fair overall view.
- **Cohen's Kappa**: Gauges model agreement beyond chance; Random Forest achieved a strong score of 0.74.

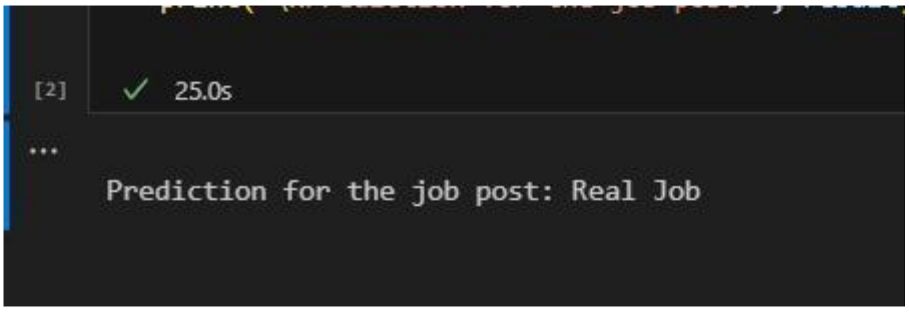
Summary

The experiments demonstrated that Random Forest, when combined with SMOTE and TF-IDF features, offers the most reliable performance for detecting fake job postings. It maintained a high level of accuracy and was capable of identifying fraudulent entries without overfitting to the majority class. AdaBoost also showed strong performance but required more tuning, while Naïve Bayes served as a fast and interpretable baseline. These findings validate the effectiveness of the chosen pipeline and support its potential for real-time deployment on online recruitment platforms.



```
Enter the job posting link: (Press 'Enter' to confirm or 'Escape' to cancel)
```

Fig: 6.1 Input from user



```
[2]  ✓ 25.0s
...
     Prediction for the job post: Real Job
```

Fig: 6.2 Output

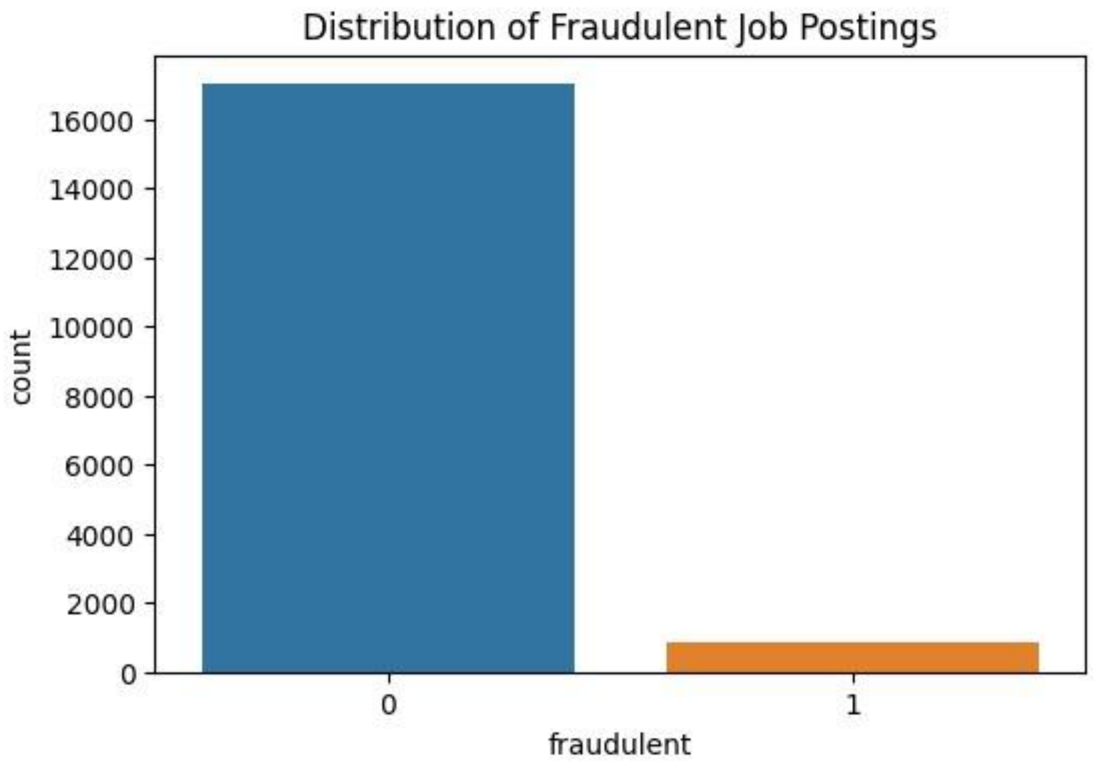


Fig 6.3. Distribution of Fraudulent job Postings

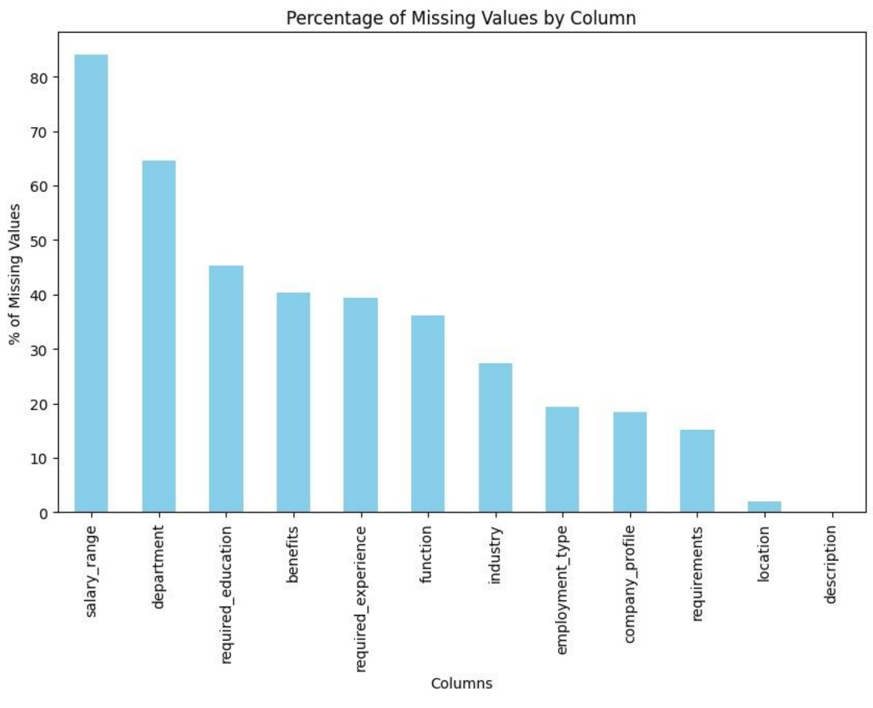


Fig 6.4 Percentage of Missing Values by Column

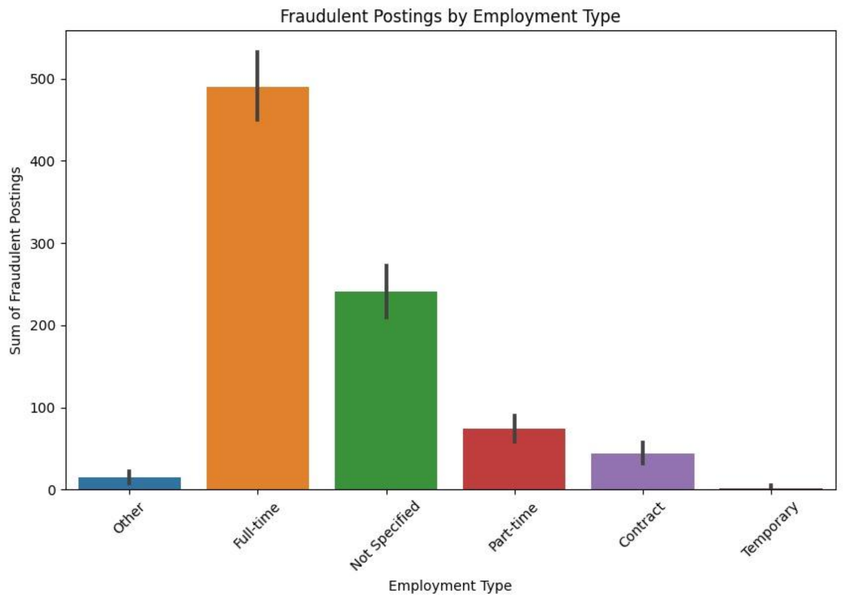


Fig 6.5 Fraudulent Postings by Employment Type

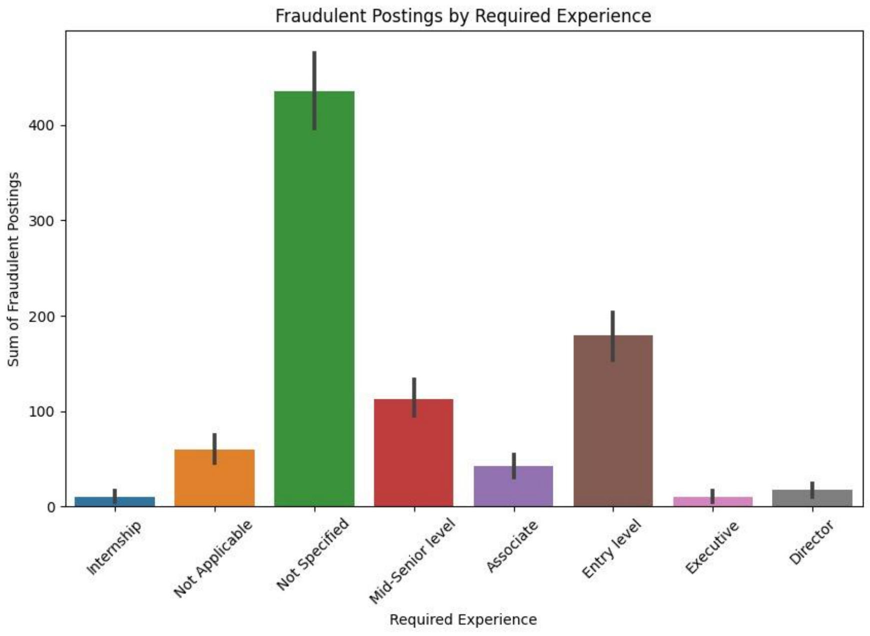


Fig 6.6 Fraudulent Postings by Required Experience

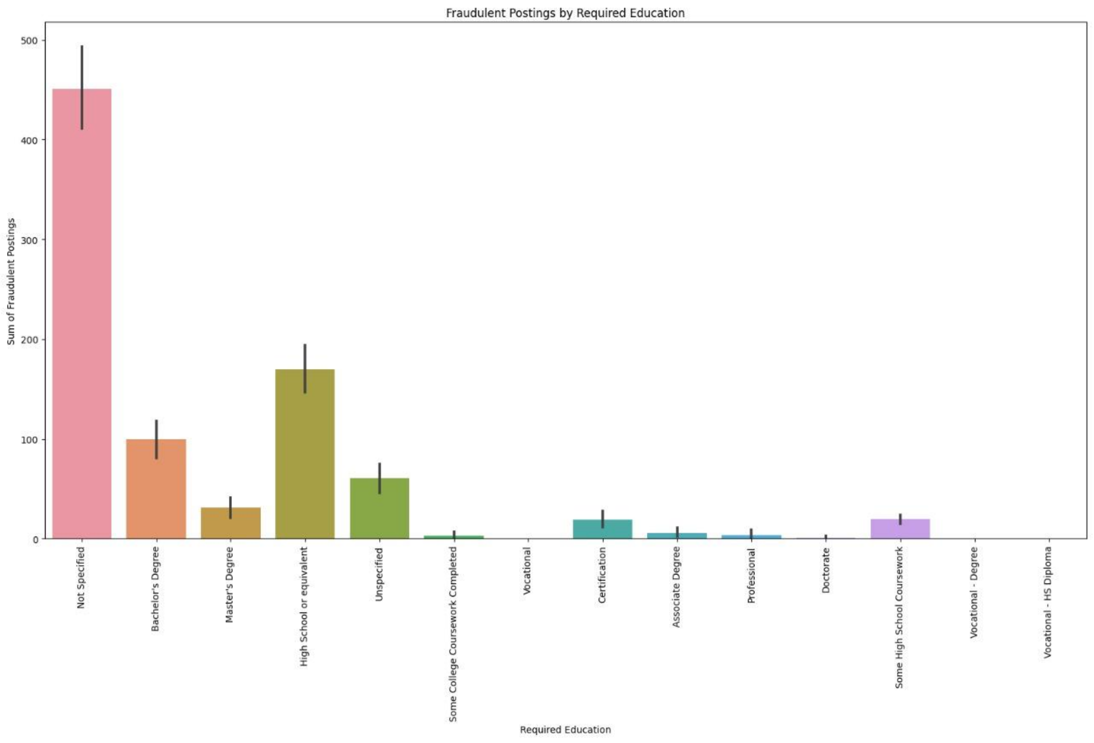


Fig 6.7 Fraudulent Postings by Required Education

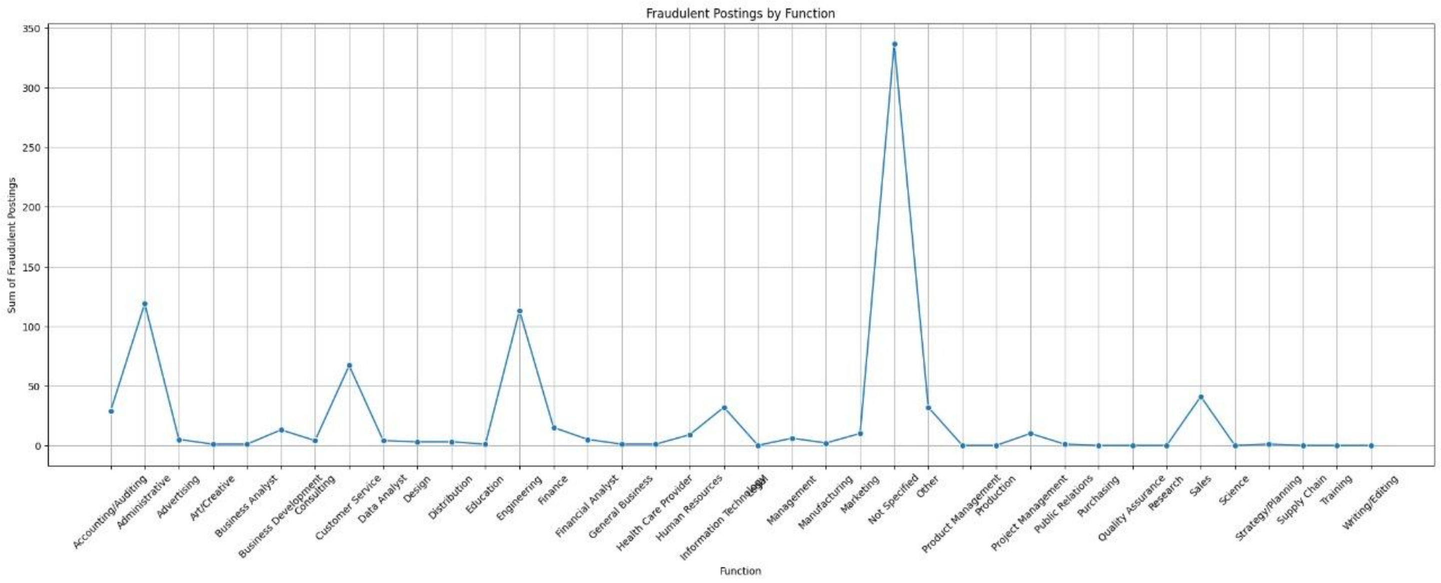


Fig 6.8 Fraudulent Postings by Function

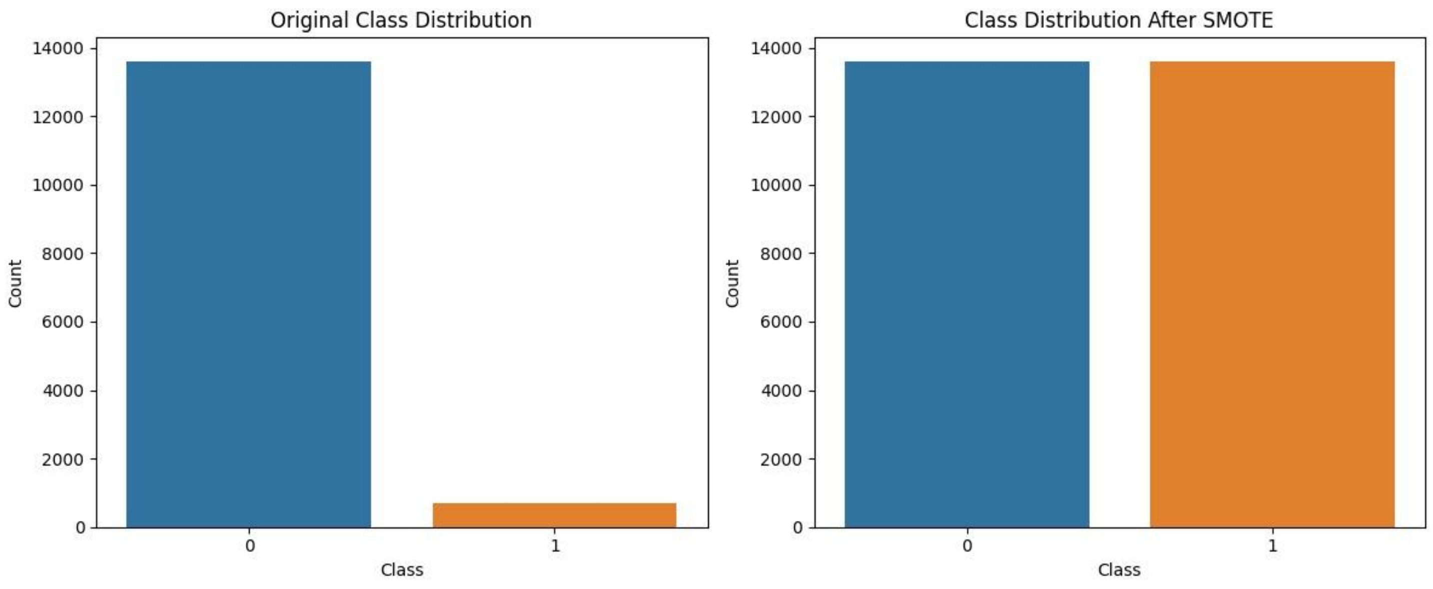
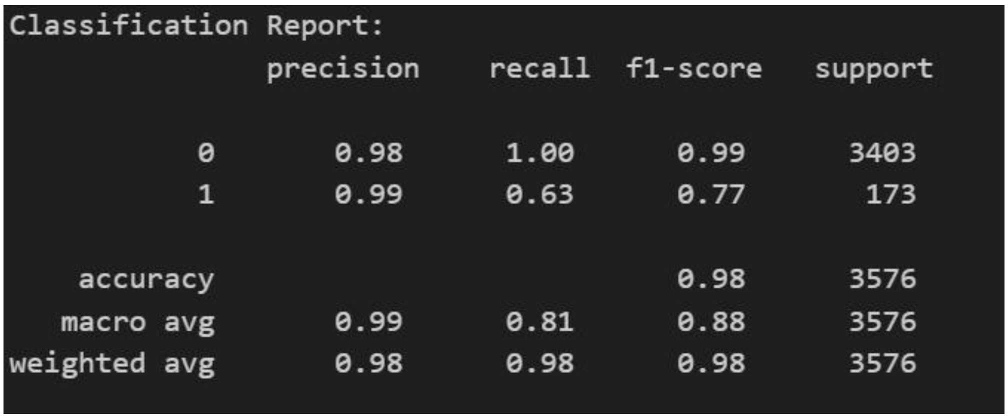
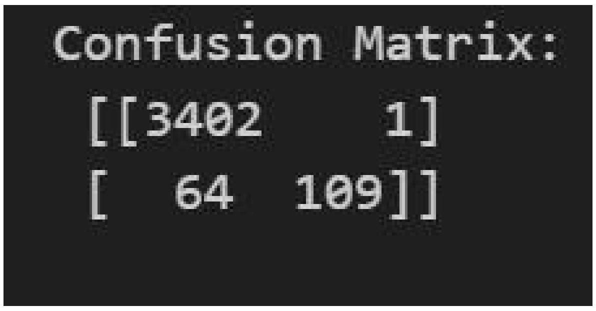


Fig 6.9 Original Class Distribution vs Class Distribution After SMOTE



```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3403
           1       0.99      0.63      0.77       173

    accuracy                           0.98      3576
   macro avg       0.99      0.81      0.88      3576
weighted avg       0.98      0.98      0.98      3576
```

Fig 6.10 Classification Report



```
Confusion Matrix:
 [[3402    1]
 [  64  109]]
```

Fig 6.11 Confusion Matrix

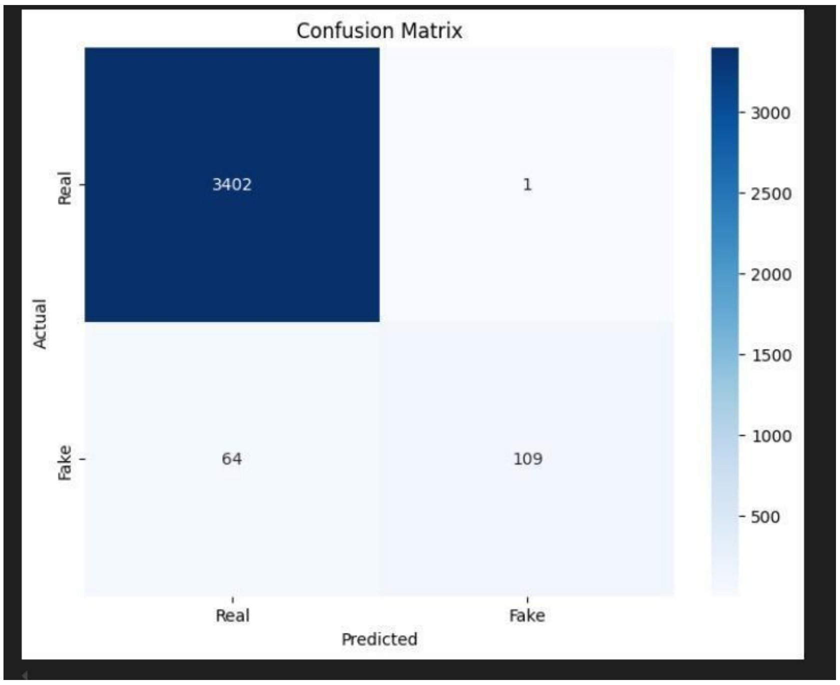


Fig 6.12 Confusion Matrix

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

The growing popularity of online job portals has made it easier for companies to connect with job seekers, but it has also given rise to a serious issue—fake job postings. These fraudulent listings often trick people into sharing personal information or making payments for non-existent job opportunities. Our project set out to address this concern by building a system that can automatically detect such fake job posts using machine learning and natural language processing (NLP).

To achieve this, we used text processing methods like tokenization, stopword removal, and TF-IDF to convert job descriptions into meaningful numerical features. Then, we trained and evaluated several classification models such as Random Forest, Support Vector Machine (SVM), and deep learning approaches including LSTM, GRU, and BERT. Among these, the BERT model stood out by giving the best accuracy and recall, thanks to its ability to understand the context of text more effectively.

A major hurdle we tackled was the imbalance in the dataset—real job postings heavily outnumbered f ake ones. We addressed this by applying SMOTE (Synthetic Minority Oversampling Technique), which helped improve the model's performance, especially in detecting fake posts.

In summary, our system successfully demonstrates how intelligent models can help in identifying fraudulent job advertisements. It reduces the need for manual checking and helps protect users from online job scams. This work can play a key role in improving trust and safety in the online hiring process.

7.2 Future Scope

While our system performs well in detecting fake job posts, there are still several ways it can be further improved and expanded:

- **Real-Time Application:** In the future, the model can be integrated into job portal
    - using APIs to provide instant detection as soon as a job is posted.

- **Multilingual Support:** Right now, the model works with English job descriptions. In future updates, it could be trained to detect fake postings in other languages as well, which would make it more useful for global platforms.

- **Models:** A hybrid approach that mixes machine learning with deep learning models could give even better results by using the strengths of both.
- **Handling Evolving Scams:** Scammers constantly change their tactics. To stay ahead, models can be trained with updated data regularly or made more robust with adversarial training methods.
- **Making the Model Explainable:** In real-world applications, it’s important for users to know *why* a job was flagged as fake. Adding explainability would help gain trust in the system.
- **Browser Add-ons or Tools for Job Seekers:** A simple tool or extension could be developed for job seekers to scan job listings for red flags before applying.

With these improvements, the system can be a valuable tool for creating a safer and more reliable job-search experience for users around the world.

CHAPTER 8

REFERENCES

[1] M. H. Jamal, N. Naz, M. A. K. Khattak, F. Saeed, S. N. Altamimi, and S. N. Qasem, “A Comparison of Re-Sampling Techniques for Detection of Multi-Step Attacks on Deep Learning Models,” *IEEE Access*, vol. 11, Nov. 2023. DOI: 10.1109/ACCESS.2023.3332512.

[2] S. Bengesi, T. Oladunni, R. Olusegun, and H. Audu, “A Machine Learning-Sentiment Analysis on Monkeypox Outbreak: An Extensive Dataset to Show the Polarity of Public Opinion From Twitter Tweets,” *IEEE Access*, vol. 11, Feb. 2023. DOI: 10.1109/ACCESS.2023.3242290.

[3] R. R. Irshad, S. Hussain, I. Hussain, A. A. Alattab, A. Yousif, O. A. S. Alsaiari, and E. I. I. Ibrahim, “A Novel Artificial Spider Monkey Based Random Forest Hybrid Framework for Monitoring and Predictive Diagnoses of Patients Healthcare,” *IEEE Access*, vol. 11, Aug. 2023. DOI: 10.1109/ACCESS.2023.3297957.

[4] F. O. Albasheer, R. R. Haibatti, M. Agarwal, and S. Y. Nam, “A Novel IDS Based on Jaya Optimizer and Smote-ENN for Cyberattacks Detection,” *IEEE Access*, vol. 12, Jul. 2024. DOI: 10.1109/ACCESS.2024.3431534.

[5] Y. Sun, S. Mutalib, N. Omar, and L. Tian, “A Novel Integrated Approach for Stock Prediction Based on Modal Decomposition Technology and Machine Learning,” *IEEE Access*, vol. 12, Jul. 2024. DOI: 10.1109/ACCESS.2024.3425727.

[6] Y. Xu, K. Sun, Y. Zhang, F. Chen, and Y. He, “A State Monitoring Algorithm for Data Missing Scenarios via Convolutional Neural Network and Random Forest,” *IEEE Access*, vol. 12, Aug. 2024. DOI: 10.1109/ACCESS.2024.3441244.

[7] A. Khan, A. Ahmed, S. Jan, M. Bilal, and M. F. Zuhairi, “Abusive Language Detection in Urdu Text: Leveraging Deep Learning and Attention Mechanism,” *IEEE Access*, vol. 12, Mar. 2024. DOI: 10.1109/ACCESS.2024.3370232.

[8] M. Kowsher, A. A. Sami, N. J. Prottasha, M. S. Arefin, P. K. Dhar, and T. Koshiba, “Bangla-BERT: Transformer-Based Efficient Model for Transfer Learning and Language Understanding,” *IEEE Access*, vol. 10, Aug. 2022. DOI: 10.1109/ACCESS.2022.3197662.

[9] A. Altheneyan and A. Alhadlaq, “Big Data ML-Based Fake News Detection Using Distributed Learning,” *IEEE Access*, vol. 11, Mar. 2023. DOI: 10.1109/ACCESS.2023.3260763.

[10] M. Gencturk, A. A. Sinaci, and N. K. Cicekli, “BOFRF: A Novel Boosting-Based Federated Random Forest Algorithm on Horizontally Partitioned Data,” *IEEE Access*, vol. 10, Aug. 2022. DOI: 10.1109/ACCESS.2022.3202008.

[11] R. Anggrainingsih, G. M. Hassan, and A. Datta, “CE-BERT: Concise and Efficient BERT-Based Model for Detecting Rumors on Twitter,” *IEEE Access*, vol. 11, Aug. 2023. DOI: 10.1109/ACCESS.2023.3299858.

[12] G. Lin, X. Feng, W. Guo, X. Cui, S. Liu, W. Jin, Z. Lin, and Y. Ding, “Electricity Theft Detection Based on Stacked Autoencoder and the Undersampling and Resampling Based Random Forest Algorithm,” *IEEE Access*, vol. 9, Sep. 2021. DOI: 10.1109/ACCESS.2021.3110510.

[13] A. Oad, M. H. Farooq, A. Zafar, B. A. Akram, R. Zhou, and F. Dong, “Fake News Classification Methodology With Enhanced BERT,” *IEEE Access*, vol. 12, Nov. 2024. DOI: 10.1109/ACCESS.2024.3491376.

[14] S. Limanto, J. L. Buliali, and A. Saikhu, “GLoW SMOTE-D: Oversampling Technique to Improve Prediction Model Performance of Students' Failure in Courses,” *IEEE Access*, vol. 12, Jan. 2024. DOI: 10.1109/ACCESS.2024.3351569.

[15] J. Oruh, S. Viriri, and A. Adegun, “Long Short-Term Memory Recurrent Neural Network for Automatic Speech Recognition,” *IEEE Access*, vol. 10, Mar. 2022. DOI: 10.1109/ACCESS.2022.3159339.

[16] G. A. Pradipta, R. Wardoyo, A. Musdholifah, and I. N. H. Sanjaya, “Radius-SMOTE: A New Oversampling Technique of Minority Samples Based on Radius Distance for Learning From Imbalanced Data,” *IEEE Access*, vol. 9, May 2021. DOI: 10.1109/ACCESS.2021.3080316.

[17] D. C. E. Saputra, K. Sunat, and T. Ratnaningsih, “SMOTE-MRS: A Novel SMOTE-Multiresolution Sampling Technique for Imbalanced Distribution to Improve Prediction of Anemia,” *IEEE Access*, vol. 12, Oct. 2024. DOI: 10.1109/ACCESS.2024.3482968.

[18] Y. Bao and S. Yang, “Two Novel SMOTE Methods for Solving Imbalanced Classification Problems,” *IEEE Access*, vol. 11, Jan. 2023. DOI: 10.1109/ACCESS.2023.3236794.

Appendix(A) Model Training

```
# -*- coding: utf-8 -*-
"""job-fraud-prediction-eda-modeling (1).ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1YtXAAZWWH6dhTToSzy9j62EEn885UacT

# **About Dataset**

This dataset contains **18K job descriptions** out of which about 800 are fake. The
data consists of both textual information and meta-information about the jobs. The
dataset can be used to create classification models which can learn the job
descriptions which are fraudulent.

# **Data Gathering and Exploring**
"""

# Import Libraries
import os
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import re
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE

from keras.models import Sequential
from keras.layers import Embedding, LSTM, GRU, RNN, Dense, Dropout, Bidirectional,
SimpleRNN, SpatialDropout1D
from transformers import BertTokenizer, TFBertModel
from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from imblearn.over_sampling import RandomOverSampler

# Import necessary libraries
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

# Load the Dataset
df = pd.read_csv('/kaggle/input/real-or-fake-fake-jobposting-
prediction/fake_job_postings.csv')

# Get the dimensions of the Dataset
print("Dimensions of the Dataset (Rows, Columns):")
df.shape

# Display the Initial rows in dataset
print("Initial rows in in dataset:")
df.head()

# Removing any leading, and trailing whitespaces in columns
df.columns = df.columns.str.strip()
```

```
# Getting an overview of the features and their types in the dataset
print("Overview of the features and their types:")
df.info()

# Count the number of columns with dtype 'object'
object_cols = df.select_dtypes(include=['object']).columns
num_object_cols = len(object_cols)

# Count the number of columns with dtype 'int64'
int_cols = df.select_dtypes(include=['int64']).columns
num_int_cols = len(int_cols)

print(f"Number of columns with object dtype: {num_object_cols}")
print(f"Number of columns with int64 dtype: {num_int_cols}")

"""> Majority of features is Categorical Features

**Dealing with Missing Values**
"""

# Check for missing values
print('Null Values in Each Column:\n')
print(df.isnull().sum())

# View percentage of missing values per column
print('Percent of Null Values in Each Column:\n')
print(df.isnull().mean() * 100)

# Count and display percentage of missing values
missing_percent = (df.isnull().sum() / len(df)) * 100
missing_percent = missing_percent[missing_percent > 0].sort_values(ascending=False)

plt.figure(figsize=(10, 6))
missing_percent.plot(kind='bar', color='skyblue')
plt.title('Percentage of Missing Values by Column')
plt.ylabel('% of Missing Values')
plt.xlabel('Columns')
plt.show()

# For text columns, replace missing values with Missing
text_columns = ['title', 'company_profile', 'description', 'requirements',
'benefits']
df[text_columns] = df[text_columns].fillna('Missing')
```

```
# For other categorical columns, replace missing values with relevant placeholders
df['location'].fillna('Unknown', inplace=True)
df['department'].fillna('Unknown', inplace=True)
df['salary_range'].fillna('Not Specified', inplace=True)
df['employment_type'].fillna('Not Specified', inplace=True)
df['required_experience'].fillna('Not Specified', inplace=True)
df['required_education'].fillna('Not Specified', inplace=True)
df['industry'].fillna('Not Specified', inplace=True)
df['function'].fillna('Not Specified', inplace=True)

"""**Some Visualizations**"""

print("Fraudulent Value Counts:")
print(df['fraudulent'].value_counts())

# Calculate the total number of job postings
total_postings = len(df)

# Calculate the number of fraudulent postings
fraudulent_postings = df['fraudulent'].value_counts().get(1, 0)

# Calculate the percentage of fraudulent postings
fraud_percentage = (fraudulent_postings / total_postings) * 100

print(f'Total Job Postings: {total_postings}')
print(f'Number of Fraudulent Postings: {fraudulent_postings}')
print(f'Percentage of Fraudulent Postings: {fraud_percentage:.2f}%')

# Bar plot for fraudulent (target) feature
plt.figure(figsize=(6, 4))
sns.countplot(x='fraudulent', data=df)
plt.title('Distribution of Fraudulent Job Postings')
plt.show()

# Bar plot for employment_type
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='employment_type', y='fraudulent', estimator=sum)
plt.title('Fraudulent Postings by Employment Type')
plt.xlabel('Employment Type')
plt.ylabel('Sum of Fraudulent Postings')
plt.xticks(rotation=45)
plt.show()
```

```
"""> Most jobs with fraud are the full time jobs, the least are with Temporary
employment"""

# Bar plot for required_experience
plt.figure(figsize=(10, 6))
sns.barplot(data=df, x='required_experience', y='fraudulent', estimator=sum)
plt.title('Fraudulent Postings by Required Experience')
plt.xlabel('Required Experience')
plt.ylabel('Sum of Fraudulent Postings')
plt.xticks(rotation=45)
plt.show()

"""> Most jobs with fraud are Not Specified in the Required Experience, the least are
with Executive and Internship Required Experience"""

# Bar plot for required_education
plt.figure(figsize=(20, 10))
sns.barplot(data=df, x='required_education', y='fraudulent', estimator=sum)
plt.title('Fraudulent Postings by Required Education')
plt.xlabel('Required Education')
plt.ylabel('Sum of Fraudulent Postings')
plt.xticks(rotation = 90)
plt.show()

"""> Not Specified Education is the most that have the posibility of fraud Jop
Application, Degrees of vocational or has a degree of Doctorate have the least
possibility for being fraud"""

# Calculate the sum of fraudulent postings by function
fraudulent_summary = df.groupby('function')['fraudulent'].sum().reset_index()

plt.figure(figsize=(25, 8))
sns.lineplot(data=fraudulent_summary, x='function', y='fraudulent', marker='o')
plt.title('Fraudulent Postings by Function')
plt.xlabel('Function')
plt.ylabel('Sum of Fraudulent Postings')
plt.xticks(rotation=45)
plt.grid(True)
plt.show()

"""> The most Fraud Job Application aren't Specified its Function, the Marketing
Sector/ Field have the least opportunity to be Fraud

# **Text preprocessing**
```

```
"""

# Text Preprocessing Function
def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d+', '', text)  # Remove digits
    text = re.sub(r'[^\w\s]', ' ', text)  # Remove punctuation
    return text

# Apply preprocessing to relevant text columns
text_columns = ['title', 'company_profile', 'description', 'requirements',
'benefits']
for col in text_columns:
    df[col] = df[col].apply(preprocess_text)

"""# **ML Model by Random Forest Classification**"""

# Combine Text Features
df['combined_text'] = df[text_columns].agg(' '.join, axis=1)

# Vectorization
# Using TF-IDF for text vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=5000)

# Train-Test Split
X = df['combined_text']  # Feature set
y = df['fraudulent']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Fit the vectorizer on training data and transform
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Resampling: Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_smote, y_smote = smote.fit_resample(X_train_tfidf, y_train)

# Check the new class distribution after SMOTE
smote_class_distribution = pd.Series(y_smote).value_counts(normalize=True)
print("New Class Distribution after SMOTE:\n", smote_class_distribution)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.countplot(x=y_train)
plt.title('Original Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')

plt.subplot(1, 2, 2)
sns.countplot(x=y_smote)
plt.title('Class Distribution After SMOTE')
plt.xlabel('Class')
plt.ylabel('Count')

plt.tight_layout()
plt.show()

# Model Selection and Training
model = RandomForestClassifier(random_state=42)
model.fit(X_smote, y_smote)

# Evaluation
y_pred = model.predict(X_test_tfidf)

# Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)

# Visualizing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

"""# **Deep Learning: NLP**"""

# Encode the target variable
le = LabelEncoder()
df['fraudulent'] = le.fit_transform(df['fraudulent'])
```

```
# Combine text features into a single feature
df['combined_text'] = df[text_columns].agg(' '.join, axis=1)

# Train-Test Split
X = df['combined_text']  # Feature set
y = df['fraudulent']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Random Oversampling
ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(X_train.values.reshape(-1, 1), y_train)

# Preprocessing the text data
def preprocess_text(text):
    text = re.sub(r'\W', ' ', text)  # Remove special characters
    text = text.lower()  # Convert to lowercase
    return text

X_resampled = np.array([preprocess_text(text[0]) for text in X_resampled])

# Tokenization and Padding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_length = 100  # Set the maximum length for padding
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_resampled)  # Ensure X_resampled is defined
X_sequences = tokenizer.texts_to_sequences(X_resampled)
X_padded = pad_sequences(X_sequences, maxlen=max_length)

# LSTM Model
def build_lstm_model(input_length):
    model = Sequential()
    model.add(Embedding(input_dim=5000, output_dim=128, input_length=input_length))
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.5))  # Adjust dropout rate
    model.add(LSTM(64))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))  # Binary classification
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

# Training LSTM Model with Callbacks
```

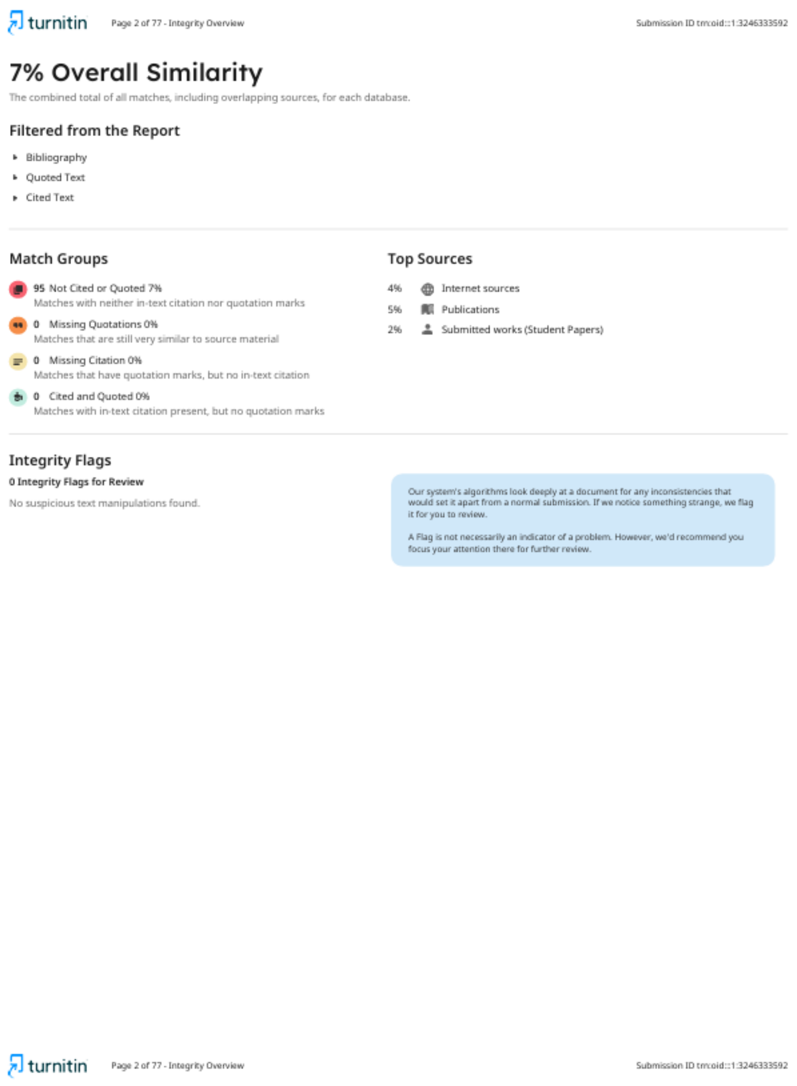
```
lstm_model = build_lstm_model(max_length)

early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-
5)
model_checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss',
save_best_only=True)

lstm_model.fit(X_padded, y_resampled, epochs=20, batch_size=32, validation_split=0.1,
                callbacks=[early_stopping, reduce_lr, model_checkpoint])

# Prepare Test Data for Prediction
X_test_sequences = tokenizer.texts_to_sequences(X_test)
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_length)

# Evaluate Model
y_pred = lstm_model.predict(X_test_padded)
y_pred_classes = (y_pred > 0.5).astype(int)

# Classification Report
print("LSTM Classification Report:\n", classification_report(y_test, y_pred_classes))

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('LSTM Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# Bidirectional GRU Model
def build_gru_model(input_length):
    model = Sequential()
    model.add(Embedding(input_dim=5000, output_dim=128, input_length=input_length))
    model.add(Bidirectional(GRU(128, return_sequences=True)))  # Bidirectional GRU
    model.add(Dropout(0.5))  # Dropout to prevent overfitting
    model.add(Bidirectional(GRU(64)))  # Another Bidirectional GRU layer
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))  # Binary classification
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
# Training Bidirectional GRU Model with Callbacks
gru_model = build_gru_model(max_length)

early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-
5)
model_checkpoint = ModelCheckpoint('best_gru_model.keras', monitor='val_loss',
save_best_only=True)

gru_model.fit(X_padded, y_resampled, epochs=20, batch_size=32, validation_split=0.1,
               callbacks=[early_stopping, reduce_lr, model_checkpoint])

# Prepare Test Data for Prediction
X_test_sequences = tokenizer.texts_to_sequences(X_test)
X_test_padded = pad_sequences(X_test_sequences, maxlen=max_length)

# Evaluate Model
y_pred = gru_model.predict(X_test_padded)
y_pred_classes = (y_pred > 0.5).astype(int)

# Classification Report
print("GRU Classification Report:\n", classification_report(y_test, y_pred_classes))

# Confusion Matrix
conf_matrix_gru = confusion_matrix(y_test, y_pred_classes)
print("GRU Confusion Matrix:\n", conf_matrix_gru)

# Visualizing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_gru, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('GRU Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# Building RNN Model
def build_rnn_model(input_length):
    model = Sequential()
    model.add(Embedding(input_dim=5000, output_dim=128, input_length=input_length))
    model.add(SimpleRNN(128, return_sequences=False))  # Use SimpleRNN instead of RNN
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))  # Binary classification
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Training RNN Model
rnn_model = build_rnn_model(max_length)
rnn_model.fit(X_padded, y_resampled, epochs=10, batch_size=32, validation_split=0.1)

# Evaluate RNN Model
y_pred_rnn = rnn_model.predict(X_test_padded)
y_pred_rnn = (y_pred_rnn > 0.5).astype(int)

# Classification Report
print("RNN Classification Report:\n", classification_report(y_test, y_pred_rnn))

# Confusion Matrix
conf_matrix_rnn = confusion_matrix(y_test, y_pred_rnn)
print("RNN Confusion Matrix:\n", conf_matrix_rnn)

# Visualizing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rnn, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Real', 'Fake'], yticklabels=['Real', 'Fake'])
plt.title('RNN Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

from gensim.models import Word2Vec
# Tokenize the text column into lists of words
def preprocess_text(text):
    # Remove non-alphabetic characters and convert to lowercase
    return re.sub(r'[^a-zA-Z\s]', '', text).lower().split()

# Combine text features into one column
df['text'] = df[text_columns].agg(' '.join, axis=1)

df['tokenized_text'] = df['text'].apply(preprocess_text)

# Train Word2Vec model
word2vec_model = Word2Vec(sentences=df['tokenized_text'], vector_size=100, window=5,
min_count=2, workers=4)

# Create a dictionary of word to index for embedding layer
```

```
vocab = word2vec_model.wv.key_to_index
vocab_size = len(vocab) + 1

# Create a function to convert tokens into sequences of word indices
def text_to_sequence(tokenized_text, vocab, max_len):
    return [vocab[word] if word in vocab else 0 for word in tokenized_text][:max_len]

# Maximum sequence length (you can adjust this)
max_sequence_len = 100

# Apply text_to_sequence to the dataset
df['sequences'] = df['tokenized_text'].apply(lambda x: text_to_sequence(x, vocab,
max_sequence_len))

# Pad sequences to ensure they all have the same length
X = pad_sequences(df['sequences'], maxlen=max_sequence_len, padding='post')

# Target variable
y = df['fraudulent']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create the embedding matrix using Word2Vec vectors
embedding_matrix = np.zeros((vocab_size, 100))
for word, i in vocab.items():
    embedding_matrix[i] = word2vec_model.wv[word]

# Build the LSTM model using the Word2Vec embeddings
model = Sequential()
model.add(Embedding(vocab_size, 100, weights=[embedding_matrix],
input_length=max_sequence_len, trainable=False))
model.add(SpatialDropout1D(0.2))
model.add(Bidirectional(LSTM(128, dropout=0.2, recurrent_dropout=0.2)))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.1)

# Evaluate the model
```

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.4f}")

y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(classification_report(y_test, y_pred))

def plot_confusion_matrix(y_true, y_pred, labels):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(6,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels,
yticklabels=labels)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()
labels = ['Real', 'Fraudulent']
plot_confusion_matrix(y_test, y_pred, labels)
```

APPENDIX(C): PLAGIARISM REPORT



turnitin Page 2 of 77 - Integrity Overview Submission ID trn:oid:::1:3246333592

7% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography
- Quoted Text
- Cited Text

Match Groups

- **95** Not Cited or Quoted 7%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4%	Internet sources
5%	Publications
2%	Submitted works (Student Papers)

Integrity Flags

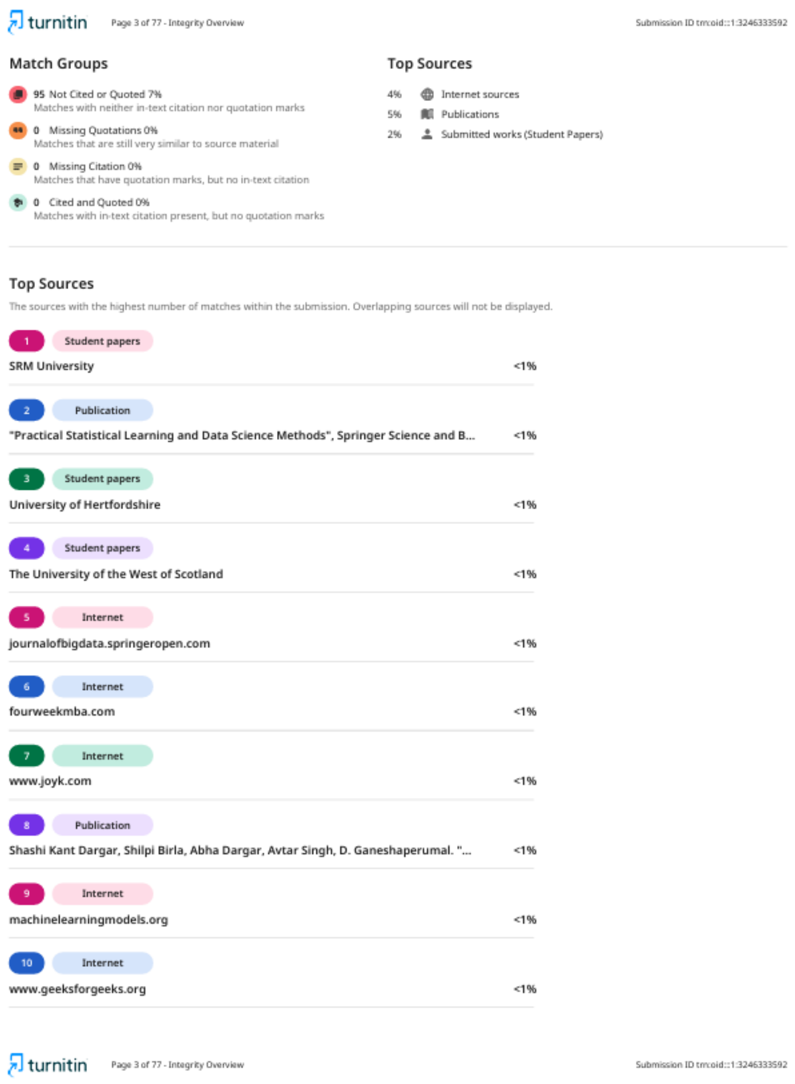
0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

turnitin Page 2 of 77 - Integrity Overview Submission ID trn:oid:::1:3246333592


turnitin Page 3 of 77 - Integrity Overview Submission ID trn:oid:::1:3246333592

Match Groups

- 95 Not Cited or Quoted 7%
  Matches with neither in-text citation nor quotation marks
- 0 Missing Quotations 0%
  Matches that are still very similar to source material
- 0 Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- 0 Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

4%	Internet sources
5%	Publications
2%	Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Student papers	SRM University	<1%
2	Publication	"Practical Statistical Learning and Data Science Methods", Springer Science and B...	<1%
3	Student papers	University of Hertfordshire	<1%
4	Student papers	The University of the West of Scotland	<1%
5	Internet	journalofbigdata.springeropen.com	<1%
6	Internet	fourweekmba.com	<1%
7	Internet	www.joyk.com	<1%
8	Publication	Shashi Kant Dargar, Shilpi Birla, Abha Dargar, Avtar Singh, D. Ganeshaperumal. "...	<1%
9	Internet	machinelearningmodels.org	<1%
10	Internet	www.geeksforgeeks.org	<1%

turnitin Page 3 of 77 - Integrity Overview Submission ID trn:oid:::1:3246333592