Fake Job Posting Detection

A

Project Report

Submitted for the partial fulfilment

of B.Tech. Degree

in

COMPUTER SCIENCE & ENGINEERING

(SELF-FINANCE)

by

Kavya Agarwal (2100520100132)

Shambhavi Singh (2100520100154)

Shivani Kumari (2100520100155)

Under the supervision of

Dr. Promila Bahadur

Department of Computer Science & Engineering

Institute of Engineering and Technology

Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh.

June, 2025

Contents

Declaration

We hereby solemnly declare that the report titled **"Fake Job Posting Detection"** is prepared and completed by us under the supervision and guidance of **Dr. Promila Bahadur**, it contains no material hitherto published by some other person which to a substantial error has been accepted by any university.

We also confirm that the report is only prepared for our academic requirement, not for any other purpose. I hereby warrant that the work we have presented does not breach any existing copyright.

Submitted by: - Date: 6th June 2025

1. Kavya Agarwal
   (2100520100132)
   Signature

2. Shambhavi Singh
   (2100520100154)
   Signature

3. Shivani Kumari
   (2100520100155)
   Signature

Certificate

This is to certify that the project report entitled **"Fake Job Posting Detection"** presented by Kavya Agarwal, Shambhavi Singh and Shivani Kumari in the partial fulfillment for the award of Bachelor of Technology in Computer Science & Engineering, is a record of work carried out by them under my supervision and guidance at the Department of Computer Science and Engineering at Institute of Engineering and Technology, Lucknow.

It is also certified that this project has not been submitted at any other Institute for the award of any other degrees to the best of my knowledge.

Dr. Promila Bahadur

Department of Computer Science and Engineering

Institute of Engineering and Technology, Lucknow

Acknowledgement

This work would not have been possible without the expert support of **Dr. Promila Bahadur**. We are especially indebted to her who has been supportive of our project goals and who worked actively to provide us with the protected academic time to pursue those goals. We are grateful to all of those with whom we have had the pleasure to work during this project. Each of the members of our CSE department has provided us extensive personal and professional guidance and taught us a great deal about both scientific research and life in general. We would especially like to thank **Prof. Girish Chandra**, the **HOD of CSE department**, as our teacher and mentor, he has taught us more than we could ever give him credit for. He has shown us, by his example, what a good teacher (and person) should be.

Nobody has been more important to us in the pursuit of this project than the members of our respective families. We would like to thank our parents, whose love and guidance are with us in whatever we pursue. They are the ultimate role models.

Abstract

Online job portals have become the preferred destination for job seekers and employers in today's digital world. They have improved the speed, ease of use, and accessibility of the job search process. However, these advantages also come with significant drawbacks, the most significant of which is the prevalence of fraudulent job advertisements. By creating phony job postings that can fool job seekers into divulging private information or even paying for positions that don't exist, scammers take advantage of the trust that people have in these platforms. In addition to causing financial loss, these scams also cause emotional distress, time wastage, and a decline in trust in online platforms.

Our "Fake Job Posting Detection" project aims to develop an intelligent, automated system that can recognize and flag questionable job postings before they are seen by job seekers in order to assist in addressing this expanding problem. The system analyzes job descriptions using machine learning to find trends or warning signs in the way they are written, organized, and displayed. It gains knowledge from actual instances of both authentic and fraudulent postings, which enables it to advance over time and keep up with changing scam tactics.

By adding this solution to job portals, we can improve their dependability and security. The system reduces the need for manual checks, speeds up the process of removing fraudulent posts, and ensures that users interact with only verified job opportunities. In addition to protecting users from fraud, it also helps maintain the credibility of hiring platforms.

The ultimate goal of this project is to create a transparent and safe online employment marketplace where employers can connect with qualified applicants without being hampered by unscrupulous actors and job seekers can apply with confidence. Solutions like these are crucial for maintaining system trust and keeping people safe as the online employment market grows.

Chapter 1

Introduction

1.1 Motivation

The growing concern over employment fraud, which disproportionately impacts vulnerable job seekers who fall for phony job postings, is what motivated this project. These frauds undermine confidence in online employment platforms by causing serious emotional distress in addition to financial loss and identity theft. The legitimacy of recruitment portals is being undermined as fraudulent listings increase in frequency. The evolving strategies of scammers, who frequently mimic respectable businesses and create convincing job descriptions, are too difficult for traditional, manual methods of fraud detection to keep up with. This demonstrates the pressing need for a machine learning-based, scalable solution that can effectively detect dubious job postings and rebuild confidence in the online employment market.

One of the biggest challenges in the fight against employment fraud is the intricacy of modern scams. Because fraudsters are constantly refining their strategies to evade detection systems, it is difficult for human reviewers to quickly and accurately identify fraudulent posts. Because there are so many job postings on popular platforms, manual moderation is not only time-consuming but also insufficient. Inspired by successful machine learning applications for detecting fake news and phishing emails, this project proposes a similar data-driven approach for detecting job fraud. By using labeled datasets of both authentic and fraudulent job postings to train models, the system is able to detect minute patterns and irregularities. This makes it possible to identify and eliminate fraudulent content in real time, which eventually makes the internet safer and more trustworthy.

1.2 Objective

This project's main objective is to develop and deploy an intelligent, automated system that uses machine learning to precisely identify fraudulent job postings. The likelihood of employment scams that prey on gullible job seekers is increasing along with the popularity of online job portals. In order to identify discrepancies and questionable trends that frequently point to fraud, this system analyzes a number of components of a job posting, including the job description, company name, salary range, application process, and contact information. The system will be trained on a labeled dataset of both authentic and fraudulent job advertisements using supervised learning algorithms. It will then be able to recognize the subtle distinctions between legitimate listings and scams, like odd wording, exaggerated compensation offers, omitted company information, or false contact details. The model will become a dependable tool for preventing fraud in real time as it learns and develops, increasing the accuracy of its detection capabilities.

But the project is more than just a technical one; it is also very motivated by the need to protect people, especially new graduates, remote job seekers, and others in vulnerable positions, from falling victim to these increasingly sophisticated scams. Fake job postings don't just waste time; they can have serious effects like losing money, having personal information stolen, and emotional trauma. This system can flag suspicious job posts before users even see them, greatly lowering the chances of someone being misled. It also helps job portals keep their credibility by making sure that the opportunities they present are safe, verified, and real.

This machine learning-based solution aims to make the online job search safer, more open, and more trustworthy for everyone involved, from job seekers to employers to platform administrators. Trust in digital platforms is important in today's world. Ultimately, this project isn't just about automation—it's about building trust and making job searching a little less risky for everyone.

1.3 Background

Job fraud is a significant issue that has gotten worse as more people use online job sites, which is what inspired this project. Since more and more people are searching for work online, scammers have developed new strategies to exploit this trend. Although job portals are helpful and have a large audience, they frequently struggle to adequately screen each job posting that is posted. This has allowed fraudulent job advertisements to pass through, deceiving job seekers into divulging personal information or even paying for fictitious positions.

It's not enough to just use old-fashioned ways to find fake job ads, like reading them by hand or waiting for users to report them. There are just too many job ads being posted every day. Scammers are getting smarter and often make posts that look real at first glance. Because of this, we really need a better solution that can keep up with these changing threats. That’s where this project comes in. The goal is to make job seekers safer and help rebuild trust in online job boards by using machine learning to automatically find and flag fake job ads.

1.4 Scope of the Project

To address the growing issue of fraudulent job postings on online recruitment platforms, the Fake Job Posting Detection project aims to design, develop, and implement an intelligent automated system that makes use of cutting-edge technologies. The following primary areas will be the focus of the project:

- ***Automatic Checking of Job Ads:*** The system will examine various aspects of job postings, such as the job title, description, company information, salary information, and contact details, to look for any unusual indications that the posting may be fraudulent.

- ***Using Machine Learning:*** The system will be trained on historical examples of authentic and fraudulent job advertisements, enabling it to distinguish between the two and correctly categorize new job postings.

- ***Instant Detection and Alerts:*** In order for the platform to act quickly before job seekers see them, the system will continuously monitor job postings and flag any that appear suspicious.

- ***Easy-to-Use Dashboard:*** Site administrators will be able to check flagged job ads and manage the verification process with ease thanks to a straightforward interface.

- ***Protecting Data and Privacy:*** The project will ensure that all private and work-related data is securely stored and handled in accordance with privacy regulations.

- ***Built to Grow and Adapt:*** The system will be able to adapt and update itself to continue detecting new forms of fraud as more job postings are received and con artists alter their strategies.

- ***Building Trust in Online Job Platforms:*** The ultimate objective is to reduce fraudulent job offers in order to make the workplace safer and more reliable for both employers and job seekers.

  .

Chapter 2

Literature Review

2.1 Related Works

2.1.1 Already existing similar models

One of the most urgent issues facing the modern digital world is cybercrime. Both people and organizations are more susceptible to attacks that could cause significant financial losses and compromised personal data as a result of the growth of online activities and linked systems. With yearly global costs estimated in the trillions, the extent of these damages is astounding. The need for cutting-edge cybersecurity measures and creative strategies to identify and stop cyberthreats before they do serious damage has increased as a result of this concerning growth. Building strong detection models and real-time defenses is crucial to protecting digital infrastructure as cybercriminals change their strategies [1].

Fake news is spreading quickly, particularly on social media, which has seriously impacted public awareness and trust. False information spreads swiftly, swaying public opinion and occasionally having real-world repercussions. This is being addressed by applying machine learning classification algorithms in conjunction with sophisticated text analysis techniques like n-gram feature extraction to more precisely identify false content. These models are able to differentiate between authentic reporting and fake news by examining the frequency and patterns of word sequences in articles. Support Vector Machines are one type of classifier that has shown excellent performance in accurately detecting misleading data. Expanding datasets and improving methods to better tackle the growing problem of misinformation are the main goals of ongoing research in this field [2].

Spam emails are still a big annoyance and a serious security risk for users everywhere. In addition to flooding inboxes, these unsolicited messages frequently contain malicious attachments or links meant to steal personal data or infect systems with malware. Based on the presence of particular keywords and structural elements like headers or sender information, methods such as Naive Bayes classifiers assess the likelihood that an email is spam. These models can adjust to changing spam

strategies and increase their accuracy by regularly training on fresh data [3].

2.1.2 Machine Learning, preprocessing, Feature Extraction and Model Training Techniques

Usually, sockpuppet detection is tackled as a binary classification problem, in which accounts are categorized as either real users or sockpuppets, which are fake or multiple accounts controlled by the same user. The thorough examination of various feature sets, such as linguistic patterns, behavioral characteristics, and network interaction metrics, is crucial to the effectiveness of detection techniques. Researchers have enhanced the detection models' accuracy, precision, recall, and F-measure by investigating these different features. Furthermore, in order to better capture the intricacies of identity deception and text categorization difficulties across various domains, research has gone beyond binary classification to multiclass classification [4].

As a fundamental part of supervised learning, classification algorithms are essential in the field of detecting fraudulent job postings. These algorithms are intended to correctly classify new, unseen job postings into one of these categories after learning from labeled training data, in which each job posting is marked as either real or fake. The significance of classification algorithms is found in their capacity to recognize intricate patterns and connections between the target class and input features (like a job description, title, and company profile). Because of this, they are particularly effective in decision-making tasks with categorical outcomes. Automating fraud detection in massive job datasets is made possible by supervised classification algorithms' ability to generalize to new data by utilizing historical examples. This helps users avoid falling for false job offers and increases the effectiveness of screening procedures [5].

Since feature extraction converts unstructured text into structured data that machine learning models can process efficiently, it is a fundamental step in text classification tasks like fake job detection. Critical textual information like word frequency, phrase patterns, syntactic structures, or semantic meanings is captured by efficient feature extraction techniques. By strengthening their capacity to distinguish between authentic and fraudulent postings, these extracted features help classifiers such as support vector machines or deep neural networks perform better. In addition to increasing accuracy, well-designed feature extraction lowers computational complexity, which expedites the training and prediction stages [6].

By transforming unstructured text into numerical vectors, vectorization is essential to this feature extraction procedure. The way these vectors depict documents maintains important details about the words they contain and their contexts. Common vectorization techniques include Word2Vec, which embeds words in continuous vector spaces to capture semantic relationships between them; TF-IDF, which weighs words according to their importance in relation to the corpus is processed using the bag-of-words approach, which involves counting the frequency of each word. Selecting the appropriate vectorization method is essential since it has a direct effect on the model's capacity to learn and generalize from text data, guaranteeing that pertinent patterns are found while reducing noise [7].

As it highlights keywords that are unique within each document, TF-IDF is particularly significant among these techniques for identifying fraudulent job postings. TF-IDF assists models in concentrating on the most informative features by assigning greater weight to terms that occur frequently in a particular document but infrequently throughout the entire corpus. By highlighting minute variations in word usage that frequently indicate fraudulent postings, this selective weighting enhances the classifier's capacity to detect misleading content. For large-scale text mining applications, TF-IDF is a viable option due to its relative efficiency and simplicity [8].

Additionally, the training and evaluation of fake job posting detection models are greatly impacted by the selection of evaluation metrics and loss functions. Loss functions guide the optimization process to increase model accuracy by quantifying the error between predicted and true labels during training. The model's performance is gauged by metrics like accuracy, precision, recall, and F1-score, especially in datasets that are unbalanced and may have a significantly lower number of fraudulent posts than authentic ones. By making sure these factors are optimized, the model will learn efficiently and generalize well to new data, reducing false positives and false negatives in real-world applications [9].

Chapter 3

Methodology

3.1 Methodology

This project's goal is to create a machine learning-based system that can identify phony job postings. To accurately identify phony job postings, the methodology employs a methodical process that includes data collection, preprocessing, feature extraction, and classification using a neural network model. The workflow is made to guarantee high performance and resilience in practical applications.

3.1.1 Data Collection

The main dataset for this project came from Kaggle, a popular platform for machine learning datasets and competitions, and was used to train and assess the model for detecting fake job postings. The publicly accessible "Fake Job Postings" dataset offers a solid basis for supervised learning in fraud detection. With 17,880 job posting entries classified as either legitimate (0) or fraudulent (1), it is well suited for machine learning-based binary classification tasks. The 18 unique features in this dataset—which include textual, categorical, and binary attributes—capture comprehensive details about every job posting. Among these characteristics are :

1. ***Title*** – The title of the job (e.g., Software Engineer, Marketing Manager), which can provide insight into the type and level of the position offered.
2. ***Location*** – Indicates the geographical location of the job, helping to detect unrealistic or mismatched job locations.
3. ***Department*** – Specifies the department in which the role is based, such as Sales, IT, or Human Resources.
4. ***Salary Range*** – Gives an estimate of the compensation offered; missing or unusually high/low values can hint at fraud.
5. ***Company Profile*** – A textual summary of the company's background and values. Fraudulent

postings often lack company details or contain generic content.

6. ***Description*** – The full job description outlining duties and responsibilities. Language patterns here are critical for detecting suspicious postings.
7. ***Requirements*** – A list of skills, qualifications, and certifications needed for the role.
8. ***Benefits*** – Any perks or benefits mentioned, such as insurance, bonuses, or remote work. Overly generous or absent benefits can be red flags.
9. ***Telecommuting*** – A binary flag (1 or 0) indicating whether the job allows remote work.
10. ***Has Company Logo*** – A binary feature showing whether the company's logo is present in the listing. Fraudulent advertisements often do not include company logos.
11. ***Has Questions*** – Indicates whether the job posting includes additional screening questions, which are more likely in genuine postings.
12. ***Employment Type*** – Specifies if the job is full-time, part-time, contract-based, or temporary.
13. ***Required Experience*** – States the experience level needed (e.g., Entry-level, Mid-Senior).
14. ***Required Education*** – Lists the educational qualifications required for the role.
15. ***Industry*** – Describes the business sector, such as Finance, IT, or Healthcare.
16. ***Function*** – Defines the core function of the role (e.g., Engineering, Customer Support).
17. ***Fraudulent*** – This is the target variable that specifies if a job posting is genuine (0) or fraudulent(1).

In order to teach the model the fundamental patterns and traits that distinguish authentic job postings from fraudulent ones, each of these features is essential. Particularly rich in context are text-based fields such as requirements, company profile, and description; these will subsequently be processed using natural language processing (NLP) techniques to extract significant patterns.

The Kaggle API will be utilized to acquire this dataset. The Kaggle API, a command-line interface tool, allows users to access datasets on the platform programmatically. The dataset can be automatically and systematically downloaded into the project environment through the API, guaranteeing reproducibility and updating simplicity.

- First, the Kaggle API will be installed and authenticated using the user's Kaggle API token.
- The dataset will then be fetched using the appropriate API command.
- The downloaded dataset will be stored locally or in a cloud-based development environment and then read into the project for preprocessing and further analysis.

3.1.2 Data Preprocessing

To ensure the dataset is clean and suitable for machine learning applications, thorough preprocessing is performed following data collection. The performance of classification models can be significantly impacted by raw data, which is frequently unstructured, inconsistent, or incomplete. Preprocessing guarantees that the data is standardized, clean, and prepared for model training and feature extraction. The model may pick up false patterns or make inaccurate predictions if this step skipped.

The dataset will undergo the following crucial preprocessing steps in order to overcome these obstacles:

- ***Removal of HTML tags*** from text fields (e.g., job descriptions) to eliminate embedded code and retain only the human-readable content.
- ***Stripping of extra white spaces*** to ensure consistent formatting across all textual inputs.
- ***Removal of emojis and special characters*** that do not contribute to the semantic meaning of job postings and may introduce noise.
- ***Conversion of all text to lowercase*** to maintain consistency and reduce the vocabulary size for textual analysis.
- ***Removal of punctuation marks*** to eliminate irrelevant symbols that may interfere with keyword detection.
- ***Encoding of categorical variables*** using techniques such as one-hot encoding or label encoding, making them compatible with machine learning models.
- ***Detection and treatment of outliers*** in numerical fields like salary range or required experience, ensuring that extreme values do not bias the model.

In order to convert unstructured, unorganized data into a format that is machine-readable, these preprocessing steps are necessary. In addition to increasing model accuracy, clean and consistent data saves computation and training time. Preprocessing enables the model to concentrate on significant patterns by eliminating extraneous or noisy data and standardizing text and numerical features, which eventually results in predictions that are more accurate and broadly applicable when identifying fraudulent job postings.

3.1.3 Feature Extraction

The goal of the feature extraction phase is to transform unprocessed textual and categorical data into a numerical format that machine learning models can comprehend and use efficiently. It is crucial to numerically represent the vast amounts of unstructured textual data found in job postings, including job descriptions, company profiles, qualifications, and benefits, without sacrificing its semantic meaning.

We will do this by applying a popular natural language processing (NLP) technique called TF-IDF (Term Frequency-Inverse Document Frequency), which determines a word's importance to a document in relation to a corpus, or collection of documents. TF-IDF is useful for classification tasks like identifying fraudulent job postings because it is especially good at differentiating words that are common and significant in a particular document but uncommon in other documents.

How TF-IDF Works

TF-IDF consists of two main components:

1. ***Term Frequency (TF):*** Indicates how frequently a term appears within a document, adjusted by the total word count to allow comparison across documents of varying lengths.

TF (word, doc) = (Number of occurrences of the word in the document) ÷ (Total number of words in the document).

2. ***Inverse Document Frequency (IDF):*** Measures how rare a word is across all documents in the corpus. Common words found across multiple documents are assigned lower weights, whereas rare or unique words are given higher importance.

IDF (word) = log(Total number of documents/Number of documents containing the word)

TF-IDF Score: A high TF-IDF value indicates that a word appears frequently in a specific document but is uncommon across the rest of the corpus, highlighting its significance within that document.

$$\text{TF-IDF} = \text{TF} \times \text{IDF}$$

Application in Our Project

In our project:

- TF-IDF will be applied to textual features such as description, requirements, company_profile, benefits, and other free-text fields in the dataset.
- Each job posting will be converted into a high-dimensional vector where each element represents the TF-IDF score of a specific word from the overall vocabulary.
- This vectorized representation will become the input feature set for the classification model.

In addition to textual features like industry, function, required education, and employment type, categorical attributes will also be encoded using one-hot or label encoding, depending on the distribution and model compatibility.

In order to prevent models from placing undue weight on larger numerical values, numerical features such as salary range will be standardized to ensure they are on a similar scale. Lastly, to cut down on noise and boost model efficiency, redundant or unnecessary features will be removed, such as fields with a lot of missing values or those with little variability. By capturing the key elements of every job posting, this feature extraction procedure produces a clear, insightful, and condensed dataset that greatly improves the classification model's capacity to distinguish between authentic and fraudulent listings.

3.1.4 Data Preparation

All processed features, including those obtained from textual data (using TF-IDF), encoded categorical variables, and standardized numerical attributes, are merged into a single, cohesive input vector that corresponds to each Once the feature extraction phase is complete, the job posting is ready for further analysis or modeling. The corresponding labels, which indicate whether a job posting is authentic or fraudulent, function as the dependent variable (target) for the classification task, whereas these input vectors are the independent variables (features).

The complete dataset is divided into two subsets—a training set and a validation set—in order to facilitate supervised learning. For evaluating the model's capacity to generalize beyond the examples it has already seen, this division is essential. The classification model is fitted to the training set, which enables it to discover correlations and patterns between the input features and their labels.

As the model is being developed, this technique aids in tracking its behavior and identifying

problems like overfitting, which occurs when a model performs well on training data but is unable to generalize to new inputs. This systematic method of preparing and assessing datasets guarantees that the resulting model is reliable and accurate, allowing it to be used in practical situations like spotting phony job advertisements on online job boards.

3.1.5 Neural Network Design

The classification model is designed as a fully connected feedforward neural network consisting of three main components: an input layer, one or more hidden layers, and an output layer.

- ***Input Layer*:** The input layer receives the feature vector xi ∈ R^n for each job posting, where n is the number of extracted features.
- ***Hidden Layers*:** Each of the network's numerous hidden layers is made up of several neurons. Through nonlinear transformations on their inputs, these layers discover intricate patterns and relationships within the data. The Rectified Linear Unit (ReLU), an activation function utilized in hidden layers, is described as follows:

  $$ReLU(z)=max(0,z)$$

  where z is a neuron's weighted sum of inputs plus bias. By adding nonlinearity, the ReLU function helps the network model complex functions and lessens the vanishing gradient issue during backpropagation.

- ***Output Layer*:** The output layer consists of a single neuron with a sigmoid activation function, which outputs a value between 0 and 1 representing the probability that a job posting is fraudulent :

  $$\sigma(z)=1/1+e^{-z}$$

.

- ***Weight Initialization*:** Weight initialization methods like Xavier (Glorot) Initialization or He Initialization are used to encourage effective training and avoid problems like vanishing or exploding gradients. By basing the initial weights on the size of the preceding layer, these techniques guarantee that the activation variance stays constant across the network.
- **Loss Function:** The model is trained by minimizing the Binary Cross-Entropy Loss, defined as :

$$\mathcal{L} = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right]$$

Fig.(1). Binary Cross-Entropy Loss

- **Optimizer:**

  The Adam optimizer, an adaptive learning rate optimization algorithm that combines the benefits of momentum and RMSProp approaches, is used to update the network weights. Adam speeds up convergence and facilitates the model's effective attainment of ideal parameters.

3.1.6 Model Training

The model will be trained using a suitable loss function, such as binary cross-entropy, to minimize the difference between predicted and actual labels. To speed up convergence and effectively update model parameters, an optimizer such as Adam or SGD will be employed. Cross-validation, which enables evaluation across several training and validation splits, will be used to make sure the model generalizes well to unseen data. This increases the robustness of the model and aids in identifying overfitting or underfitting. Performance indicators like accuracy, precision, recall, and F1-score will be continuously tracked throughout training. The model's ability to distinguish between genuine and fraudulent job postings is maintained at a high level thanks to these metrics. To avoid overfitting, regularization strategies like early stopping may also be applied.

3.1.7 Model Evaluation

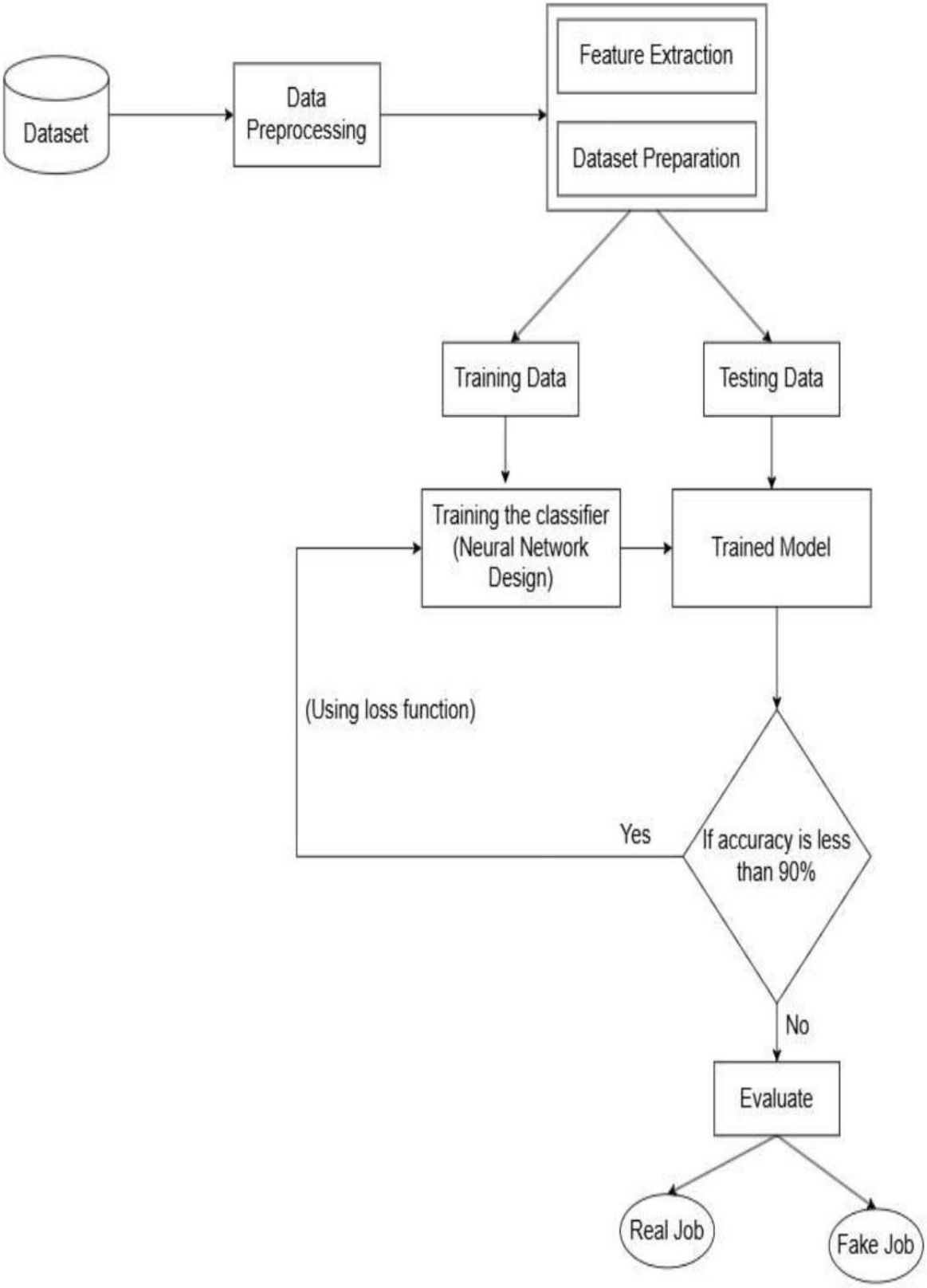
Several metrics will be used to evaluate the model's performance in order to determine how accurate and capable it is of classifying job postings. The outcomes will direct any adjustments required to enhance the model's functionality. Important aspects like the model's recall in identifying all fraudulent postings, accuracy in identifying true positives, and ability to accurately differentiate between genuine and fraudulent listings will all be evaluated. In order to make sure the model is not overfitting to particular patterns, the evaluation will also concentrate on how well

it handles various job categories and how well it generalizes across a variety of datasets.

In order to prevent bias, the model's capacity to handle class imbalances between authentic and fraudulent postings will also be investigated. To find possible flaws, like false positives or false negatives, and direct future improvements, a thorough examination of misclassifications will also be carried out. The model's accuracy, precision, and recall will be improved by adjusting parameters, retraining with more data, or changing feature extraction methods in response to the evaluation's findings. This will guarantee that the model can successfully identify fraudulent job postings.

3.1.8 Output and Model Deployment

A dependable and automated method for identifying phony job postings on online platforms will be provided by the final trained model, which will categorize job postings as authentic or fraudulent. The model's output will help job seekers lower their risk of scams by clearly indicating whether a particular job posting is legitimate or suspicious. The model will be implemented through a web page where users can enter job-related information and get an instant prediction of the posting's legitimacy, making it accessible and easy to use. For both individuals and organizations, this output interface guarantees real-time usability. Additionally, the model's predictions can be incorporated into hiring platforms to automatically weed out fraudulent listings before they are seen by users.



`

Fig.(2). Flowchart For Model Implementation

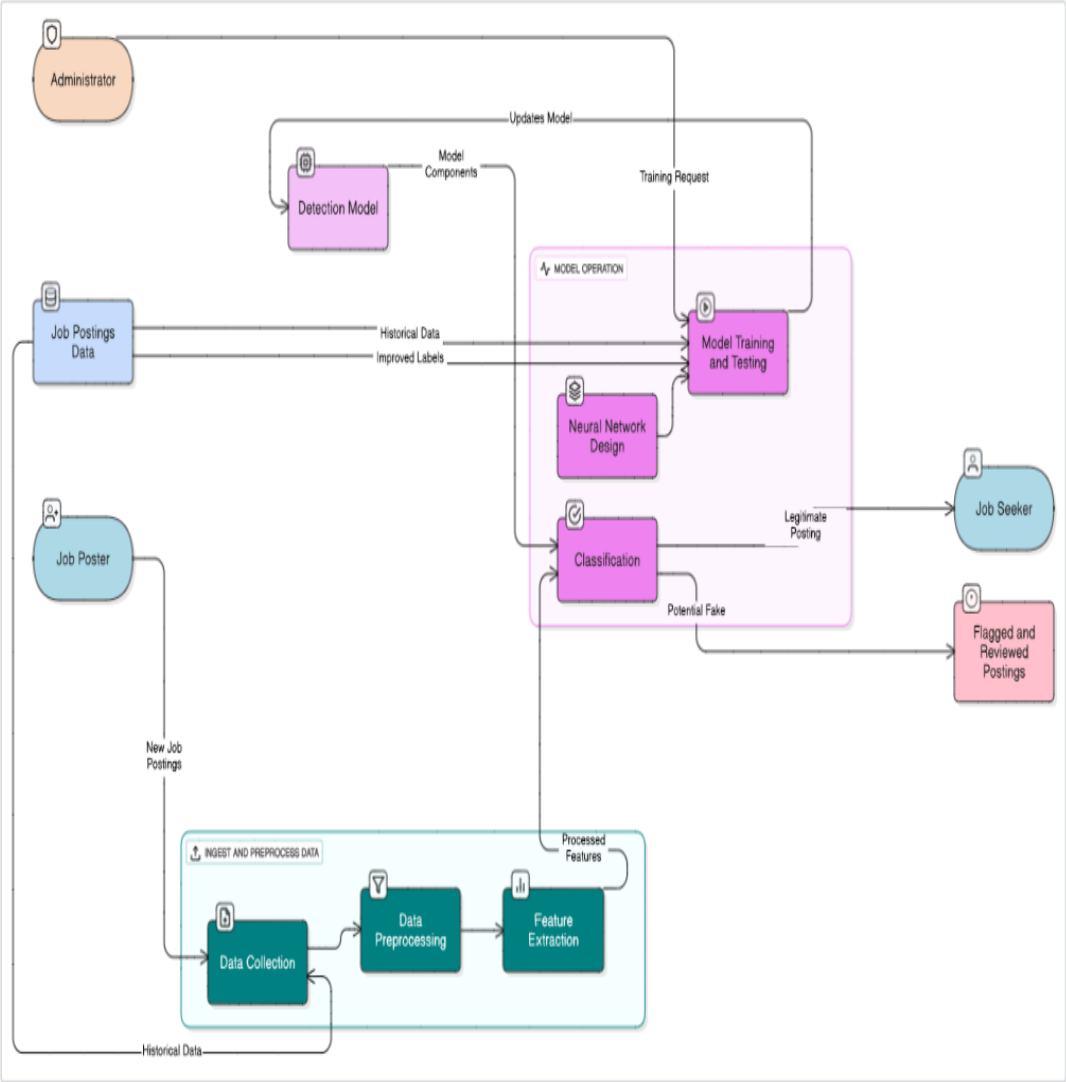


Fig.(3). Data Flow Diagram of the Model

Chapter 4

EXPERIMENTAL RESULT

4.1 Confusion Matrix:

Confusion matrix, at the end of the model training and evaluation process, a confusion matrix is generated to assess the performance of the classification model. This matrix presents a tabular comparison between the actual and predicted labels, comprising four key components: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). These relate to both legitimate and fraudulent job postings that were either accurately or inaccurately predicted in our context. In addition to calculating overall accuracy, this matrix aids in the computation of more informative metrics like precision, recall, and F1-score. The model's performance in handling imbalanced datasets, where one class (like actual jobs) may be noticeably more prevalent than the other, is better illustrated by these extra metrics. For a sensitive application like fake job detection, the confusion matrix is especially helpful because it shows the types of mistakes the model is making. For example, we want to prevent fraudulent jobs from being classified as real if the model has a high number of false negatives. On the other hand, a high percentage of false positives can indicate that genuine job postings are mistakenly reported as fraudulent, which could negatively impact the user experience. As a result, the confusion matrix directs additional model improvement in addition to offering performance insights.

4.2 Class Statistics:

To better understand the model's performance in each class—legitimate and fraudulent job postings—we also examine class-wise statistics in addition to the confusion matrix. When there is a class imbalance and one class may dominate the dataset, this thorough breakdown is extremely crucial. By calculating metrics such as class-specific precision, recall, F1-score, and specificity, class statistics enable us to assess the model's capacity to differentiate between each class. Beyond these, more sophisticated metrics that provide deeper insights into classification balance and prediction confidence include the Area Under the ROC Curve (AUC), G-mean, Matthews Correlation Coefficient (MCC), and entropy-based measures. Class statistics are important because

they can show whether the model is performing fairly across both classes or is biased toward the majority class. For instance, the model might exhibit a high overall accuracy but be useless in practice if it performs exceptionally well for class 0 (legitimate jobs) but poorly for class 1 (fraudulent jobs). We make sure the model is accurate and dependable in identifying fraudulent job postings by assessing these class-specific metrics. This helps with decision-making, model tuning, and, eventually, a more trustworthy deployment.

4.3 Web Interface:

Using HTML and CSS for the frontend design and the Flask web framework, a web interface will be created to enable end users to access the trained model. Through a straightforward form, users will be able to enter job posting details like the job title, description, company profile, and other pertinent fields. The backend Flask server will process the input after the user submits the form, transform it using the vectorizers that were saved, and then send it to the machine learning model. Without requiring technical expertise, users will be able to assess job authenticity quickly and effectively thanks to this instant feedback mechanism.

In order to give recruiters, job portal administrators, and job seekers an interactive and user-friendly platform, the web interface is essential. Even non-technical users can take advantage of the model's capabilities thanks to its user-friendly design. This interface also establishes the groundwork for future scalability, as it can be expanded into a mobile application, integrated into current job portals, or deployed on cloud platforms after validation. The web interface improves the overall dependability and trust in online job platforms by providing a real-time, accessible solution that connects the machine learning model to practical use cases.

Chapter 5

CONCLUSION

This project employs machine learning methods to build a system capable of identifying fraudulent job postings.The training and testing dataset came from Kaggle, which offered a large collection of job postings classified as either authentic or fraudulent. We carefully preprocessed the data before training the model, handling missing values, cleaning text fields, and standardizing formats to enhance input quality and guarantee better learning. In order to create a model that could generalize effectively and prevent overfitting as a result of noisy or inconsistent data, this preprocessing step was essential.

To extract significant features from the text data, we employed the TF-IDF (Term Frequency–Inverse Document Frequency) technique. By quantifying the significance of each word in the job postings, TF-IDF allowed the model to concentrate on key terms that are more likely to differentiate between real and fraudulent listings. A neural network that was trained and cross-validated to guarantee accuracy and robustness was used to implement the classification model. Strong precision and recall scores, along with a high accuracy of roughly 96%, demonstrated the model's effectiveness in accurately identifying both fraudulent and legitimate jobs.

We developed a web interface using HTML, CSS, and the Flask framework to make the solution user-friendly and accessible. Users can enter job-related information on this page, including the title, description, and requirements, and get an immediate indication of whether the job posting is authentic or fraudulent. This offers a user-friendly experience that makes it useful in the real world, particularly for HR platforms and job seekers. In addition to simplifying interaction, the interface lays the groundwork for future integration into mobile or web applications.

Overall, our project brings together essential steps like preprocessing, feature extraction, model training, evaluation, and deployment into a unified pipeline for detecting fraudulent job listings. The model’s strong performance and the addition of a web interface make it both effective and practical. With further improvements—such as additional feature inputs or cloud-based

deployment—this system has the potential to evolve into a powerful tool for combating online job scams.

APPENDIX A

CODE SNIPPETS

A.1 Main Code :

app.py | index.html | style.css | main.py | kaggle.json | download_data.py | job_posting.p

```python
import torch
import pandas as pd
import random
from job_posting import JobPosting, JobPostingsDataset
import network
import joblib
from pycm import ConfusionMatrix
from skorch import NeuralNetClassifier
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_predict

CSV_FILENAME = "job_dataset.csv"
CSV_FILE_DELIMITER = ","
EPOCHS_COUNT = 200
LEARNING_RATE = 0.0001
BATCH_SIZE = 32
KFOLD_PARTITIONS_COUNT = 5
FAKE_TO_REAL_RATIO = 1.0/20.0
TRAINING_PARALLEL_JOBS_COUNT = -1  # -1 means maximum


def prepare_dataset():
    global job_postings
    global all_job_posting_data
    global all_job_posting_targets
    global device
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    random.shuffle(job_postings)
    job_postings = JobPostingsDataset(job_postings)
    job_postings.prepare_all_text_vectorizers()
    all_job_posting_data = torch.tensor([job_posting.get_data_list() for job_posting in job_postings]).float().to(device)
```

```
def read_job_postings_data_from_csv(csv_filename=CSV_FILENAME, csv_file_delimiter=CSV_FILE_DELIMITER):
    global job_postings
    job_postings = []
    csv_contents = pd.read_csv(csv_filename, delimiter=csv_file_delimiter, dtype=str)
    print("Size of the dataset is :")
    print(csv_contents.shape)
    # Reads a CSV file into a pandas DataFrame, treating all values as string
    rows = len(csv_contents)
    cols = len(csv_contents.iloc[0])
    for i in range(rows):
        data = csv_contents.iloc[i]
        job_posting = JobPosting(data["title"], data["location"], data["department"], data["salary_range"],
                                 data["company_profile"], data["description"], data["requirements"],
                                 data["benefits"], data["telecommuting"], data["has_company_logo"],
                                 data["has_questions"], data["employment_type"], data["required_experience"],
                                 data["required_education"], data["industry"], data["function"], data["fraudulent"])
        job_postings.append(job_posting)


def initialize_network():
    global model
    global optimizer
    global criterion
    model = network.Network().to(device)
    criterion = torch.nn.BCEWithLogitsLoss
    optimizer = torch.optim.Adam
    for m in model._modules:
        network.normal_init(model._modules[m], 0, 1)


def print_confusion_matrix(actual_labels, predicted_labels):
    cm = ConfusionMatrix(actual_labels, predicted_labels)
    print(cm)
```

```
if __name__ == "__main__":
    print("Reading CSV file")
    read_job_postings_data_from_csv()
    print("Preparing dataset")
    prepare_dataset()
    print("Initializing network")
    initialize_network()
    print("Creating classifier")
    classifier = NeuralNetClassifier(network.Network, max_epochs=EPOCHS_COUNT, lr=LEARNING_RATE,
                                     train_split=None, criterion=criterion, optimizer=optimizer,
                                     batch_size=BATCH_SIZE,
                                     criterion__pos_weight=torch.tensor([FAKE_TO_REAL_RATIO, 1.0]))
    print("Training")
    predictions = cross_val_predict(classifier, all_job_posting_data, all_job_posting_targets,
                                    cv=KFOLD_PARTITIONS_COUNT, verbose=1, n_jobs=TRAINING_PARALLEL_JOBS_COUNT)
    predicted_labels = torch.from_numpy(predictions)
    predicted_labels = torch.argmax(predicted_labels, dim=1).tolist()
    actual_labels = torch.argmax(all_job_posting_targets, dim=1).tolist()
    print_confusion_matrix(actual_labels, predicted_labels)
    print("Training input feature size:", all_job_posting_data.shape[1])

    print("Training final model on full dataset...")
    final_model = NeuralNetClassifier(
        network.Network,
        max_epochs=EPOCHS_COUNT,
        lr=LEARNING_RATE,
        train_split=None,
        criterion=criterion,
        optimizer=optimizer,
        batch_size=BATCH_SIZE,
        criterion__pos_weight=torch.tensor([FAKE_TO_REAL_RATIO, 1.0])
    )
```

```
        final_model.fit(all_job_posting_data, all_job_posting_targets)

        # Predict on the training data
        train_predictions = final_model.predict(all_job_posting_data)

        # Get actual labels
        actual_labels = torch.argmax(all_job_posting_targets, dim=1).cpu().numpy()

        if len(train_predictions.shape) > 1 and train_predictions.shape[1] > 1:
            train_predictions = np.argmax(train_predictions, axis=1)

        train_accuracy = accuracy_score(actual_labels, train_predictions)
        print(f"Training Accuracy: {train_accuracy * 100:.2f}%")

        # Save model
        final_model.save_params(f_params="saved_model.pt")
        print("Model saved to saved_model.pt")

        vectorizers = {
            'title': job_postings.title_vectorizer,
            'location': job_postings.location_vectorizer,
            'department': job_postings.department_vectorizer,
            'company_profile': job_postings.company_profile_vectorizer,
            'description': job_postings.description_vectorizer,
            'requirement': job_postings.requirement_vectorizer,
            'benefit': job_postings.benefit_vectorizer,
            'employment_type': job_postings.employment_type_vectorizer,
            'required_experience': job_postings.required_experience_vectorizer,
            'required_education': job_postings.required_education_vectorizer,
            'industry': job_postings.industry_vectorizer,
            'function': job_postings.function_vectorizer
        }
```

A.2 Preprocessing Job Posting Dataset and Creating Vectorizers:

```
import torch
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from copy import deepcopy

SMALL_TEXT_VECTOR_SIZE = 10
MEDIUM_TEXT_VECTOR_SIZE = 25
LARGE_TEXT_VECTOR_SIZE = 50


class JobPosting:
    def __init__(self, title="", location="", department="", salary_range="", company_profile="",
                 description="", requirement="", benefit="", telecommuting="", has_company_logo="",
                 has_question="", employment_type="", required_experience="",
                 required_education="", industry="", function="", fraudulent=""):
        self.title = title if isinstance(title, str) else ""
        self.location = location if isinstance(location, str) else ""
        self.department = department if isinstance(department, str) else ""
        self.salary_range =[salary_range if isinstance(salary_range, str) else ""]
        self.company_profile = company_profile if isinstance(company_profile, str) else ""
        self.description = description if isinstance(description, str) else ""
        self.requirement = requirement if isinstance(requirement, str) else ""
        self.benefit = benefit if isinstance(benefit, str) else ""
        self.telecommuting = telecommuting if isinstance(telecommuting, str) else ""
        self.has_company_logo = has_company_logo if str(isinstance(has_company_logo, int)) else ""
        self.has_question = has_question if isinstance(has_question, str) else ""
        self.employment_type = employment_type if isinstance(employment_type, str) else ""
        self.required_experience = required_experience if isinstance(required_experience, str) else ""
        self.required_education = required_education if isinstance(required_education, str) else ""
        self.industry = industry if isinstance(industry, str) else ""
        self.function = function if isinstance(function, str) else ""
        self.fraudulent = fraudulent if isinstance(fraudulent, str) else ""
```

```
    def get_data_list(self):
        return [*self.title, *self.location, *self.department, *self.salary_range, *self.company_profile,
                *self.description, *self.requirement, *self.benefit, self.telecommuting, self.has_company_logo,
                self.has_question, *self.employment_type, *self.required_experience,
                *self.required_education, *self.industry, *self.function]

    def get_target_list(self):
        if int(self.fraudulent) == 0:
            return [1.0, 0.0]  # Real Job Posting
        else:
            return [0.0, 1.0]  # Fake Job Posting

    def get_target(self):
        return int(self.fraudulent)  # Converts Fraudulent to an integer 0/1

    def get_data_tensor(self):
        return torch.tensor(self.get_data_list())  # Converts Job data list into pytorch tensor


class JobPostingsDataset(torch.utils.data.Dataset):  # used to load and manage data efficiently when training models
    def __init__(self, job_postings_list=[]):
        self.job_postings_list = job_postings_list
        self.vectorized_job_postings_dict = {}

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

    def prepare_all_text_vectorizers(self):
        self.title_vectorizer = TfidfVectorizer(max_features=MEDIUM_TEXT_VECTOR_SIZE)
        all_titles_list = [self.preprocess_text(job_posting.title) for job_posting in self.job_postings_list]
        self.title_vectorizer.fit_transform(all_titles_list)
```

```
        all_employment_types_list = [self.preprocess_text(job_posting.employment_type) for job_posting in
                                     self.job_postings_list]
        self.employment_type_vectorizer.fit_transform(all_employment_types_list)

        self.required_experience_vectorizer = TfidfVectorizer(max_features=SMALL_TEXT_VECTOR_SIZE)
        all_required_experiences_list = [self.preprocess_text(job_posting.required_experience) for job_posting in
                                         self.job_postings_list]
        self.required_experience_vectorizer.fit_transform(all_required_experiences_list)

        self.required_education_vectorizer = TfidfVectorizer(max_features=SMALL_TEXT_VECTOR_SIZE)
        all_required_educations_list = [self.preprocess_text(job_posting.required_education) for job_posting in
                                        self.job_postings_list]
        self.required_education_vectorizer.fit_transform(all_required_educations_list)

        self.industry_vectorizer = TfidfVectorizer(max_features=MEDIUM_TEXT_VECTOR_SIZE)
        all_industries_list = [self.preprocess_text(job_posting.industry) for job_posting in self.job_postings_list]
        self.industry_vectorizer.fit_transform(all_industries_list)

        self.function_vectorizer = TfidfVectorizer(max_features=MEDIUM_TEXT_VECTOR_SIZE)
        all_functions_list = [self.preprocess_text(job_posting.function) for job_posting in self.job_postings_list]
        self.function_vectorizer.fit_transform(all_functions_list)

    @staticmethod
    def preprocess_text(text):
        text = re.sub("<[^>]*>", "", text)  # Remove HTML Tags
        symbols = re.findall("(?::|;|=)(?:-)?(?:\)|\(|D|P)", text)  # Find Emojis
        text = (re.sub("[\W]+", " ", text.lower()) + " ".join(symbols).replace("-", ""))  # Convert to lowercase
        return text

    def __getitem__(self, index):
        if index in self.vectorized_job_postings_dict:
            return self.vectorized_job_postings_dict[index]
        else:
```

```
            vectorized_job_posting.company_profile = \
            self.company_profile_vectorizer.transform([vectorized_job_posting.company_profile]).toarray()[0]
            vectorized_job_posting.description = \
            self.description_vectorizer.transform([vectorized_job_posting.description]).toarray()[0]
            vectorized_job_posting.requirement = \
            self.requirement_vectorizer.transform([vectorized_job_posting.requirement]).toarray()[0]
            vectorized_job_posting.benefit = \
            self.benefit_vectorizer.transform([vectorized_job_posting.benefit]).toarray()[0]
            vectorized_job_posting.employment_type = \
            self.employment_type_vectorizer.transform([vectorized_job_posting.employment_type]).toarray()[0]
            vectorized_job_posting.required_experience = \
            self.required_experience_vectorizer.transform([vectorized_job_posting.required_experience]).toarray()[0]
            vectorized_job_posting.required_education = \
            self.required_education_vectorizer.transform([vectorized_job_posting.required_education]).toarray()[0]
            vectorized_job_posting.industry = \
            self.industry_vectorizer.transform([vectorized_job_posting.industry]).toarray()[0]
            vectorized_job_posting.function = \
            self.function_vectorizer.transform([vectorized_job_posting.function]).toarray()[0]
            if len(list(vectorized_job_posting.salary_range[0].split("-"))) == 2:
                try:
                    vectorized_job_posting.salary_range = tuple(
                        map(int, vectorized_job_posting.salary_range[0].split("-")))
                except:
                    vectorized_job_posting.salary_range = (0, 0)
            else:
                vectorized_job_posting.salary_range = (0, 0)
            vectorized_job_posting.telecommuting = int(vectorized_job_posting.telecommuting)
            vectorized_job_posting.has_company_logo = int(vectorized_job_posting.has_company_logo)
            vectorized_job_posting.has_question = int(vectorized_job_posting.has_question)
            vectorized_job_posting.fraudulent = int(vectorized_job_posting.fraudulent)
            self.vectorized_job_postings_dict[index] = vectorized_job_posting
            return self.vectorized_job_postings_dict[index]
```

A.3 Neural Network Training

```
import torch
from torch import nn, optim
from torch.nn import functional as F

NETWORK_INPUT_SIZE = 341
NETWORK_OUTPUT_SIZE = 2


class Network(nn.Module):
    def __init__(self, input_size=NETWORK_INPUT_SIZE, output_size=NETWORK_OUTPUT_SIZE):
        super(Network, self).__init__()
        self.fc1 = nn.Linear(input_size, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 64)
        self.fc4 = nn.Linear(64, 32)
        self.fc5 = nn.Linear(32, 16)
        self.fc6 = nn.Linear(16, 8)
        self.fc7 = nn.Linear(8, 4)
        self.fc8 = nn.Linear(4, output_size)

    def forward(self, x):
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        x = F.relu(x)
        x = self.fc4(x)
        x = F.relu(x)
        x = self.fc5(x)
        x = F.relu(x)
        x = self.fc6(x)
```

```
        x = self.fc6(x)
        x = F.relu(x)
        x = self.fc7(x)
        x = F.relu(x)
        x = self.fc8(x)
        return x


def normal_init(m, mean, std):
    if isinstance(m, nn.Linear):
        m.weight.data.normal_(mean, std)
        m.bias.data.zero_()
```

A.4 File using saved model for predicting job as real or fake

```
import torch
import joblib
from job_posting import JobPosting
from network import Network

VECTOR_FILE = "vectorizers.pkl"
MODEL_FILE = "saved_model.pt"


def load_vectorizers():
    return joblib.load(VECTOR_FILE)


def load_model():
    model = Network()
    model.load_state_dict(torch.load(MODEL_FILE, map_location=torch.device("cpu")))
    model.eval()
    return model


def vectorize_job_posting(job_posting, vectorizers):
    # print("Available vectorizer keys:", vectorizers.keys())
    # Apply TF-IDF transformations
    features = []
    features += vectorizers["title"].transform([job_posting.title]).toarray()[0].tolist()
    features += vectorizers["location"].transform([job_posting.location]).toarray()[0].tolist()
    features += vectorizers["department"].transform([job_posting.department]).toarray()[0].tolist()
    features += vectorizers["company_profile"].transform([job_posting.company_profile].toarray()[0].tolist()
    features += vectorizers["description"].transform([job_posting.description].toarray()[0].tolist()
    features += vectorizers["requirement"].transform([job_posting.requirement]).toarray()[0].tolist()
    features += vectorizers["benefit"].transform([job_posting.benefit]).toarray()[0].tolist()
    features.append(int(job_posting.telecommuting))
```

```
    features.append(int(job_posting.has_question))
    features += vectorizers["employment_type"].transform([job_posting.employment_type]).toarray()[0].tolist()
    features += vectorizers["required_experience"].transform([job_posting.required_experience]).toarray()[0].tolist()
    features += vectorizers["required_education"].transform([job_posting.required_education]).toarray()[0].tolist()
    features += vectorizers["industry"].transform([job_posting.industry]).toarray()[0].tolist()
    features += vectorizers["function"].transform([job_posting.function]).toarray()[0].tolist()

    # Parse salary_range
    if isinstance(job_posting.salary_range, str) and "-" in job_posting.salary_range:
        try:
            low, high = map(int, job_posting.salary_range.split("-"))
            features += [low, high]
        except:
            features += [0, 0]
    else:
        features += [0, 0]
    # try:
    #     features.append(int(job_posting.fraudulent) if job_posting.fraudulent != "" else 0)
    # except ValueError:
    #     features.append(0)  # Default to 0 if fraudulent is not a valid integer
    return torch.tensor(features, dtype=torch.float32).unsqueeze(0)


def predict(job_posting):
    vectorizers = load_vectorizers()
    model = load_model()
    vectorized_input = vectorize_job_posting(job_posting, vectorizers)
    with torch.no_grad():
        output = model(vectorized_input)
        prediction = torch.argmax(torch.softmax(output, dim=1)).item()
        return "Real" if prediction == 0 else "Fake"
```

```
if __name__ == "__main__":
    # Example test case
    # sample_job = JobPosting(
    #         title="Online Survey Taker - Earn ₹5,000/day!", #fake
    #         location="Work from Anywhere",
    #         department="Customer Feedback",
    #         salary_range="₹5,000 - ₹10,000 daily",
    #         company_profile="We are a global insights firm seeking individuals to give feedback on daily products. No previous experience required.",
    #         description="Complete simple online surveys and earn money. No skills required. Just 15 minutes per day needed!",
    #         requirement="Internet connection, basic reading ability, willingness to complete 5-10 surveys daily.",
    #         benefit="Daily payments via Paytm or UPI, flexible hours, work from mobile or computer.",
    #         telecommuting="1",
    #         has_company_logo="0",
    #         has_question="0",
    #         employment_type="Part-time",
    #         required_experience="No experience",
    #         required_education="No formal education",
    #         industry="Market Research",
    #         function="Data Collection"
    #     )

    result = predict(sample_job)
    print("Prediction for sample job:",result)
```

A.5 Web Page using Flask

```python
from flask import Flask, render_template, request
from job_posting import JobPosting
from predict import predict  # Your prediction logic

app = Flask(__name__)


@app.route("/", methods=["GET", "POST"])
def home():
    form_data = {}
    result = None

    if request.method == "POST":
        form = request.form
        form_data = form.to_dict()

        job = JobPosting(
            title=form.get("title", ""),
            location=form.get("location", ""),
            department=form.get("department", ""),
            salary_range=form.get("salary_range", ""),
            company_profile=form.get("company_profile", ""),
            description=form.get("description", ""),
            requirement=form.get("requirements", ""),
            benefit=form.get("benefits", ""),
            telecommuting=form.get("telecommuting", "0"),
            has_company_logo=form.get("has_company_logo", "0"),
            has_question=form.get("has_questions", "0"),
            employment_type=form.get("employment_type", ""),
            required_experience=form.get("required_experience", ""),
            required_education=form.get("required_education", ""),
            industry=form.get("industry", ""),
```

```python
            required_experience=form.get("required_experience", ""),
            required_education=form.get("required_education", ""),
            industry=form.get("industry", ""),
            function=form.get("function", "")
        )

        result = predict(job)

    return render_template("index.html", result=result, form_data=form_data)


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

A.6 HTML of Webpage

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Job Posting Authenticity Checker</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}" />
</head>
<body>
  <div class="container">
    <div class="title">Job Posting Authenticity Checker</div>
    <div class="content">
      <form action="/" method="POST">
        <div class="user-details">
          <div class="input-box">
            <span class="details required">Job Title</span>
            <input
              type="text"
              name="title"
              placeholder="e.g. Software Engineer"
              required
              value="{{ form_data.get('title', '') }}"
            />
          </div>
          <div class="input-box">
            <span class="details required">Location</span>
            <input
              type="text"
              name="location"
              placeholder="e.g. New York, NY"
              required
              value="{{ form_data.get('location', '') }}"
```

```
              type="text"
              name="industry"
              placeholder="e.g. Software"
              value="{{ form_data.get('industry', '') }}"
            />
          </div>
          <div class="input-box">
            <span class="details">Function</span>
            <input
              type="text"
              name="function"
              placeholder="e.g. Engineering"
              value="{{ form_data.get('function', '') }}"
            />
          </div>
        </div>
        <div class="button">
          <input type="submit" value="Check Authenticity" />
        </div>
      </form>

      {% if result %}
      <div
        class="prediction-result"
        style="margin-top: 30px; font-size: 20px; font-weight: bold; color: {% if result == 'Real' %}green{% else %}red{% endif %};"
      >
        Prediction: {{ result }}
      </div>
      {% endif %}
    </div>
  </div>
</body>
</html>
```

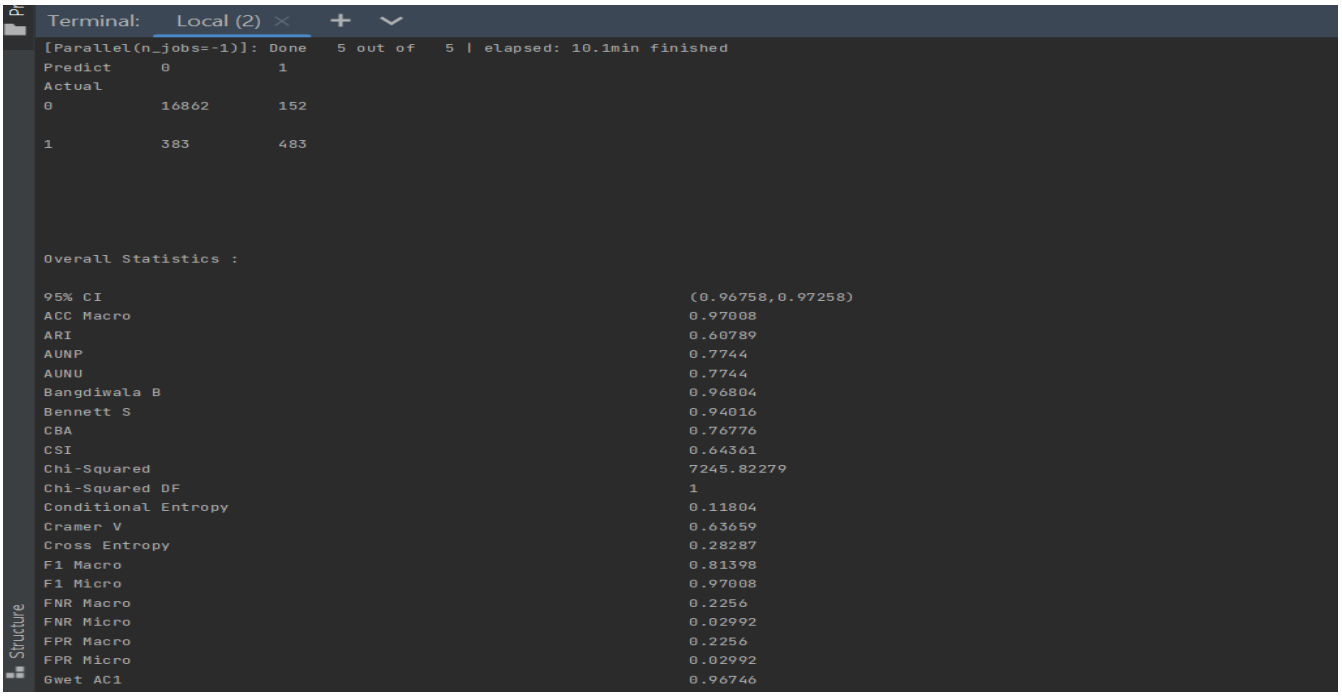
A.7 CSS of Webpage

```css
/* Import Google Fonts - Poppins */
@import url('https://fonts.googleapis.com/css2?family=Poppins:wght@200;300;400;500;600;700&display=swap');

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: 'Poppins', sans-serif;
}

body {
  min-height: 100vh;
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 20px;
  background: linear-gradient(135deg, #74ebd5, #acb6e5);
}

.container {
  width: 100%;
  max-width: 720px;
  background-color: #fff;
  padding: 30px 35px;
  border-radius: 10px;
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
}

.container .title {
  font-size: 28px;
  font-weight: 600;
  position: relative;
  margin-bottom: 20px;
  color: #333;
```

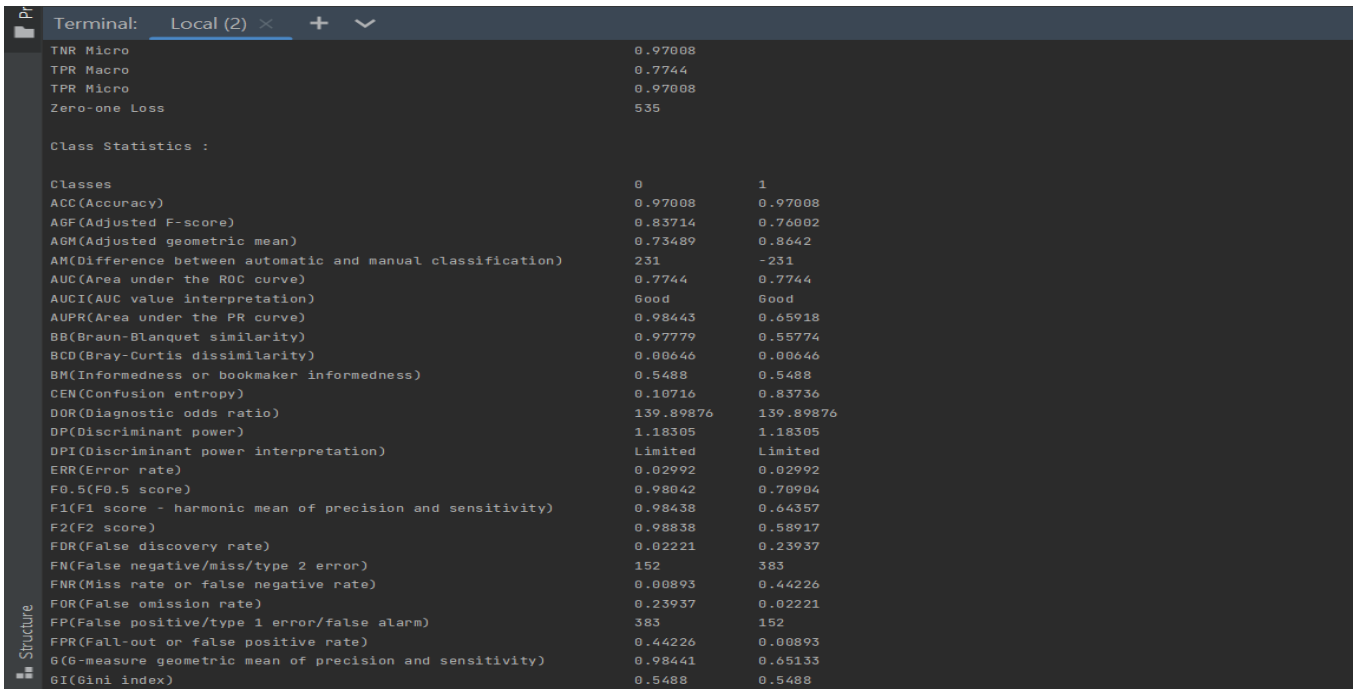
```css
  font-weight: 600;
  letter-spacing: 1px;
  cursor: pointer;
  transition: background 0.3s ease;
}

form .button input:hover {
  background: linear-gradient(-135deg, #74ebd5, #acb6e5);
}

/* Responsive */
@media (max-width: 768px) {
  .user-details {
    flex-direction: column;
  }

  form .user-details .input-box {
    width: 100%;
  }
}

@media (max-width: 480px) {
  .container {
    padding: 20px;
  }

  .container .title {
    font-size: 22px;
  }

  form .button input {
    font-size: 16px;
  }
}
```

APPENDIX B
SNAPSHOTS

B.1 Confusion Matrix Outcome and Accuracy

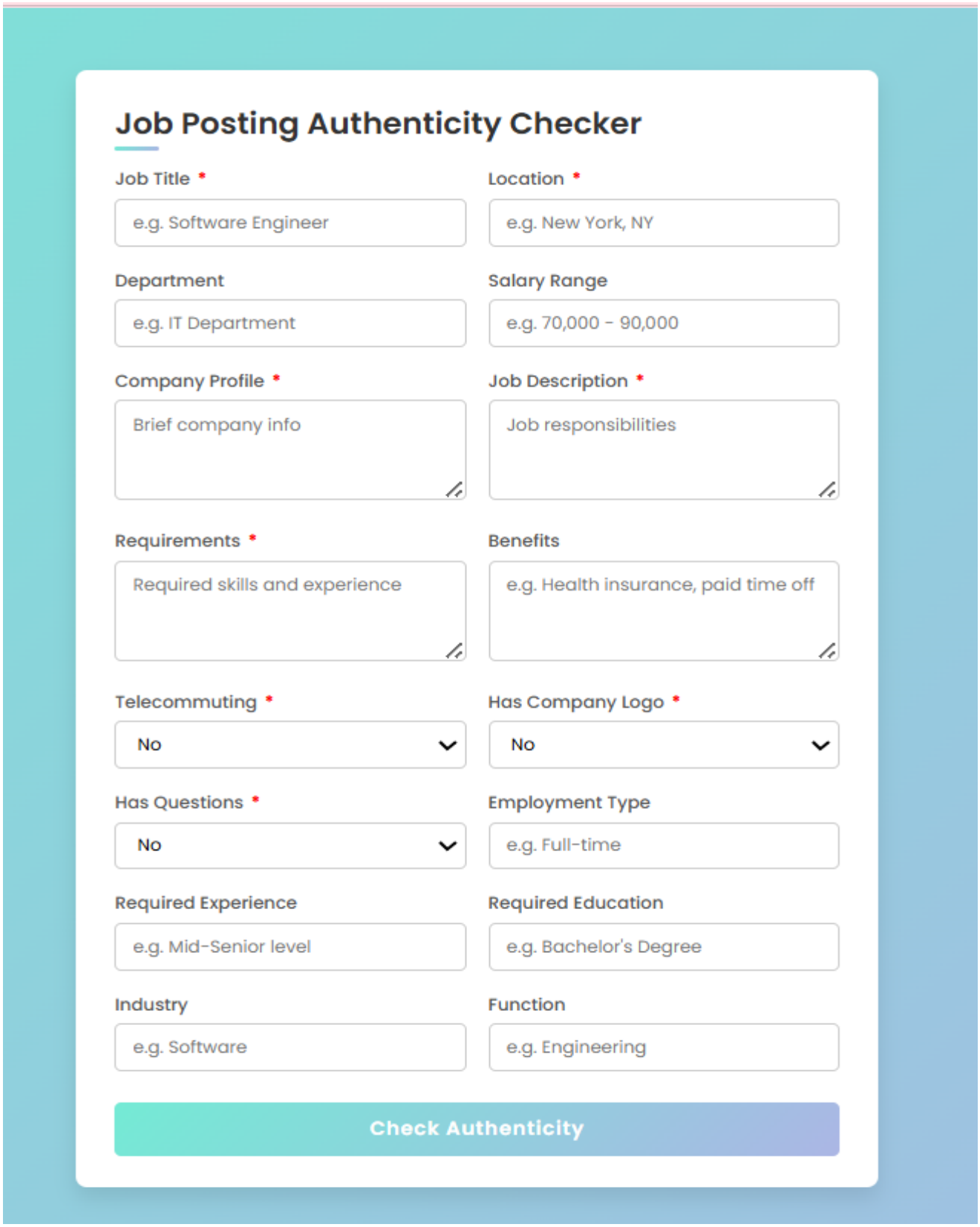


```
Terminal:  Local (2)
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed: 10.1min finished
Predict     0           1
Actual
0           16862       152

1           383         483



Overall Statistics :

95% CI                                                          (0.96758,0.97258)
ACC Macro                                                       0.97008
ARI                                                             0.60789
AUNP                                                            0.7744
AUNU                                                            0.7744
Bangdiwala B                                                    0.96804
Bennett S                                                       0.94016
CBA                                                             0.76776
CSI                                                             0.64361
Chi-Squared                                                     7245.82279
Chi-Squared DF                                                  1
Conditional Entropy                                             0.11804
Cramer V                                                        0.63659
Cross Entropy                                                   0.28287
F1 Macro                                                        0.81398
F1 Micro                                                        0.97008
FNR Macro                                                       0.2256
FNR Micro                                                       0.02992
FPR Macro                                                       0.2256
FPR Micro                                                       0.02992
Gwet AC1                                                        0.96746
```

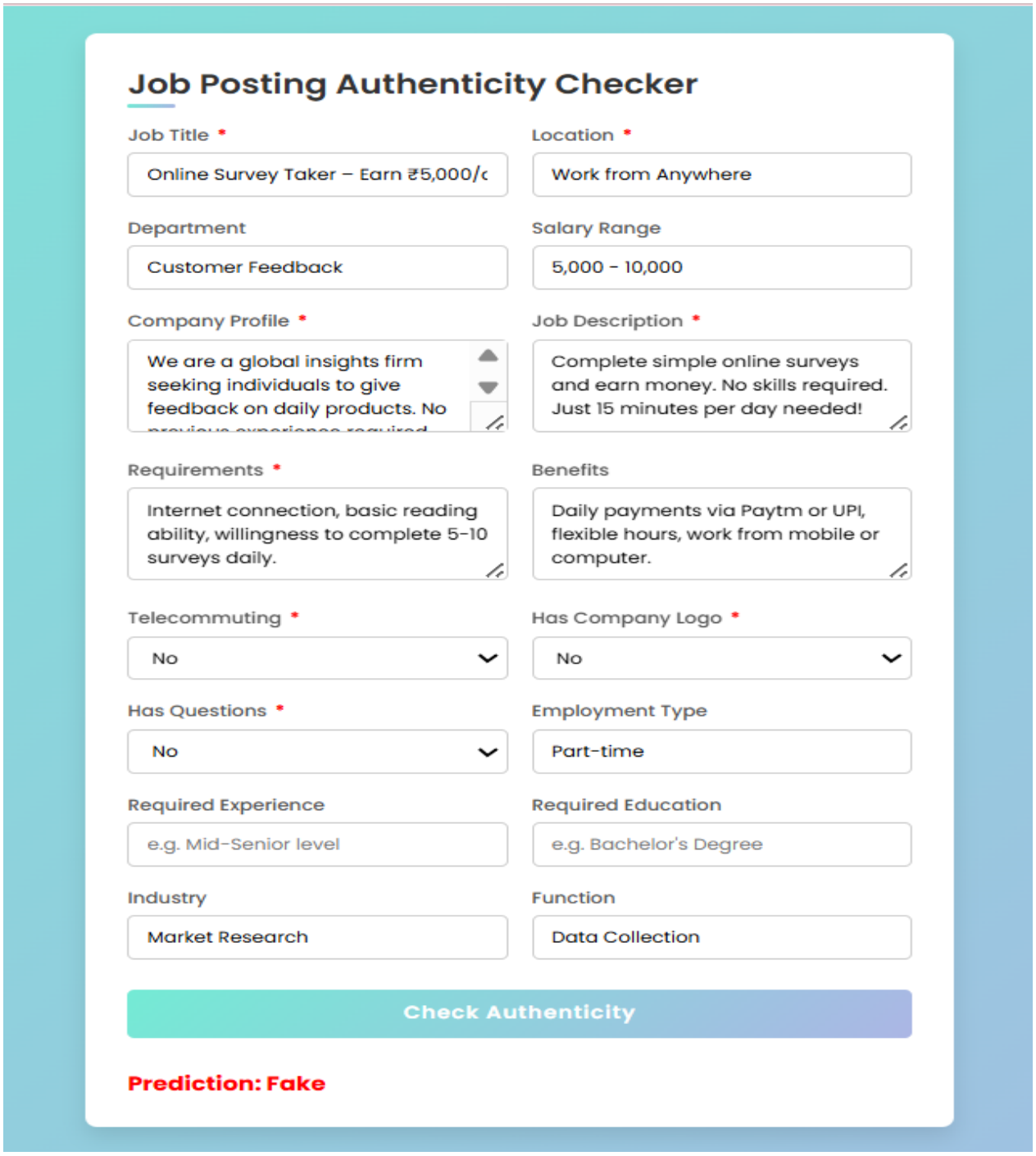


```
Terminal:  Local (2)
TNR Micro                                                       0.97008
TPR Macro                                                       0.7744
TPR Micro                                                       0.97008
Zero-one Loss                                                   535

Class Statistics :

Classes                                                         0             1
ACC(Accuracy)                                                   0.97008       0.97008
AGF(Adjusted F-score)                                           0.83714       0.76002
AGM(Adjusted geometric mean)                                    0.73489       0.8642
AM(Difference between automatic and manual classification)      231           -231
AUC(Area under the ROC curve)                                   0.7744        0.7744
AUCI(AUC value interpretation)                                  Good          Good
AUPR(Area under the PR curve)                                   0.98443       0.65918
BB(Braun-Blanquet similarity)                                   0.97779       0.55774
BCD(Bray-Curtis dissimilarity)                                  0.00646       0.00646
BM(Informedness or bookmaker informedness)                      0.5488        0.5488
CEN(Confusion entropy)                                          0.10716       0.83736
DOR(Diagnostic odds ratio)                                      139.89876     139.89876
DP(Discriminant power)                                          1.18305       1.18305
DPI(Discriminant power interpretation)                          Limited       Limited
ERR(Error rate)                                                 0.02992       0.02992
F0.5(F0.5 score)                                                0.98042       0.70904
F1(F1 score - harmonic mean of precision and sensitivity)       0.98438       0.64357
F2(F2 score)                                                    0.98838       0.58917
FDR(False discovery rate)                                       0.02221       0.23937
FN(False negative/miss/type 2 error)                            152           383
FNR(Miss rate or false negative rate)                           0.00893       0.44226
FOR(False omission rate)                                        0.23937       0.02221
FP(False positive/type 1 error/false alarm)                     383           152
FPR(Fall-out or false positive rate)                            0.44226       0.00893
G(G-measure geometric mean of precision and sensitivity)        0.98441       0.65133
GI(Gini index)                                                  0.5488        0.5488
```

B.2 UI of the Webpage



B.3 Webpage predicting a fake job



REFERENCES

[1]. Morgan, S. and Menlo Park, C. (2017) *Cybercrime Report from the Editors at Cybersecurity Ventures*. Herjavec Group, Toronto

[2]. Ahmed, H., Traore, I., & Saad, S. (2017*).* "Detection of online fake news using n-gram analysis and machine learning techniques" in *international conference on intelligent, secure, and dependable systems in distributed and cloud environments, 127-138*

[3]. T. Sultana, K.A. Sapnaz, F. Sana, J. Najath, "Email based Spam Detection", *International Journal of Engineering Research & Technology (IJERT), Vol. 9 Issue 06, June-2020*

[4]. R. Kumari, S. Srivastava, "Machine Learning: A Review on Binary Classification", *International Journal of Computer Applications (0975 – 8887) Volume 160 – No 7, February 2017*

[5]. A.F.A.H. ALNUAIMI , T. H.K. ALBALDAWI, "An overview of machine learning classification techniques", *BIO Web of Conferences 97, 00133 (2024), ISCKU 2024*

[6]. Shaker, Saif & Alhajim, Dhafer & Al-khazaali, Ahmed & Hussein, Hussein & Athab, Ali., "Feature Extraction based Text Classification: A review", *JOURNAL OF ALGEBRAIC STATISTICS Volume 13, No. 1, 2022, p. 646-653*

[7]. Kozhevnikov, Vadim & Pankratova, Evgeniya, "Research of the Text Data Vectorization and Classification Algorithms of Machine Learning." *International Scientific Journal Theoretical & Applied Science, Vol. 85, Issue 5, (2020)*

[8]. S. Qaiser, R. Ali, "Text Mining: Use of TF-IDF to Examine the Relevance of Words to Documents", *International Journal of Computer Applications (0975 – 8887) Volume 181 – No.1, July 2018*

[9]. Terven, Juan & Cordova-Esparza, Diana-Margarita & Ramirez-Pedraza, Alfonzo Chávez Urbiola, Edgar. (2023). "Loss Functions and Metrics in Deep Learning. A Review". *10.48550/arXiv.2307.02694*.