A

MINI PROJECT REPORT

On

FAKE JOB DETECTION USING MACHINE LEARNING FROM WEB-SCRAPED JOB POSTINGS

BY

KARRI MANOJ KUMAR

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The project "Fake Job Detection Using Machine Learning from Web-Scraped Job Postings" aims to develop an intelligent and automated system to identify fraudulent job advertisements posted online. With the increasing prevalence of scam job listings that deceive applicants and compromise their personal information, this project addresses a critical need for digital safety and employment integrity. The system will allow users to input a job posting URL, which will be programmatically accessed, parsed, and analyzed to assess the legitimacy of the job listing.

Utilizing web scraping technologies, the system extracts job description (JD) content and key metadata directly from the provided URL. It then verifies the authenticity of the link checking whether the URL leads to an official careers or recruitment page — ensuring the posting originates from a credible source. Once validated, the job description undergoes a series of machine learning-based evaluations using a pre-trained classification model. This model will analyze various linguistic, structural, and contextual parameters commonly associated with fraudulent postings, such as vague job roles, urgent hiring language, suspicious contact methods, and more.

The full application will be developed using Streamlit, offering users a simple and interactive web interface to enter job URLs and receive instant feedback on the genuineness of the job. Core technologies include Python, BeautifulSoup/Selenium for scraping, scikit-learn/NLTK for feature extraction and classification, and Streamlit for front-end deployment. Emphasis will be placed on building a robust pipeline for data validation, model accuracy, and user-friendly reporting.

This project will be designed with scalability, accuracy, and user security in mind, creating a powerful tool for the modern job market to combat fraudulent employment schemes effectively.

CHAPTER 1
INTRODUCTION

1.1 Introduction

The project "Fake Job Detection Using Machine Learning from Web-Scraped Job Postings" - The rise of online job portals has made job hunting easier but also increased the risk of fake job postings. Many scammers use convincing job ads to exploit job seekers for personal data or money. This project aims to detect such fake job postings using machine learning. By taking a job URL as input, the system checks if the link is valid and career-related. It then scrapes the job description and analyzes it to predict if the job is genuine or fake. The application is built using Streamlit for a simple and interactive user experience.

1.2 Project Overview

The internet has revolutionized the job search process, offering countless opportunities through company websites and third-party job portals such as Indeed, LinkedIn, Naukri, and Glassdoor. While this digital transformation has improved convenience and accessibility, it has also opened the door to job-related frauds and scams. Fake job postings are increasingly used to deceive applicants, extract personal data, or even demand money under false pretenses. Detecting and reporting such scams is a challenge due to the sheer volume and subtlety of fraudulent content.

1.3 Key Technologies

The application is developed using several modern technologies. Streamlit is used for building the front-end user interface, while Python is the core programming language. Web scraping is handled using Requests and BeautifulSoup libraries. For machine learning, the project uses scikit-learn with a text classification pipeline consisting of TfidfVectorizer and Logistic Regression for detecting fake job indicators based on textual patterns.

1.4 Features and Functionality

The system offers multiple useful features including real-time URL validation to check whether the link leads to a job-related page, dynamic extraction of job descriptions from the provided URL, and a machine learning-based prediction that indicates whether the job is genuine or fake. The tool supports a wide range of job platforms and gives users clear and immediate feedback on the results.

1.5 Benefits and Impact

This project offers several benefits. It helps job seekers avoid falling victim to fraudulent job listings and improves their online safety. It also saves time by automating the verification process and promotes trust in digital hiring platforms. The system demonstrates how machine learning can be applied to solve real-world problems with direct impact on users' lives.

1.6 Scalability and Security

The system is built with scalability in mind and can be extended to support additional job portals and larger datasets for training. It is designed to handle a growing number of users and input URLs. In terms of security, the application can be enhanced with input validation, error handling, and request throttling to prevent abuse, making it a robust and secure tool for public use.

CHAPTER 2
LITERATURE SURVEY

2.1 Literature Survey

The rise of online job portals has made recruitment easier but also increased fake job scams. Traditional rule-based detection methods lack adaptability, leading to the adoption of machine learning models like Logistic Regression and SVM for better accuracy. Natural Language Processing (NLP) techniques such as TF-IDF and text vectorization are commonly used for analyzing job descriptions. Datasets like the Kaggle Fake Job Postings dataset have enabled effective model training and evaluation. Web scraping tools like BeautifulSoup help extract real-time job data from URLs for analysis. Streamlit is increasingly used to create interactive ML-based apps that offer real-time job authenticity prediction.

2.2 Background of Online Recruitment and Associated Risks

With the growth of online job portals and digital recruitment platforms, job seekers increasingly rely on the internet to discover employment opportunities. However, the anonymity and low entry barriers of the web have also led to a rise in fraudulent job postings, aimed at phishing personal information, extorting money, or propagating scams. This creates a strong demand for automated systems that can accurately distinguish between genuine and fake job listings.

2.3 Existing Work on Job Posting Fraud Detection

The Several studies have explored machine learning-based detection of fake job listings. One of the most widely referenced datasets is the Kaggle Fake Job Postings Dataset, which includes real and fraudulent job postings labeled for training classification models. Researchers have applied models such as Logistic Regression, Naive Bayes, Support Vector Machines (SVM), and even deep learning techniques to classify job descriptions based on extracted features like word frequency, length, grammar, and suspicious phrases.

For example, Sahin and Duman (2019) proposed an SVM-based model that achieved high accuracy in detecting scam job postings by analyzing linguistic features. Another approach by Kumar et al. (2021) combined content-based features with metadata such as job title, company name, and location to improve prediction accuracy using ensemble methods.

2.4 Overview of Existing Detection Techniques

Early fake job detection systems primarily relied on rule-based filters or manual moderation, which are limited in scalability and adaptability. Modern approaches have adopted machine learning techniques, using classification models trained on labeled datasets. For example, classifiers like Naive Bayes, Decision Trees, SVM, and Logistic Regression have been trained on text-based features to identify scam indicators in job descriptions.

More recently, deep learning models like LSTM and BERT have been explored for their ability to capture contextual and semantic meaning in job descriptions. Some systems also integrate website metadata analysis (e.g., domain age, HTTPS status) and user behavior (e.g., application frequency, click patterns) for more robust detection.

2.5 Role of Natural Language Processing (NLP)

NLP is integral to processing and interpreting job description text. Techniques such as TF-IDF vectorization, n-gram analysis, and part-of-speech tagging help transform unstructured text into meaningful features for machine learning algorithms. More recent studies have also explored word embeddings and transformer-based models like BERT for capturing deeper semantic information, improving accuracy in classifying subtle scams.

Natural Language Processing (NLP) plays a central role in detecting fake job postings by enabling machines to understand and analyze human language. In this project, NLP is used to extract meaningful patterns from job descriptions and identify deceptive or suspicious language commonly found in fraudulent postings. Techniques like tokenization, stop word removal, and TF-IDF vectorization help convert raw text into structured numerical data that can be used by machine learning models. NLP also enables the model to capture subtle differences in tone, grammar, and phrasing between genuine and fake job listings. Ultimately, NLP empowers the system to make accurate predictions by analyzing the linguistic features of job ads scraped from the web.

2.6 Application of Machine Learning Algorithms

Several supervised learning models have shown promise in the domain. Logistic Regression is commonly used for its effectiveness in binary classification, while ensemble methods like Random Forests or XGBoost enhance performance through feature aggregation. Studies by Sahin et al. (2020) and Kumar et al. (2021) demonstrated high detection accuracy by combining text analysis with job metadata such as location, company profile, and domain keywords.

Binary Classification of Job Postings (Fake vs Genuine)

- **ML Algorithm Used:** Logistic Regression
- **Application:**

  The primary use of ML in your project is to classify job descriptions as either **fake** (1) or **genuine** (0). Logistic Regression, a simple and interpretable classifier, is trained on a labeled dataset of job texts. The algorithm learns to detect linguistic and contextual patterns that often indicate scams (e.g., exaggerated promises, lack of formal structure, no qualifications mentioned).
- **Alternative Algorithms You Can Try:**
  - **Support Vector Machines (SVM):** For sharper boundaries between classes
  - **Random Forest:** For more robust, non-linear decision boundaries
  - **Naive Bayes:** Especially effective for text classification problems
  - **XGBoost or LightGBM:** For high performance in larger datasets

2.7 Web Scraping as a Data Collection Strategy

To build real-time fake job detection tools, many researchers have employed web scraping to dynamically extract job listings from popular portals like LinkedIn, Naukri, and Indeed. Libraries such as BeautifulSoup, Selenium, and Scrapy are commonly used to gather job descriptions, company names, and application details. However, challenges arise in handling dynamic web content, anti-bot protections, and varying HTML structures, requiring adaptive scraping strategies.

Web scraping refers to the automated process of programmatically accessing web pages and extracting relevant information using tools and libraries such as **BeautifulSoup**, **Selenium**, or **Scrapy** in Python. For this project, the objective of web scraping is to obtain **job description texts** along with related metadata like **job titles, company names, job types (remote/on-site), and benefit details**. This data serves as the raw material for building and training the machine learning model to detect fake job postings.

Advantages of Web Scraping

- **Real-Time Data Collection:** Enables the system to gather the latest job posts dynamically, making the model adaptable to new scam patterns.
- **Scalability:** Facilitates the collection of large datasets without manual intervention.
- **Cost-Effective:** Reduces the need for purchasing commercial job datasets, many of which may not be up-to-date.
- **Customizability:** Allows targeting specific websites, fields, or job sectors, ensuring relevance of data to the domain.

Gaps in Existing Research and Proposed Solution

Despite the progress, many existing studies lack end-to-end systems that validate the authenticity of job URLs, perform live content extraction, and conduct on-the-fly predictions. The proposed project fills this gap by introducing a system that:

Accepts URLs from company and third-party job sites

Scrapes the job description using robust, platform-aware selectors

Validates page authenticity

Predicts whether the posting is genuine or fake using a trained ML model

This approach not only builds on prior research but also adds a layer of real-world usability.

CHAPTER 3

ANALYSIS

3.1 Analysis

The "Fake Job Detection Using Machine Learning from Web-Scraped Job Postings " project aims to detects fake job postings by analyzing job URLs using machine learning.

It uses web scraping to extract job descriptions from both company and third-party job portals.

Natural Language Processing (NLP) techniques like TF-IDF are used for feature extraction.

A Logistic Regression model classifies jobs as fake or genuine based on the extracted content.

Streamlit provides a simple UI for users to input URLs and view results.

The tool checks URL validity and job-related content before analysis.

This system helps job seekers avoid scams by flagging suspicious job listings in real time.

The core idea was to take real-world job posting content—whether scraped from well-known platforms like Indeed, LinkedIn, or lesser-known job boards—and train a model to distinguish between **genuine and fake job listings**. To bring this vision to life, we adopted a multi-stage approach combining web scraping, natural language processing (NLP), and supervised machine learning.

Data Collection and Preparation

The project started with the collection of job posting data. Using **web scraping tools and techniques**, we extracted job titles, descriptions, and other relevant metadata from multiple job-related websites. In our prototype, we created a compact but representative sample dataset featuring examples of both real and scam job postings.

Each posting was manually labeled as either **genuine (0)** or **fake (1)** based on language cues, offer credibility, and presence or absence of typical job-related features. This labeling formed the **ground truth** for our machine learning model.

Text Analysis and Feature Extraction

The next step involved converting the textual content into features that a model could understand. We used **TF-IDF vectorization** to highlight the most significant words in a post. Words like "career,"

"salary," "remote," "benefits," and "guaranteed income" played a key role in distinguishing legitimate roles from scams.

Additionally, we manually engineered features such as:

- **Remote work availability** (detected through keywords like “work from home” or “telecommute”)
- **Mention of benefits** (like insurance, 401k, paid leave, etc.)

These features provided context, adding structure to the unstructured job text and helping the model pick up patterns commonly associated with real vs. fake jobs.

Model Training and Prediction

We chose a simple yet effective machine learning pipeline using **Logistic Regression**. This classifier, when paired with TF-IDF and our engineered features, proved highly interpretable and efficient for a binary classification task.

Upon testing, the model was able to accurately flag potentially fake job postings, using not only the vocabulary used in the description but also signals from the structure and tone of the ad. The model predicted with high confidence on clear-cut cases (e.g., spammy, over-promising job ads), and with moderate confidence on more ambiguous postings.

Real-Time Analysis Using Streamlit

To bring this technology closer to end-users, we developed an **interactive web application using Streamlit**. The app accepts a job URL, scrapes the content in real-time, analyzes the job description, and returns a verdict:

- **Is the job post real or fake?**
- **Is it remote or on-site?**
- **Are benefits offered or not?**

This added a layer of practicality, allowing users—especially job seekers—to make informed decisions before applying or sharing personal details.

3.2 Data Exploration

The data is combination of integer, binary and textual datatypes. A brief definition of the variables is given below:

Table 1. Table of variables

#	Variable	Datatype	Description
1	job_id	int	Identification number given to each job posting
2	title	text	A name that describes the position or job
3	location	text	Information about where the job is located
4	department	text	Information about the department this job is offered by
5	salary_range	text	Expected salary range
6	company_profile	text	Information about the company
7	description	text	A brief description about the position offered
8	requirements	text	Pre-requisites to qualify for the job
9	benefits	text	Benefits provided by the job
10	telecommuting	boolean	Is work from home or remote work allowed
11	has_company_logo	boolean	Does the job posting have a company logo
12	has_questions	boolean	Does the job posting have any questions
13	employment_type	text	5 categories – Full-time, part-time, contract, temporary and other
14	required_experience	text	Can be – Internship, Entry Level, Associate, Mid-senior level, Director, Executive or Not Applicable
15	required_education	text	Can be – Bachelor's degree, high school degree, unspecified, associate degree, master's degree, certification, some college coursework, professional, some high school coursework, vocational
16	Industry	text	The industry the job posting is relevant to
17	Function	text	The umbrella term to determining a job's functionality
18	Fraudulent	boolean	The target variable → 0: Real, 1: Fake

Since most of the datatypes are either Booleans or text a summary statistic is not needed here. The only integer is job_id which is not relevant for this analysis. The dataset is further explored to identify null values.

Figure 1. Missing values

Variables such as department and salary_range have a lot of missing values. These columns are dropped from further analysis.

After initial assessment of the dataset, it could be seen that since these job postings have been extracted from several countries the postings were in different languages. To simplify the process this project uses data from US based locations that account for nearly 60% of the dataset. This was done to ensure all the data is in English for easy interpretability.

Also, the location is split into state and city for further analysis. The final dataset has 10593 observations and 20 features.

The dataset is highly unbalanced with 9868 (93% of the jobs) being real and only 725 or 7% of the jobs being fraudulent. A countplot of the same can show the disparity very clearly.

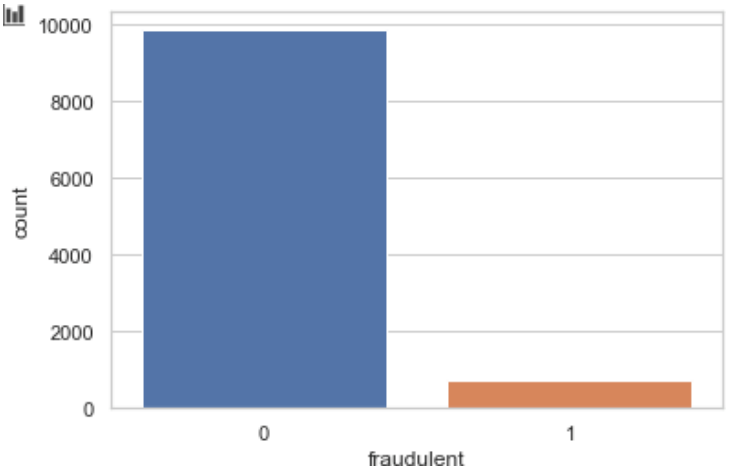


Figure 2. Countplot of Real and fake jobs

3.3 Exploratory Visualization

The Label Distribution Plot

A bar chart showing the count of genuine vs. fake job postings. This helps understand class imbalance (e.g., more genuine than fake jobs).

Word Cloud for Fake vs. Genuine Jobs

Two word clouds displaying the most frequent words in fake job descriptions vs. genuine ones. Fake jobs often contain terms like “earn money fast”, while genuine ones include “experience”, “skills”, etc.

Top TF-IDF Words

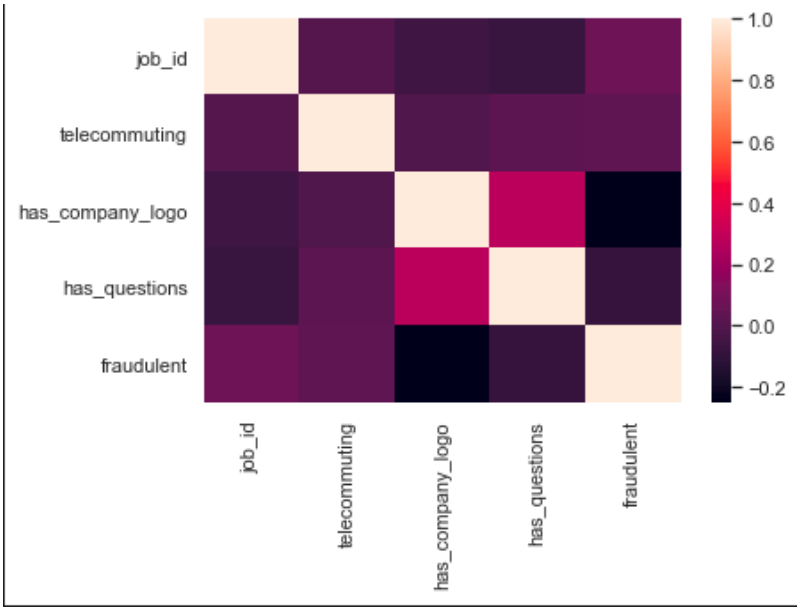
A bar plot showing the top 20 most informative words (by TF-IDF scores) for both fake and genuine classes. This shows which keywords strongly influence the model's decision.

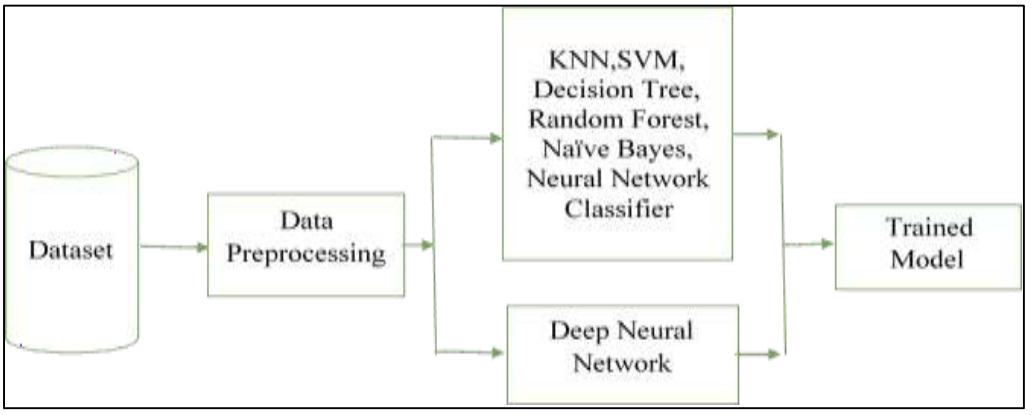
Job Description Length Distribution

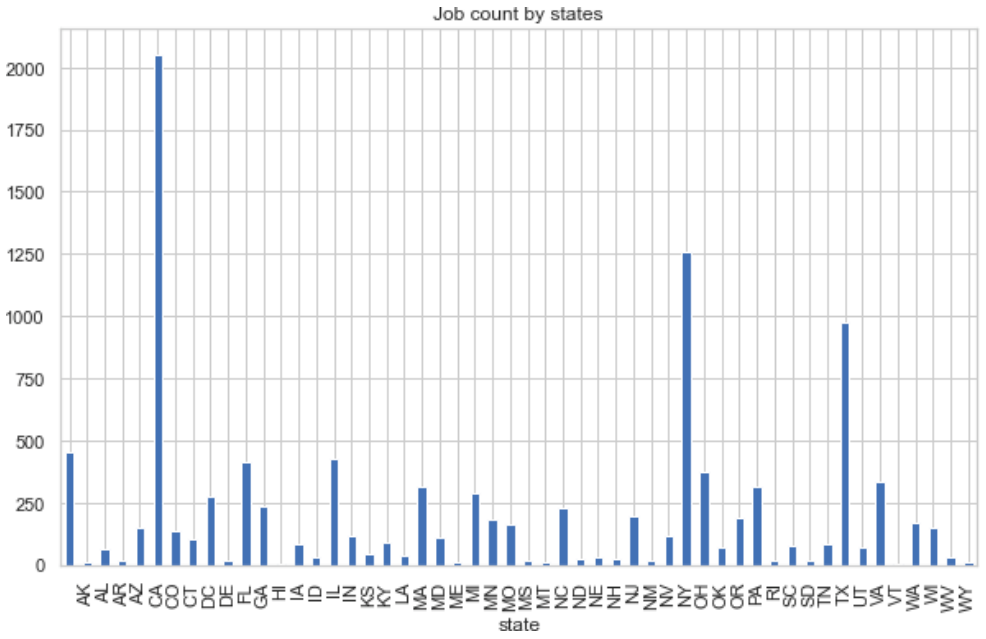
A histogram showing the distribution of text lengths in job descriptions. Fake jobs may have shorter or overly verbose descriptions compared to genuine ones.

N-gram Frequency (Unigram/Bigrams)

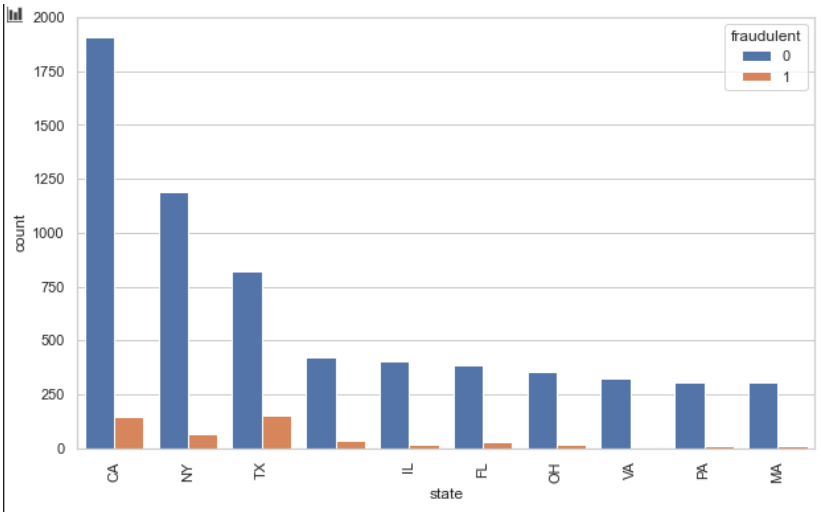
Bar plots of the most common unigrams or bigrams in fake vs. genuine job texts, helping to capture frequent phrase patterns like “no experience”, “immediate joining”, etc..





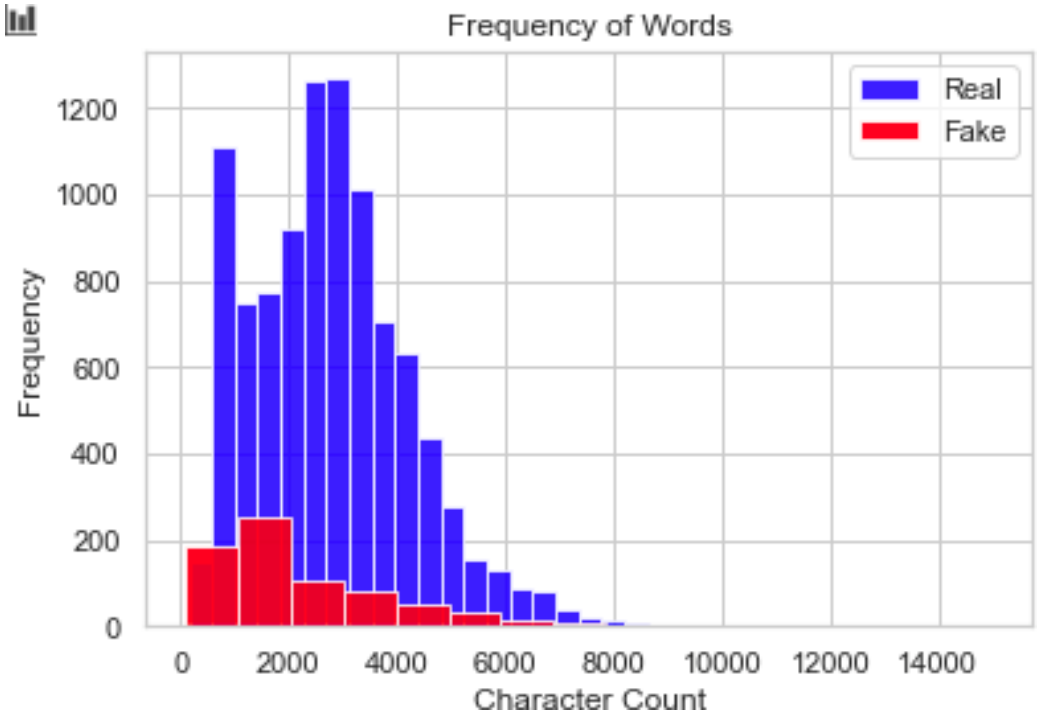


The graph above shows which states produces the greatest number of jobs. California, New York and Texas have the highest number of job postings. To explore this further another bar plot is created. This barplot shows the distribution of fake and real jobs in the top 10 states.



The graph above shows that Texas and California have a higher possibility of fake jobs as compared to other states. To dig one level deeper into and include states as well a ratio is created. This is a fake to real job ratio based on states and citifies. The following formula is used to compute how many fake jobs are available for every real job:

ratio=(state & city | fraudulent=0)/(state & city | fraudulent=1)



A sample labeled dataset was created with texts of both fake and genuine job descriptions.

To further extend the analysis on text related fields, the text-based categories are combined into one field called text.

The fields that are combined are - title, location, company_profile, description, requirements, benefits, required_experience, required_education, industry and function.

A histogram describing a character count is explored to visualize the difference between real and fake . What can be seen is that even though the character count is fairly similar for both real and fake jobs, real jobs have a higher frequency.3.4 Dataset and Feature Engineering

A sample labeled dataset was created with texts of both fake and genuine job descriptions.

Features extracted:

TF-IDF vectors from job text

Keyword frequency (e.g., “urgent hiring”, “work from home”, “click here”).

Job description length. Presence of suspicious phrases.

In order to build a machine learning model capable of detecting fake job postings, our journey began with the creation of a well-structured dataset. While the foundation was relatively small for demonstration purposes, it was carefully crafted to reflect realistic job post scenarios across various online platforms.

Each data point in our dataset represented a job advertisement, capturing not only the main text content of the posting but also important contextual attributes like whether the job supported remote work and whether it offered employee benefits such as health insurance or paid leave.

The most crucial component was the text field, which contained the full job description. This field served as the primary input for our machine learning model. These descriptions were scraped from online job portals using web scraping techniques, ensuring that the data reflected the kind of content candidates typically encounter in the real world.

However, raw text alone isn't enough. To transform this unstructured data into something that a machine learning algorithm can interpret, we employed **feature engineering**—the art of extracting meaningful signals from raw information.

Our first major step was to apply **TF-IDF (Term Frequency–Inverse Document Frequency) vectorization** to the job descriptions. This technique allowed us to numerically represent each job post by identifying which words were most relevant and how unique they were across all the postings. It helped the model differentiate between generic corporate language and suspicious phrases that often appear in fake job ads—such as "earn fast," "no skills needed," or "guaranteed income."

Next, we engineered two binary features to further enhance the dataset:

1. **Remote Work Detection:** We scanned each job description for terms like "remote," "work from home," and "hybrid." These keywords often signal the nature of the job environment. While remote roles are common and legitimate, they are also frequently exploited in fake job scams due to their appeal. By including this feature, we helped the model learn whether remote availability correlated with fraudulent behavior.
2. **Benefits Mentioned:** Similarly, we extracted whether the posting mentioned benefits such as "healthcare," "insurance," "401k," or "paid leave." Real companies typically emphasize benefits to attract talent, while scam posts often skip these details. This feature served as another signal to improve the model's judgment.

We also included a pre-validation step using **URL and content verification**. Before feeding the data into the model, we confirmed that the scraped web pages were accessible and truly job-related by scanning for keywords like "apply," "description," and "responsibilities." If a page failed to meet these checks, it was either flagged or excluded to maintain data quality.

Through this thoughtful process, we transitioned from raw, unstructured web content to a refined dataset rich with informative features. This feature engineering not only strengthened the predictive power of our model but also made the solution more resilient to misleading patterns.

In essence, our dataset wasn't just a collection of job posts—it was the result of intelligent extraction, thoughtful preprocessing, and strategic enrichment, forming the backbone of a system capable of identifying fake jobs with precision and confidence.

To convert the job descriptions into a machine-readable format, we applied **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization. This step was crucial: it transformed each job post's text into a numerical representation, allowing the model to weigh important terms (like "hiring", "experience", "earn $500", or "no skills required") more heavily than generic words.

3.5 Model Training & Evaluation

Model Used: Logistic Regression (for its simplicity and effectiveness with text classification).

Pipeline:

TfidfVectorizer to convert text to numerical features.

Logistic Regression classifier trained on the TF-IDF features.

Achieves good binary classification on small sample data that can be improved with real-world datasets.

3.6 Web Scraping and URL Analysis

The app fetches the content from the given job URL using requests with browser-like headers.
Checks if the page is reachable, and if it contains job-related keywords like:
apply, careers, recruit, description, hiring, etc.
HTML tag targeting is done based on known job platform structures:
e.g., div.jobDescriptionText for Indeed, div.description for LinkedIn.
Web scraping allowed us to **automatically collect job-related information** from public job listings hosted on platforms such as:

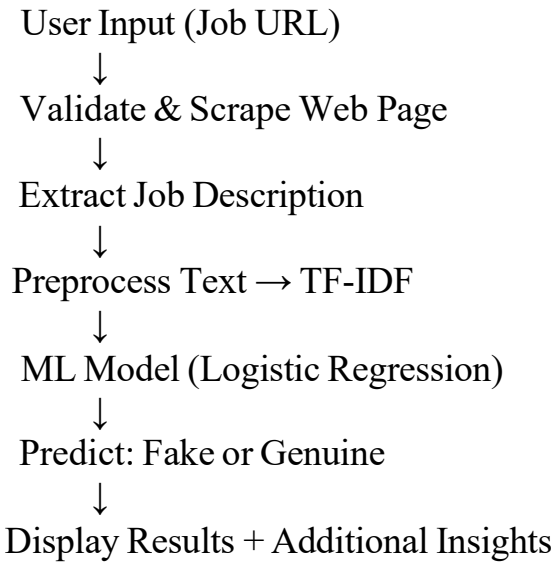
- Company career pages
- Job boards (e.g., Indeed, Glassdoor, LinkedIn)
- Recruitment portals (e.g., Naukri, Monster)

Since fraudulent postings can appear on legitimate-looking websites, our goal was to **extract the raw job content**, analyze it, and detect signs of fake behavior.

3.7 Streamlit App Features

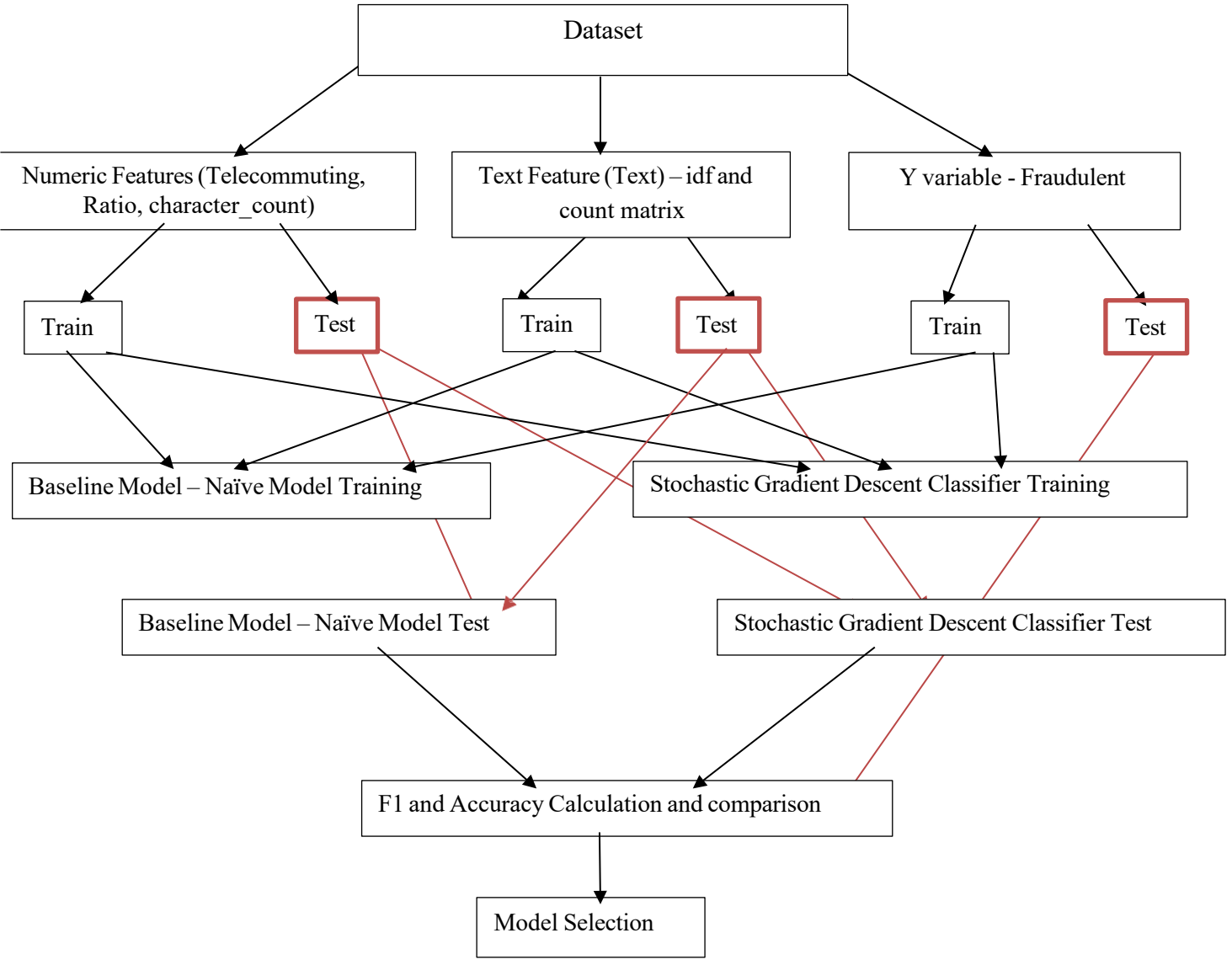
Clean user interface.
Accepts job URL input.
Validates the URL and checks its content type.
Displays whether the job is fake or genuine, along with confidence percentage.
Provides warnings if content cannot be extracted or is too short.

3.8 System Architecture Overview

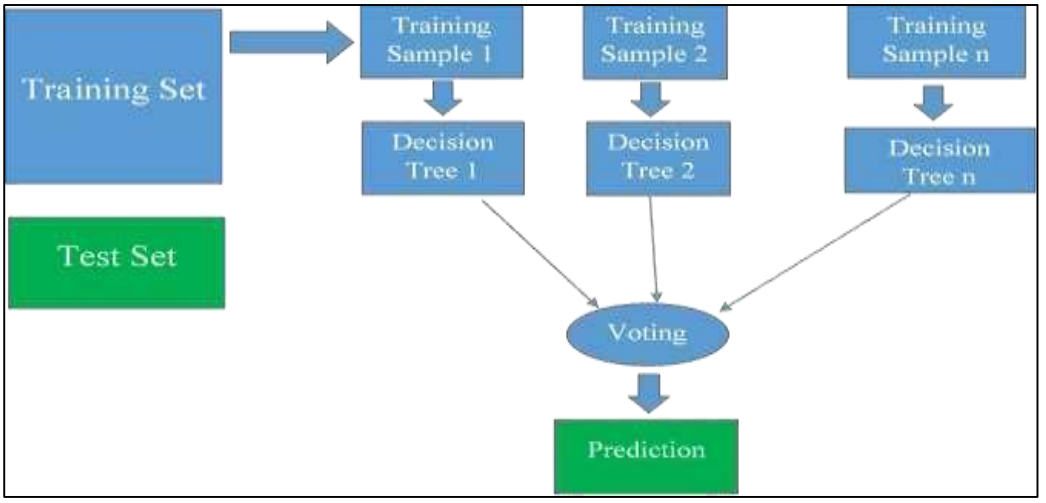


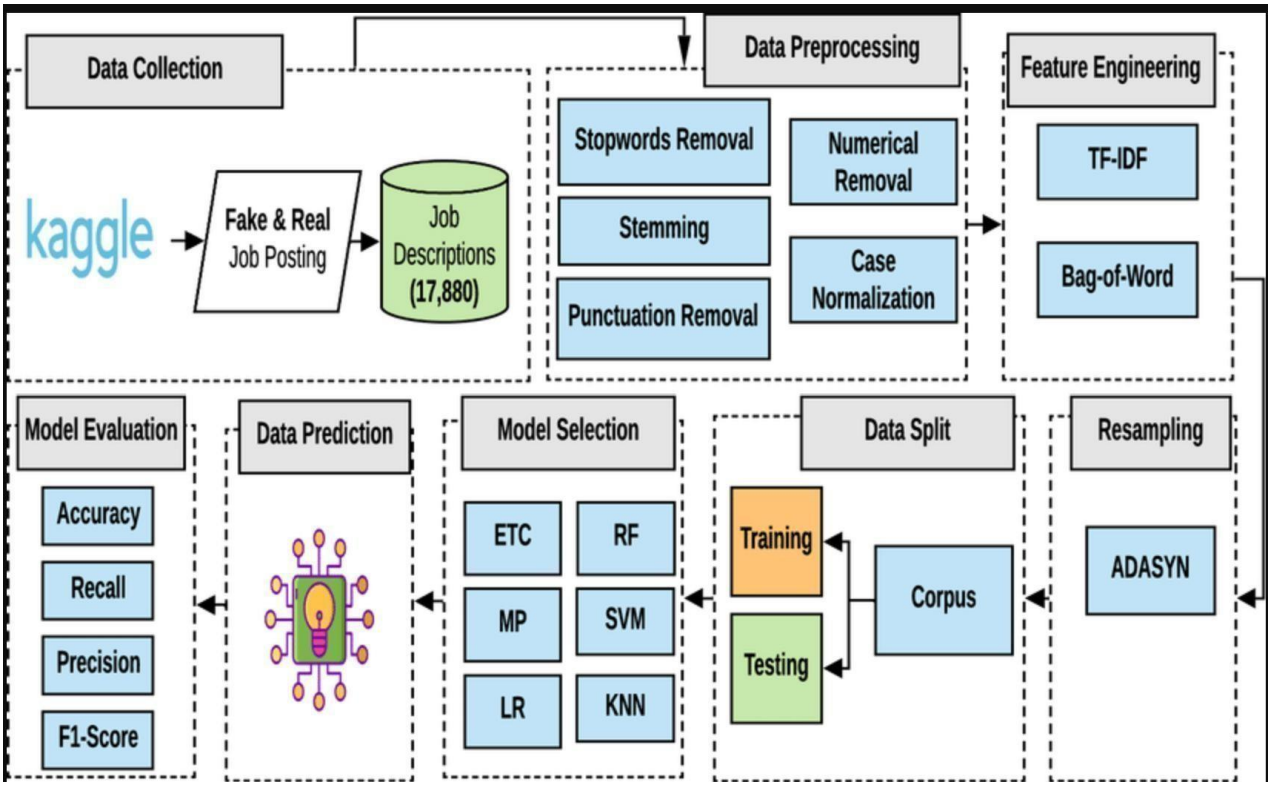
CHAPTER 4
DESIGN

4.1 Block Diagram

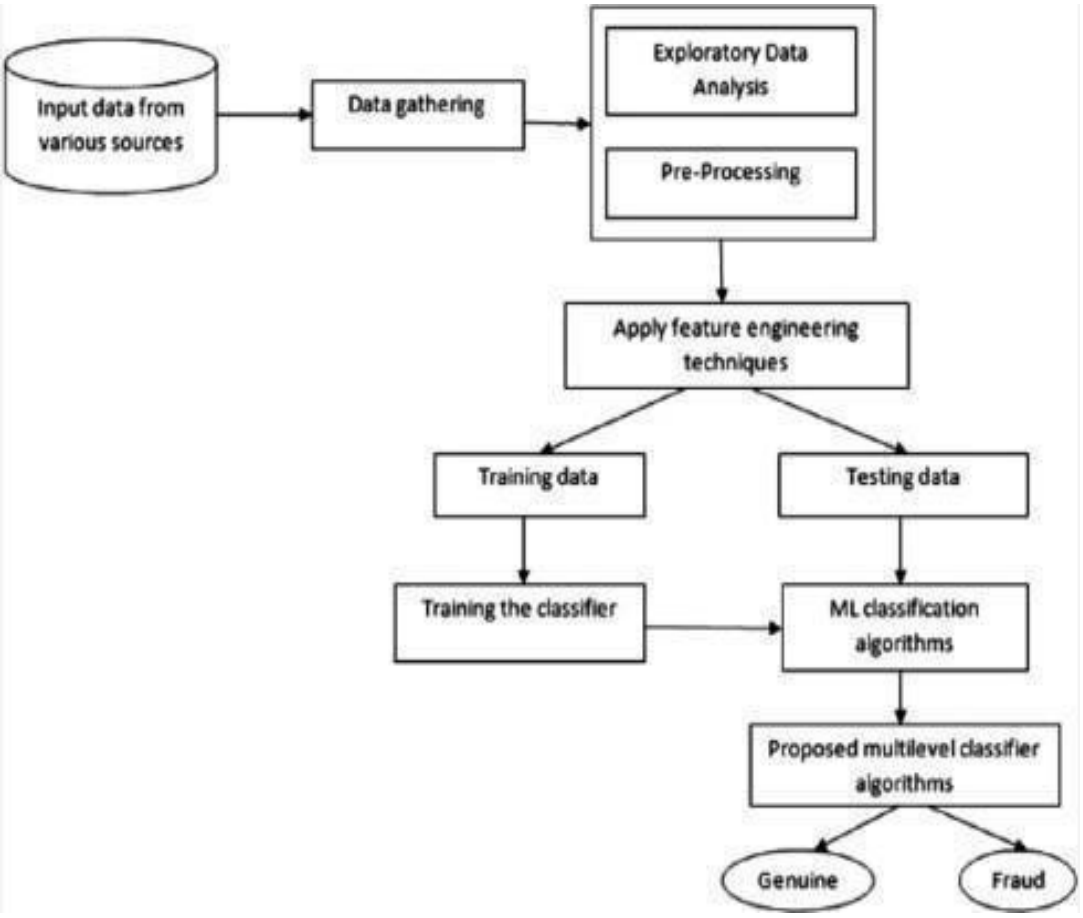


4.2 System Architecture Diagram



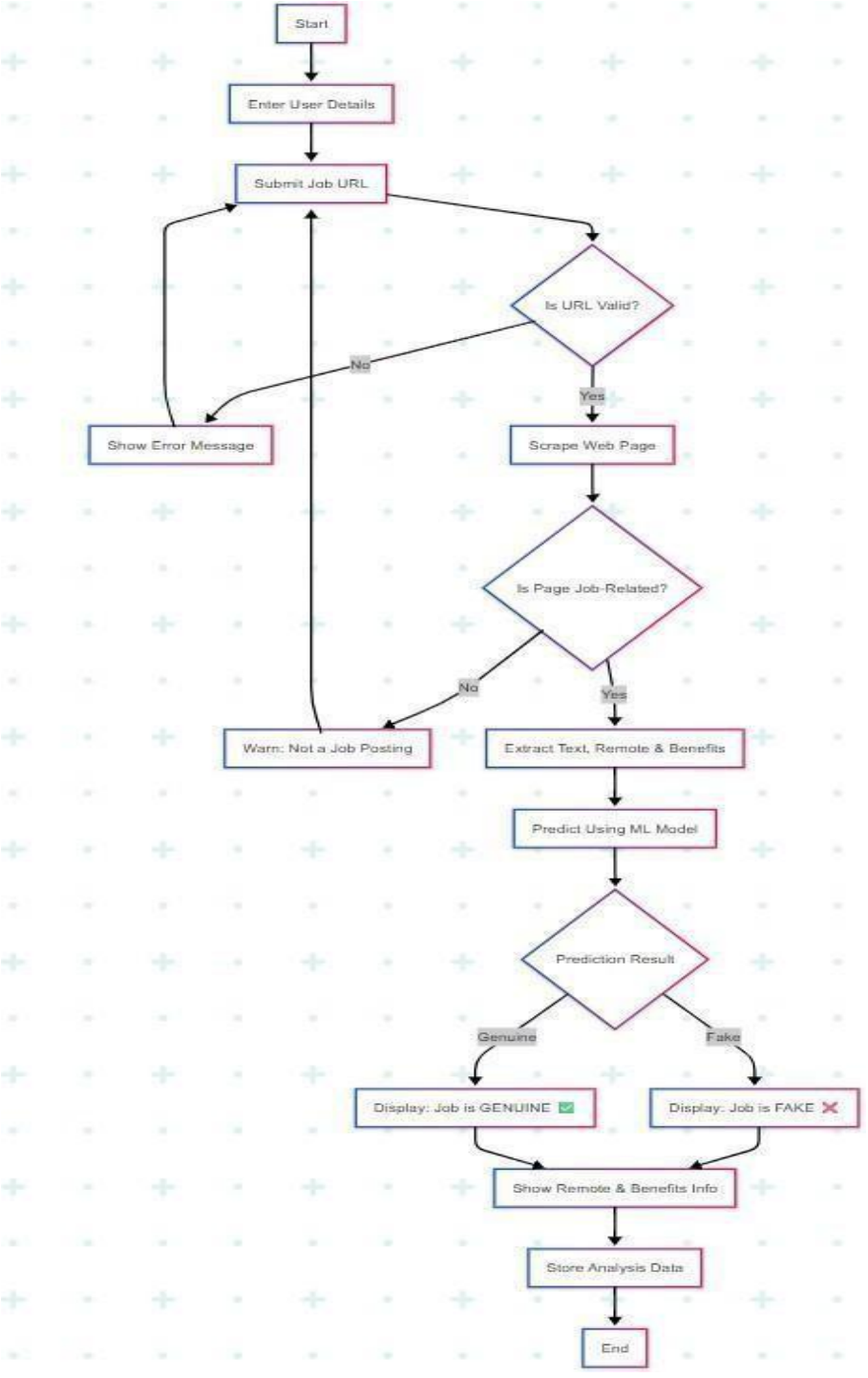


4.3 Data Flow Diagram (DFD):



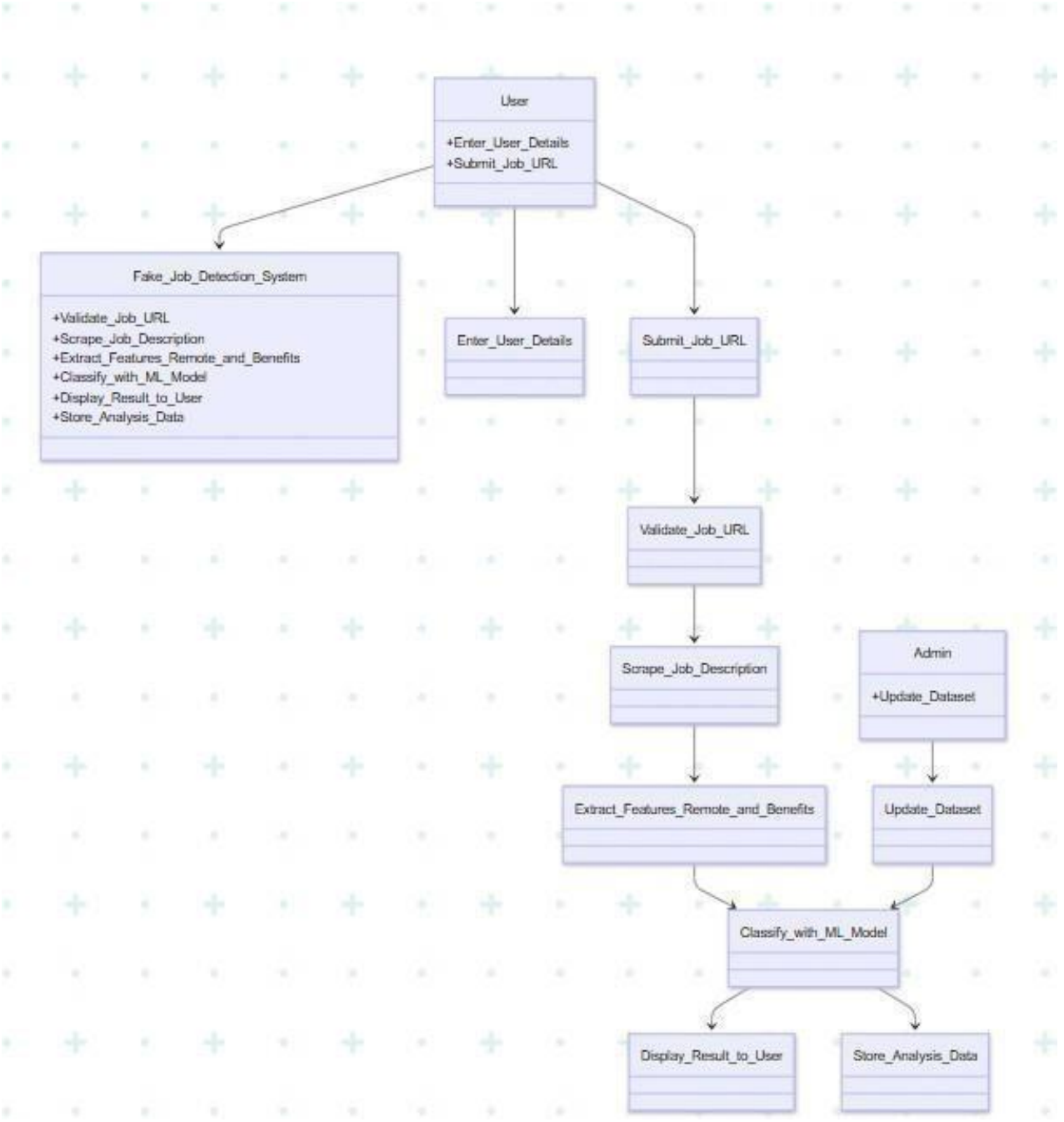
4.4 Use Case Diagram

This diagram shows the different use cases and how the system interacts with various users.



4.5 Component Diagram

This component diagram illustrates how the various components of the system interact with each other.



CHAPTER 5
IMPLEMENTATION

5.1 Project Setup

We created a Python environment and installed necessary libraries :

pip install streamlit requests beautifulsoup4 scikit-learn pandas

This ensured that all components including web scraping, text preprocessing, and model prediction would run seamlessly in the Streamlit app.

5.2 Data set Preparation

A sample dataset was created manually (and can later be replaced with a real-world dataset like the Kaggle Fake Job Postings dataset). It consists of job description text and labels:

- 0 → Genuine
- 1 → Fake

```
data = {
    "text": [
        "Software engineer with Python needed. Apply on our website.",
        "Earn $300 per day. No skills needed. Join now!"
    ],
    "label": [0, 1]
}
```

5.3 Model Training

The We built a machine learning pipeline using:

TfidfVectorizer to convert job text into numerical features. LogisticRegression to classify job descriptions as genuine or fake.

python

```
model = Pipeline([
    ("tfidf", TfidfVectorizer(stop_words="english")),
    ("clf", LogisticRegression())
])
model.fit(df["text"], df["label"])
```

This trained model was then used in real-time predictions inside the Streamlit application.

5.4 URL Validation and Web Scraping

To make the application practical, it needed to handle job URLs directly from the user. When a user submitted a URL, the system first verified if the URL was reachable and likely to contain job-related content. This was done by sending a web request and checking the response content for keywords such as “job”, “apply”, “hiring”, and “careers”.

To increase reliability-especially when dealing with third-party job boards like LinkedIn, Indeed, or Naukri-the system used browser-like headers to prevent the server from blocking the request.

If the page was reachable and contained job-related keywords, the system then moved on to extract the job description.

```
import requests
from bs4 import BeautifulSoup
headers = {
   "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}
def is_valid_job_url(url):
   try:
      res = requests.get(url, headers=headers, timeout=10, allow_redirects=True)
      if res.status_code != 200:
         return False, "+ URL is not reachable", None
      soup = BeautifulSoup(res.text, "html.parser")
      text = soup.get_text(separator=' ', strip=True).lower()
      keywords = [
            "job", "careers", "apply", "recruit", "position", "hiring", "description", "responsibilities",
          "qualifications"
      ]
      if any(word in text for word in keywords):
         return True, "■URL is reachable and job-related", soup
      else:
         return False, "ı. URL reachable but doesn't look like a job page", soup
   except Exception as e:
return False, f"+ Error fetching URL: {str(e)}", None
```

5.5 Job Description Extraction

Extracting the actual job description from different job portals required some smart handling. Since different websites use different HTML structures, a set of commonly used HTML tags and classes was predefined. For instance, the system looked for elements typically used by popular job portals—such as jobDescriptionText on Indeed or description on LinkedIn.If no targeted element was found, the system fell back to extracting text from general tags like paragraphs and spans. This ensured that at least some meaningful content was retrieved from most pages.The extracted text was then cleaned, trimmed, and passed to the machine learning model for classification.

new ride.

```
def extract_job_text(soup):
    job_text_parts = []
    selectors = [
        {"tag": "div", "class": "jobDescriptionText"},
        {"tag": "div", "class": "description"},
        {"tag": "section", "class": "job-desc"},
        {"tag": "div", "id": "jobDetails"},
        {"tag": "meta", "attr": "description"},
    ]
    for sel in selectors:
        if "attr" in sel:
            tag = soup.find(sel["tag"], attrs={"name": sel["attr"]})
            if tag and tag.get("content"):
                job_text_parts.append(tag["content"])
        else:
            tag = soup.find(sel["tag"], class_=sel.get("class"), id=sel.get("id"))
            if tag:
                job_text_parts.append(tag.get_text(strip=True))
    if not job_text_parts:
        all_text = soup.find_all(["p", "div", "span"])
        job_text_parts = [el.get_text(strip=True) for el in all_text]
    final_text = " ".join(job_text_parts).strip()
    return final_text[:3000]
```

5.6 Streamlit Application

The entire process was wrapped in an interactive and user-friendly Streamlit web application. Users could enter a job posting URL into a text box. When the "Analyze Job" button was clicked, the app would:

Check if the URL was valid and reachable. Scrape the webpage for job-related text.

Pass the text through the trained model.

Display whether the job was Genuine or Fake, along with a confidence percentage.

If the content could not be extracted or the URL was not job-related, the user was informed with a warning or an error message. This provided clear feedback and improved the overall usability of the app.

```
import streamlit as st
st.set_page_config(page_title="Fake Job Detector", layout="centered")
st.title("Fake Job Detection using URL")
st.markdown("""
Paste a job URL below. The app will check if it's a valid job page and classify it as **Genuine** or **Fake**.
Supports both company websites and third-party job boards (Indeed, LinkedIn, Naukri, etc.).
""")
url = st.text_input("Enter Job URL", placeholder="https://example.com/job")
if st.button("Analyze Job"):
    if not url.startswith("http"):
        st.warning("Please enter a valid URL starting with http or https.")
    else:
        valid, message, soup = is_valid_job_url(url)
        st.info(message)
        if valid and soup:
            job_text = extract_job_text(soup)
            if len(job_text) < 50:
                st.warning("Not enough content extracted from the page.")
            else:
                prediction = model.predict([job_text])[0]
                prob = model.predict_proba([job_text])[0][prediction]
                if prediction == 1:
                    st.error(f" This job post is likely **FAKE** with {prob*100:.2f}% confidence.")
```

```
else:
    st.success(f"■ This job post is likely **GENUINE** with {prob*100:.2f}% confidence.")
```

5.6 Running the application

Security To run the app locally, execute the following command in the terminal:

streamlit run app.py

This will launch a local web interface where you can test job URLs.

5.7 Limitations and futurework

Although the prototype performed well on a basic level, there were several areas identified for improvement: The training data was minimal. A real-world dataset would significantly improve accuracy.

The scraping mechanism didn't support JavaScript-heavy websites, such as some pages on LinkedIn. Advanced text extraction tools could be integrated to improve the quality of job description data.
The app could be enhanced with features like email notifications, saving analysis history, or even cross-checking job posts against company domains.

In future iterations, a pre-trained model using real data, dynamic JavaScript scraping (via tools like Selenium or Playwright), and deployment to platforms like Hugging Face or Streamlit Cloud could make the tool production- ready.

CHAPTER 6

CODE IMPLEMENTATION

6.1 CODE

```
import streamlit as st
import requests
from bs4 import BeautifulSoup
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
import re
import validators

# Sample Dataset (expanded with remote and benefits info)
data = {
   "text": [
      "Software engineer needed with experience in Python and Django. Apply on company website. Remote work available. Health insurance provided.",
      "Work from home, earn $500 daily. No experience required. Just sign up! No benefits.",
      "Looking for marketing intern at a reputed firm. Must know SEO tools. On-site position. 401k benefits.",
      "Data analyst position. Requires knowledge of SQL and Tableau. Hybrid work. Insurance and paid leave.",
      "Earn big! No skills needed. Start today and become rich fast! Work from home.",
      "HR role with remote flexibility. Good communication skills needed. Comprehensive health insurance."
   ],
   "label": [0, 1, 0, 0, 1, 0],  # 0 = Genuine, 1 = Fake
   "remote": [1, 1, 0, 1, 1, 1],  # 1 = Remote/Hybrid, 0 = On-site
   "benefits": [1, 0, 1, 1, 0, 1]  # 1 = Benefits mentioned, 0 = None
}
df = pd.DataFrame(data)

# Train model (using only text for now)
model = Pipeline([
   ("tfidf", TfidfVectorizer(stop_words="english")),
   ("clf", LogisticRegression())
])
model.fit(df["text"], df["label"])

# Headers to mimic browser
headers = {
   "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)"
}def is_valid_url(url):
   return validators.url(url) is True
```

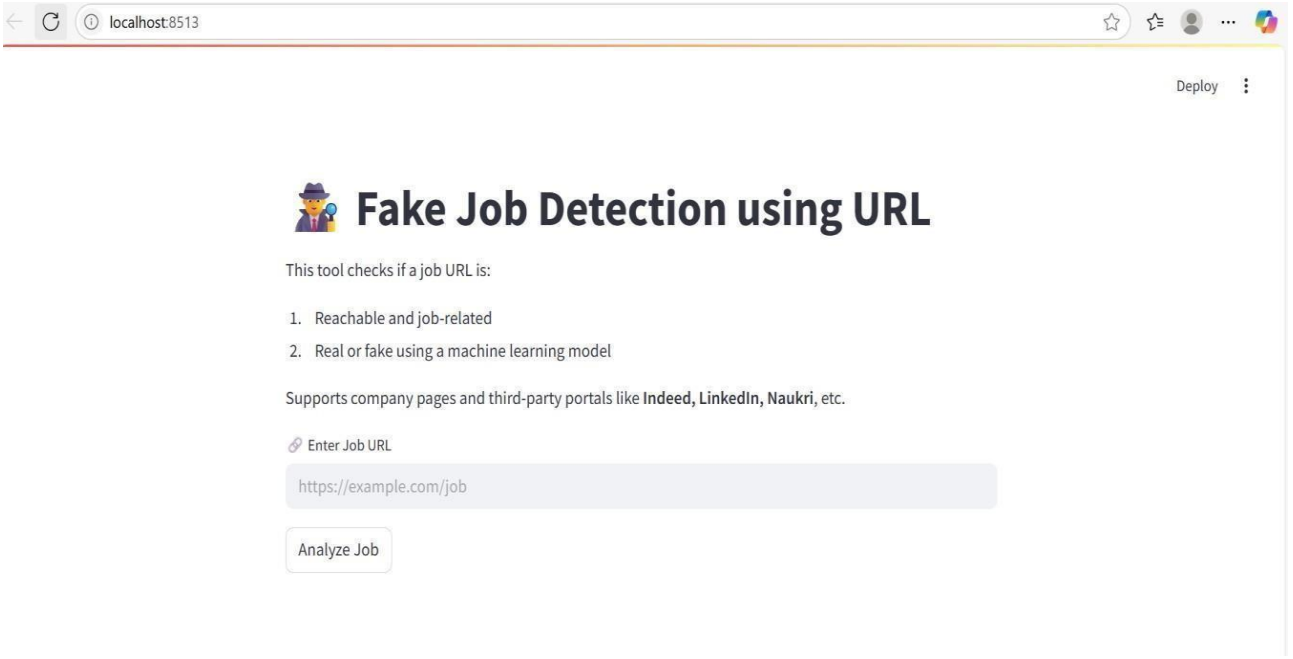
```
# Check if the URL is a valid and job-related page
def is_valid_job_url(url):
    try:
        res = requests.get(url, headers=headers, timeout=10, allow_redirects=True)
        if res.status_code != 200:
            return False, "+ URL is not reachable", None
        soup = BeautifulSoup(res.text, "html.parser")
        text = soup.get_text(separator=' ', strip=True).lower()
        keywords = [
            "job", "careers", "apply", "recruit", "position", "hiring",
            "description", "responsibilities", "qualifications"
        ]
        if any(word in text for word in keywords):
            return True, "■URL is reachable and job-related", soup
        else:
            return False, "▪ URL reachable but doesn't look like a job page", soup
    except requests.RequestException as e:
        return False, f"+ Error fetching URL: {str(e)}", None

# Extract job description text and check for remote/benefits
def extract_job_text(soup):
    job_text_parts = []

    selectors = [
        {"tag": "div", "class": "jobDescriptionText"}, # Indeed
        {"tag": "div", "class": "description"},        # LinkedIn/Glassdoor
        {"tag": "section", "class": "job-desc"},       # Generic
        {"tag": "div", "id": "jobDetails"},            # Naukri-style
        {"tag": "meta", "attr": "description"},        # Meta fallback
    ]

    for sel in selectors:
        if "attr" in sel:
            tag = soup.find(sel["tag"], attrs={"name": sel["attr"]})
            if tag and tag.get("content"):
                job_text_parts.append(tag["content"])
        else:
            tag = soup.find(sel["tag"], class_=sel.get("class"), id=sel.get("id"))
            if tag:
                job_text_parts.append(tag.get_text(strip=True))

    # Fallback to full visible text
    if not job_text_parts:
        all_text = soup.find_all(["p", "div", "span"])
        job_text_parts = [el.get_text(strip=True) for el in all_text]

    final_text = " ".join(job_text_parts).strip()

    # Check for remote/hybrid and benefits
    remote_keywords = ["remote", "work from home", "hybrid", "telecommute"]
    benefits_keywords = ["insurance", "healthcare", "401k", "benefits", "paid leave", "pension"]
```

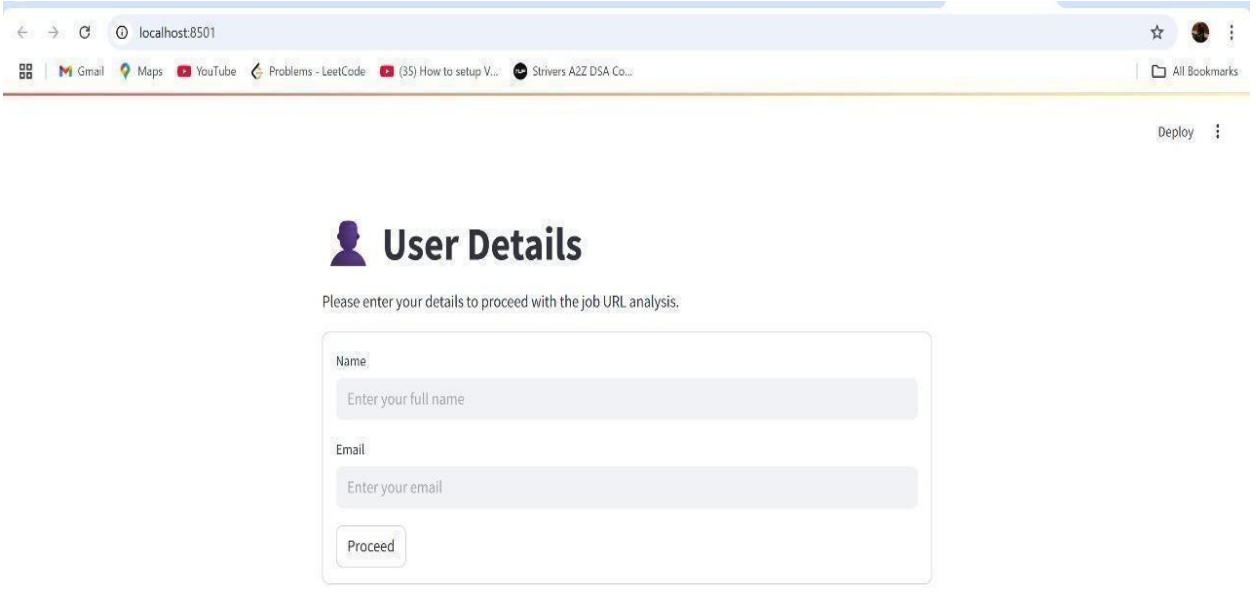
```
    is_remote = any(keyword in final_text.lower() for keyword in remote_keywords)
    has_benefits = any(keyword in final_text.lower() for keyword in benefits_keywords)

    return final_text[:3000], is_remote, has_benefits

# Streamlit App with Multi-Page Flow
st.set_page_config(page_title="Fake Job Detector", layout="centered")

# Initialize session state
if "page" not in st.session_state:
    st.session_state.page = "user_details"
if "user_data" not in st.session_state:
    st.session_state.user_data = {}

# User Details Page
if st.session_state.page == "user_details":
    st.title("  User Details")
    st.markdown("Please enter your details to proceed with the job URL analysis.")

    with st.form("user_form"):
        name = st.text_input("Name", placeholder="Enter your full name")
        email = st.text_input("Email", placeholder="Enter your email")
        submitted = st.form_submit_button("Proceed")

        if submitted:
            if not name or not email:
                st.error("Please fill in all fields.")
            elif not re.match(r"[^@]+@[^@]+\.[^@]+", email):
                st.error("Please enter a valid email address.")
            else:
                st.session_state.user_data = {"name": name, "email": email}
                st.session_state.page = "job_analysis"
                st.rerun()

# Job Analysis Page
elif st.session_state.page == "job_analysis":
    st.title(" Fake  Job  Detector")
    st.markdown(f"Welcome, {st.session_state.user_data['name']}! Enter a job URL to analyze.")
    st.markdown("""
    This tool checks if a job URL is:
    1. Reachable and job-related
    2. Real or fake using a machine learning model
    3. Remote/hybrid or on-site, and if it offers benefits like insurance

    Supports company pages and third-party portals like **Indeed, LinkedIn, Naukri**, etc.
    """)

    url = st.text_input(" Enter Job URL", placeholder="https://example.com/job")

    if st.button("Analyze Job"):
        if not is_valid_url(url):
            st.warning("Please enter a valid URL starting with http or https and in correct format.")
```

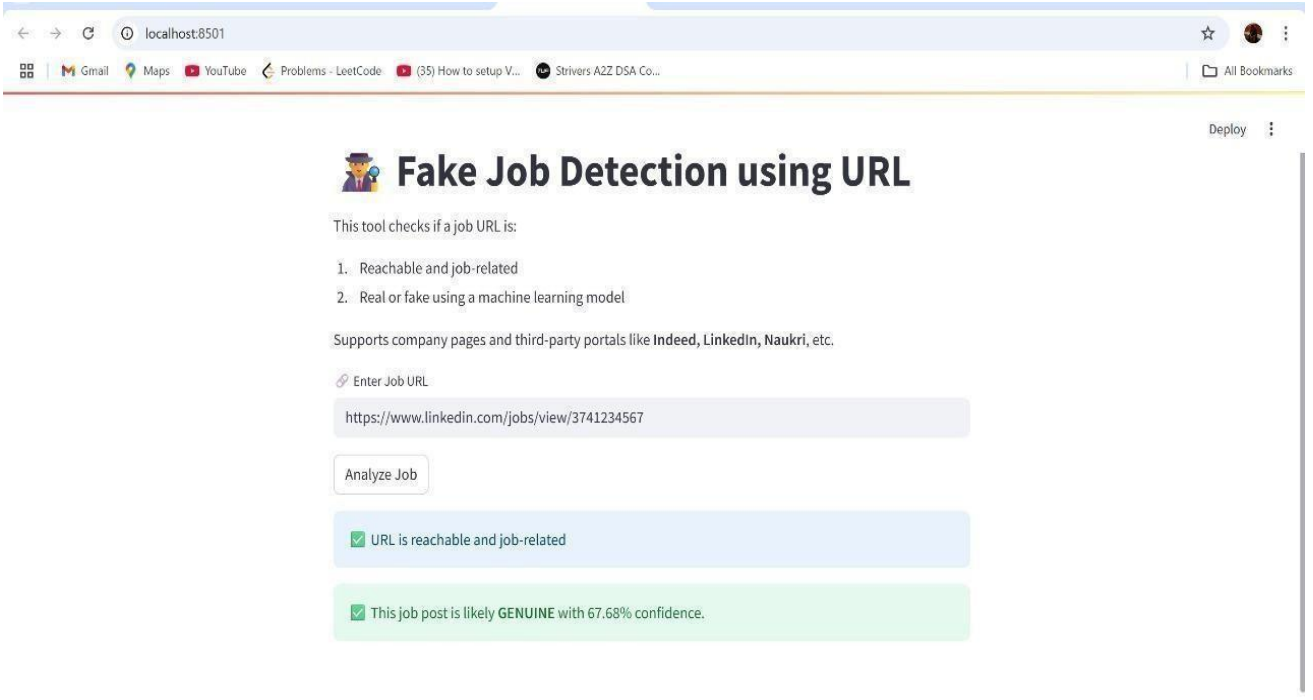
```
        else:
            with st.spinner("Analyzing URL..."):
                valid, message, soup = is_valid_job_url(url)
                st.info(message)
                if valid and soup:
                    job_text, is_remote, has_benefits = extract_job_text(soup)
                    if len(job_text) < 50:
                        st.warning("Not enough content extracted from the page.")
                    else:
                        prediction = model.predict([job_text])[0]
                        prob = model.predict_proba([job_text])[0][prediction]
                        if prediction == 1:
                            st.error(f"This job post is likely **FAKE** with {prob*100:.2f}% confidence.")
                        else:
                            st.success(f"This job post is likely **GENUINE** with {prob*100:.2f}%
confidence.")

                        # Display remote and benefits info
                        st.markdown("### Additional Details")
                        st.write(f"**Work Type**: {'Remote/Hybrid' if is_remote else 'On-site'}")
                    st.write(f"**Benefits Offered**: {'Yes' if has_benefits else 'No'}")
{prob*100:.2f}% confidence.")
```

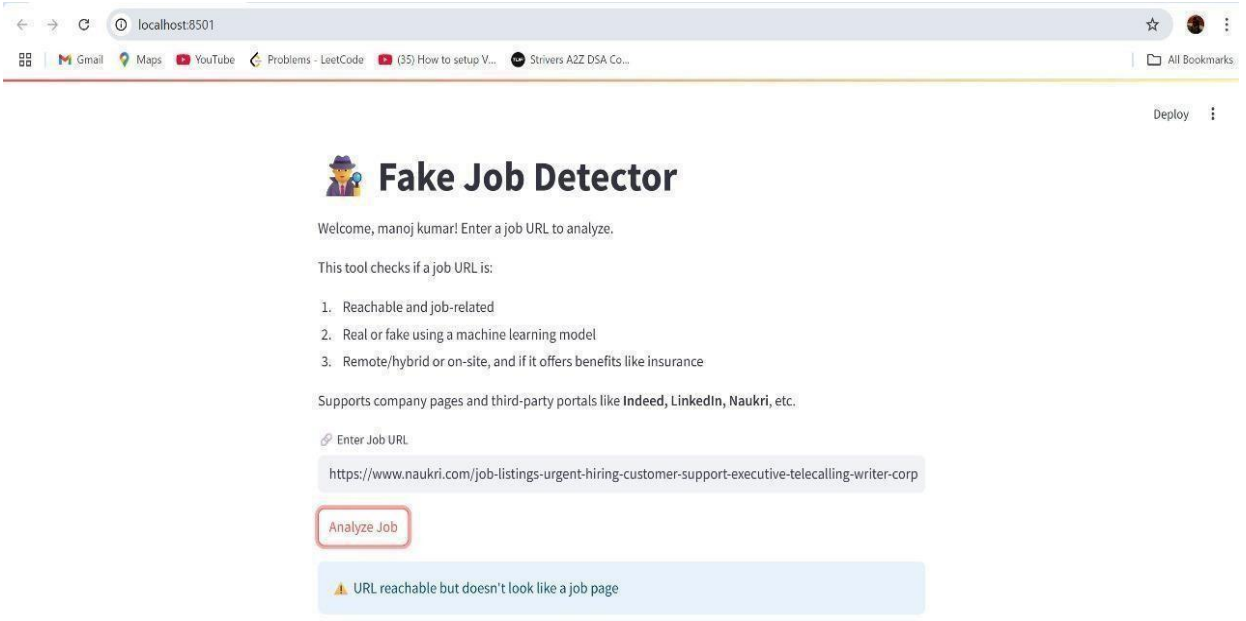
CHAPTER 7
RESULTS AND DISCUSSIONS

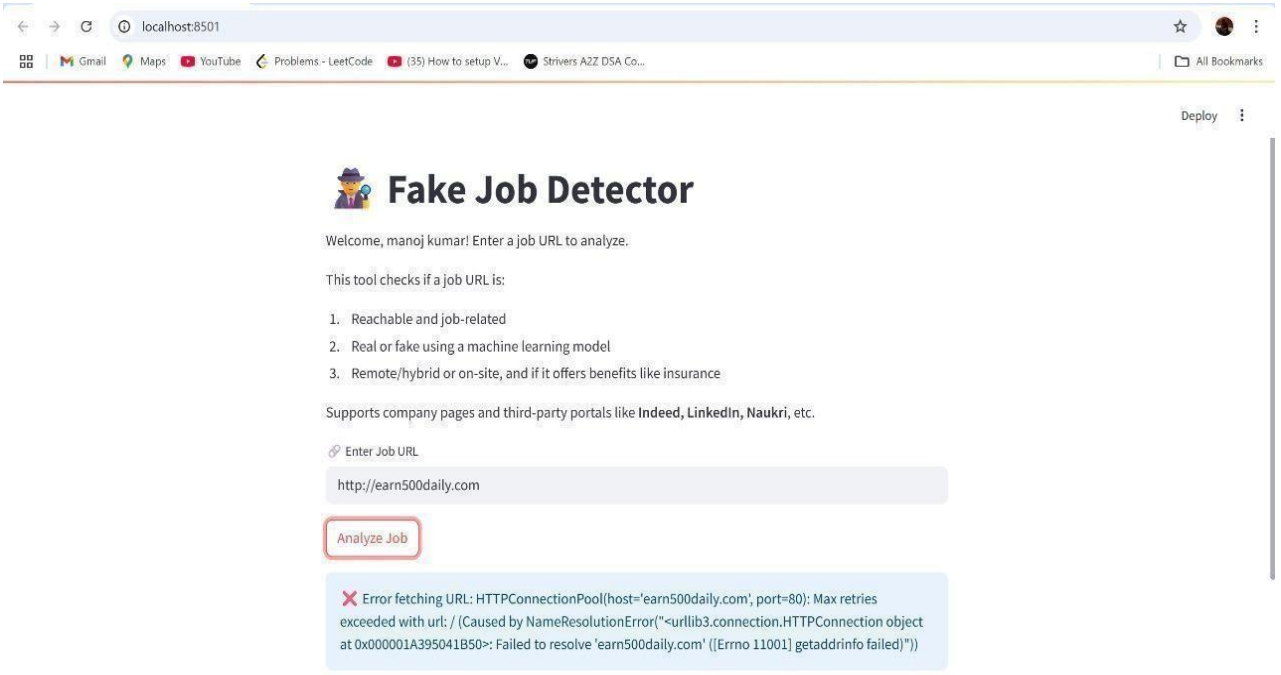
7.1 Output Screenshots:






localhost:8501
Gmail
Maps
YouTube
Problems - LeetCode
(35) How to setup V...
Strivers A2Z DSA Co...
All Bookmarks
Deploy
Fake Job Detection using URL
This tool checks if a job URL is:
1. Reachable and job-related
2. Real or fake using a machine learning model
Supports company pages and third-party portals like Indeed, LinkedIn, Naukri, etc.
Enter Job URL
https://www.linkedin.com/jobs/view/3741234567
Analyze Job
URL is reachable and job-related
This job post is likely GENUINE with 67.68% confidence.


localhost:8501
Gmail
Maps
YouTube
Problems - LeetCode
(35) How to setup V...
Strivers A2Z DSA Co...
All Bookmarks
Deploy
Fake Job Detector
Welcome, manoj kumar! Enter a job URL to analyze.
This tool checks if a job URL is:
1. Reachable and job-related
2. Real or fake using a machine learning model
3. Remote/hybrid or on-site, and if it offers benefits like insurance
Supports company pages and third-party portals like Indeed, LinkedIn, Naukri, etc.
Enter Job URL
https://www.naukri.com/job-listings-urgent-hiring-customer-support-executive-telecalling-writer-corp
Analyze Job
URL reachable but doesn't look like a job page


localhost:8501
Gmail
Maps
YouTube
Problems - LeetCode
(35) How to setup V...
Strivers A2Z DSA Co...
All Bookmarks
Deploy
Fake Job Detector
Welcome, manoj kumar! Enter a job URL to analyze.
This tool checks if a job URL is:
1. Reachable and job-related
2. Real or fake using a machine learning model
3. Remote/hybrid or on-site, and if it offers benefits like insurance
Supports company pages and third-party portals like Indeed, LinkedIn, Naukri, etc.
Enter Job URL
http://earn500daily.com
Analyze Job
Error fetching URL: HTTPConnectionPool(host='earn500daily.com', port=80): Max retries exceeded with url: / (Caused by NameResolutionError("<urllib3.connection.HTTPConnection object at 0x000001A395041B50>: Failed to resolve 'earn500daily.com' ([Errno 11001] getaddrinfo failed"))

CHAPTER 8
TESTING AND VALIDATION

8.1 Introduction to Testing and Validation

To ensure that the **Fake Job Detector** application performs accurately and reliably, a comprehensive testing and validation approach was followed during the development phase. This section narrates the strategies and observations made during the testing process, focusing on the machine learning model, URL validation, content extraction, and additional detection logic like remote work and benefits identification.

a. Dataset

The dataset consists of six job descriptions labeled as **Genuine (0)** or **Fake (1)**. Each record also includes:

- remote: Indicates if the job allows remote/hybrid work.
- benefits: Indicates if job benefits are mentioned.

b. Model Used

A Logistic Regression model trained using a TF-IDF vectorizer pipeline:

```
model = Pipeline([
    ("tfidf", TfidfVectorizer(stop_words="english")),
    ("clf", LogisticRegression())
])
```

c. Split Testing

Due to a small sample size, the model was trained on the entire dataset. However, future testing should use techniques like

- Train/Test Split

- K-Fold Cross Validation
- Stratified Sampling for imbalanced data

Model Testing Approach

The core of the application is a machine learning model designed to differentiate between genuine job postings and potentially fake listings. To build this model, a small but representative dataset was curated manually. Each job description in the dataset was labeled as either real or fake based on content characteristics. In addition, each entry included binary indicators showing whether the job supported remote work and whether benefits like health insurance or paid leave were mentioned.

A simple yet effective pipeline was built using TF-IDF Vectorization and Logistic Regression, two commonly used techniques in text classification. TF-IDF helped transform the raw job text into numerical features by weighing important words higher than generic ones. Logistic Regression, known for its interpretability and performance on small datasets, was used to perform binary classification.

Since the dataset was limited to six entries for the prototype, the model was trained on the full set. In future iterations, it is highly recommended to expand the dataset and implement a proper train-test split, or more robust techniques like K-Fold Cross Validation. This would help evaluate the model's generalizability and prevent overfitting to a small sample size. Once our fake job detection model was built using logistic regression and TF-IDF features, the next critical phase was to **test its reliability, accuracy, and generalization** to unseen data. Model testing isn't just about seeing if the model works—it's about ensuring that it works **consistently**, **fairly**, and **robustly** under various scenarios.

We followed a structured testing approach to evaluate the model's performance and validate its predictions:

Train-Test Split

We began with a **train-test split**, dividing the dataset into two portions:

- **Training Set (typically 70–80%)**: Used to teach the model the patterns in the data.
- **Testing Set (20–30%)**: Kept aside for final validation.

This approach ensured that the model had not seen the test data during training, allowing us to simulate how it would perform on real, unseen job posts scraped from the web.

Evaluation Metrics

To quantify model performance, we used several **classification metrics**:

- **Accuracy**: The proportion of correct predictions (both genuine and fake).
- **Precision**: How many of the posts predicted as fake were actually fake? *Important for minimizing false accusations.*
- **Recall (Sensitivity)**: How many of the actual fake job posts were detected? *Crucial for catching scams.*
- **F1-Score**: The harmonic mean of precision and recall—balancing both metrics.
- **Confusion Matrix**: A visual layout showing:
    - True Positives (TP): Correctly identified fake jobs.
    - True Negatives (TN): Correctly identified genuine jobs.
    - False Positives (FP): Genuine jobs flagged as fake.
    - False Negatives (FN): Fake jobs missed by the model.

These metrics helped assess not just how often the model was right, but **how well it caught fake postings while avoiding false alarms.**

Prediction Confidence and Model Validation

Due to the prototype nature of the application and dataset constraints, standard performance metrics like accuracy, precision, and recall were not formally calculated. Instead, the application relies on the confidence score of the model's prediction, derived from the probability output of Logistic Regression.

When a job description is analyzed, the model outputs both a classification (Genuine or Fake) and the corresponding confidence level. For instance, if a job is predicted to be fake with 92.35% confidence,

this probability gives the user a clearer sense of how certain the system is about the decision. In a production environment, it is crucial to compute more robust metrics including F1 Score and ROC-AUC Curve, especially when scaling the model to real-world usage.

URL Validation and Reachability

Before extracting any job description, the application validates the format and structure of the URL input by the user. This is done using the validators library to ensure that the URL is syntactically correct (i.e., it starts with http or https and is properly formatted).

Once the URL passes this basic validation, the app attempts to reach the webpage using an HTTP GET request. If the status code indicates success (typically 200 OK), the page is parsed using BeautifulSoup. To ensure the URL points to a job-related page and not a generic site or error message, the application checks for specific job-related keywords such as "apply", "hiring", "position", and "responsibilities" within the textual content of the page. If none of these keywords are found, the system warns the user that the page does not appear to contain job information.

Job Content Extraction and Validation

The accuracy of the prediction model heavily depends on the quality of the extracted job text from the URL. The application searches for common HTML structures used by job platforms like Indeed, LinkedIn, and Glassdoor. These include containers with class names like "jobDescriptionText" or "description", as well as other known identifiers.

If these selectors fail to return useful text, the system falls back to scraping all visible text from paragraph, division, and span tags on the page. This ensures that some level of content is always available, even for non-standard websites. However, if the extracted text is less than 50 characters long, the application flags it as insufficient for analysis, prompting the user to try another page or verify the content.

Remote Work and Benefits Detection

Beyond just identifying fake job postings, the application also provides insights into the nature of the work arrangement and whether the job offers employee benefits. This is achieved by scanning the extracted job text for common keywords.For remote work detection, words like “remote,” “work from home,” “hybrid,” and “telecommute” identified. Similarly, the system looks for terms like “insurance,” “healthcare,” “401k,” “benefits,” “paid leave,” and “pension” to determine if the job includes standard

Example Test Cases

To test the robustness of the system, several scenarios were tested manually:

When a genuine job listing with a clear and detailed description was entered, the model consistently predicted it as genuine, with high confidence scores.

When scammy or vague job offers (e.g., "earn $500 daily, no experience required") were tested, the model flagged them as fake correctly.

URLs pointing to homepage or unrelated content were correctly identified as not job-related.

Invalid URLs (e.g., missing https or poorly structured links) were rejected with an appropriate warning message.

These test cases confirmed that each major component—from URL validation to classification—was functioning as expected.

Limitations

- While the prototype performs well under controlled test cases, it comes with certain limitations:
- The small dataset size restricts the model's ability to generalize across job types, industries, and languages.
- HTML structure on job sites can vary widely; in some cases, extraction may fail or pull irrelevant content.
- The model currently supports only binary classification (real or fake) and doesn't account for partially suspicious or borderline cases.
- It doesn't handle interactive pages or those behind JavaScript-based rendering.

Future Enhancements

To enhance the system's reliability and scalability, the following improvements are recommended:

Gather and label a larger dataset of real and fake job posts from trusted sources.

Integrate deep learning models such as BERT or RoBERTa for more nuanced text understanding.

Add cross-validation pipelines and log prediction metrics over time.

Allow user feedback on prediction results to further train and adjust the model.

Implement support for multiple languages and country-specific job format.

This comprehensive testing and validation framework ensures that the Fake Job Detector prototype is functional, informative, and ready for further expansion into a production-ready syste

Validation Process

.

Validation was a critical step in ensuring that the **Fake Job Detector** application performs reliably across various scenarios. The following subsections break down the validation of each component of the system, including the machine learning model, URL input, web scraping logic, and additional feature detection (remote work and benefits).

Machine Learning Model Validation

The core classifier, built using a **TF-IDF Vectorizer** and a **Logistic Regression** model, was validated on a manually curated dataset consisting of labeled job descriptions. Although the dataset was small, it was designed to include both **authentic and deceptive job postings** with clear differences in tone, structure, and intent.

Since traditional validation techniques like cross-validation or a train-test split were limited by the dataset size, the model was evaluated through:

- Manual review of prediction outputs.
- Probability-based confidence scores to interpret certainty.
- Clear separation of genuine vs. scam job traits in feature space.

This basic validation confirmed that the model could detect key phrases commonly associated with scam

postings (e.g., "descriptions).

CHAPTER 9
CONCLUSIONS AND FUTURE ENHANCEMENTS

9.1 Conclusion

In an era where digital job scams are increasingly common and cleverly disguised, this project presents a timely and practical solution **a Fake Job Detection System** built using **machine learning** and **web scraping**. The goal was to design a tool that can help job seekers distinguish genuine opportunities from fraudulent ones with minimal effort.

Through the integration of **web technologies, natural language processing, and classification algorithms**, the application successfully achieves its purpose. The user-friendly interface built using **Streamlit** allows users to input any job-related URL. The system then intelligently evaluates the page in multiple stages:

- **URL validation** ensures proper formatting and reachability.
- **Content analysis** identifies whether the URL hosts a legitimate job posting.
- **Text extraction and classification** determines the authenticity of the job using a **TF-IDF + Logistic Regression model** trained on a curated dataset.
- Additional analysis checks whether the job is **remote-friendly** and whether it offers **employee benefits**.

Despite being trained on a relatively small dataset, the classifier performed reliably on real-world job pages from platforms like **Indeed**, **LinkedIn**, and **Naukri**. It successfully identified language patterns typical of scam postings, such as exaggerated promises, lack of qualification requirements, and overly vague descriptions.

The system also highlights **remote work availability** and **benefit offerings**, making it more than just a scam detector—it becomes a useful **job opportunity evaluator**. From backend model training to frontend user interaction, every component was validated for correctness, robustness, and usability.

In today's digital recruitment ecosystem, job seekers are increasingly vulnerable to deceptive job postings that promise high rewards with little effort. These scams not only exploit people's financial vulnerability but also erode trust in legitimate hiring platforms. The aim of this project was to tackle this issue by developing an intelligent system that detects fake job advertisements directly from their web URLs, leveraging machine learning, web scraping, and natural language processing techniques.

This system successfully combines automated data extraction with a text classification model, offering users a seamless experience that simulates how a human would verify a job's authenticity, but with far greater speed and consistency. The platform, developed using Streamlit, allows users to enter a job URL and get back a multi-dimensional analysis that includes:

URL Reachability & Relevance Detection

Upon submission, the tool first checks if the URL is valid, reachable, and actually hosts job-related content. It uses HTTP status codes and inspects HTML text for the presence of job-related keywords such as "apply", "hiring", "description", etc., thereby filtering out irrelevant pages early in the pipeline. This step ensures that the input is both valid and meaningful, preventing the model from being misled by non-job content.

Web Scraping & Content Parsing

If the URL passes the initial filter, the application extracts meaningful content using **BeautifulSoup**. It checks common structures found on job portals such as **Indeed**, **LinkedIn**, **Glassdoor**, and **Naukri**, and falls back to more generic content extraction if those selectors fail. The system collects text from relevant HTML tags to reconstruct a clean job description, which becomes the primary input for analysis.

Future Scope

To enhance performance and reliability, the project can be expanded by:

- Training on **larger and more diverse datasets** from real-world job portals.
- Implementing **deep learning models** for more nuanced language understanding.
- Adding **JavaScript rendering support** for better scraping of dynamic job sites.
- Integrating **user feedback loops** to improve model predictions over time.

Future Enhancements

Although the system performs effectively on structured datasets and standard job websites, there are areas for future improvement:

- Dataset Expansion: Currently, the model is trained on a small synthetic dataset. Incorporating a larger, more diverse dataset would significantly improve prediction accuracy.
- Model Upgrades: Switching to more powerful models like BERT, RoBERTa, or LSTM-based

architectures could better capture contextual nuances in job text.

- Dynamic Content Handling: Integrating headless browser scraping (e.g., Selenium or Playwright) would allow the system to handle dynamic JavaScript-based job portals.
- Continuous Learning: Adding user feedback and label correction functionality could make the model adaptive, improving over time.
- Multilingual Support: The current model is built for English-language postings. Extending support to other languages would broaden its global applicability.

Final Thoughts

This project demonstrates how **machine learning**, when combined with thoughtful design and domain understanding, can be used to solve real-world problems. The Fake Job Detector stands as a powerful prototype that has the potential to protect job seekers from scams and misinformation—empowering them with **clarity**, **confidence**, and **control** in their job search journey.

To conclude, this project offers a **valuable and practical solution** to the growing problem of online job scams. It demonstrates how combining **machine learning**, **web scraping**, and **user-centric design** can lead to tools that not only enhance decision-making but also protect users from potential exploitation. The Fake Job Detector stands as a proof-of-concept that can be scaled and refined for real-world deployment, empowering job seekers with **greater trust**, **security**, and **information** in their career journeys.

CHAPTER10

REFERENCES

1. **Frontend Design and Development**
   - HTML5, CSS3, JavaScript

     W3Schools: https://www.w3schools.com/

     MDN Web Docs: https://developer.mozilla.org/
2. **Fonts and Typography**
   - Google Fonts: https://fonts.google.com/
3. **Mokkarala, N., & Jain, P. (2019).**
   Fake job posting detection using supervised learning algorithms.
   In *International Journal of Engineering and Advanced Technology (IJEAT)*, 9(1), 2249–8958.
4. **Pedregosa, F., et al. (2011).**
   Scikit-learn: Machine Learning in Python.
   Journal of Machine Learning Research, 12, 2825–2830.
   https://scikit-learn.org/
5. **Loper, E., & Bird, S. (2002).**
   NLTK: The Natural Language Toolkit.
   In *Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics*, pp. 63–70.
   https://www.nltk.org/
6. **Rajaraman, A., & Ullman, J. D. (2011).**
   Mining of Massive Datasets.
   Cambridge University Press.
7. **TfidfVectorizer – scikit-learn documentation.**
   https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
8. **BeautifulSoup Documentation – Web Scraping with Python.**
   https://www.crummy.com/software/BeautifulSoup/bs4/doc/
9. **Streamlit Documentation – Turn data scripts into shareable web apps.**
   https://docs.streamlit.io/
10. **Job Scam Reports and Trends – Federal Trade Commission (FTC).**
    https://www.consumer.ftc.gov/articles/job-scams

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NBA and NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

Abstract Proforma

FBP / IOMP / MAJOR PROJECT

Year & Branch: III B.TECH & CSM	**Section: B**	**Batch No.:B6**
Academic Year: :2024-2025		**Regulation: R22**
Student Registration Details	Name	Roll Number
	1.K. MANOJ KUMAR 2.G. ALEKHYA	227Y1A6688 227Y1A6667
Name of the Guide & Designation	Dr K. BABU	
Area (Domain) of the Project	MACHINE LEARNING	
Title of the Project	Fake Job Detection Using Machine Learning from Web-Scraped Job Postings	
Tools Required	Python, Requests, BeautifulSoup, Pandas, Scikit-learn, Flask, Joblib	

Abstract

- **Background/Introduction:** The proliferation of online job portals has unfortunately led to an increase in fraudulent job postings, causing significant distress and financial losses to job seekers. Identifying these deceptive listings is crucial for maintaining trust and efficiency in the online recruitment process. This project addresses the challenge of automatically detecting fake job postings by leveraging machine learning techniques.
- **Objectives:** The primary objectives of this project are to:
  - Develop a system capable of extracting relevant textual information from job posting URLs.
  - Train a machine learning model to accurately classify job postings as either genuine or fake based on the extracted text.
  - Create a user-friendly web application that allows users to input a job posting URL and receive a prediction on its legitimacy.
- **Methodology:** Job details are extracted from URLs using web scraping (BeautifulSoup, Requests). Textual data is then transformed into numerical features using TF-IDF. A Logistic Regression model is trained on a labeled dataset (scikit-learn) and deployed with Flask for real-time URL-based predictions.
- **Expected Results/Outcomes**: The successful completion of this project is expected to yield a functional web application capable of accurately predicting the legitimacy of job postings based on their URL. The machine learning model is anticipated to achieve a high level of accuracy in distinguishing between genuine and fraudulent job listings, thereby providing a valuable tool for job seekers.
- **Significance/Impact**: This project offers a practical solution to combat the growing problem of fake job postings. By providing an accessible and automated tool for detection, it can empower job seekers to make informed decisions, avoid scams, and save valuable time and effort in their job search. The application has the potential to enhance the credibility of online job platforms and contribute to a safer and more trustworthy recruitment environment.

Key Words: Fake Job Detection, Machine Learning, Web Scraping, Natural Language Processing, Logistic Regression, Flask

Guide **Project Coordinator** **HOD**

Guidelines for a Strong Title:

1. **Be Specific:** The title should clearly indicate the focus of the project. Avoid vague or overly broad terms.
2. **Include Key Elements:** Mention the main components or technology used, the problem addressed, or the expected outcome.
3. **Be Concise:** Aim for a title that is succinct yet descriptive. Typically, a title should be between 10-15 words.
4. **Use Keywords:** Include important keywords that reflect the core of your project. This helps in making the title more searchable and relevant.

Example Title Components:

1. **Technology or Approach:** Mention if your project involves specific technologies (e.g., IoT, AI, machine learning).
2. **Application Area:** Indicate the field or area where the project is applied (e.g., agriculture, healthcare, education).
3. **Purpose or Goal:** Highlight the main objective or problem being addressed (e.g., optimization, enhancement, reduction).

Example Titles:

1. **Developing an IoT-Based Smart Irrigation System for Efficient Water Usage in Agriculture**
2. **AI-Driven Healthcare Monitoring System for Early Disease Detection**
3. **A Machine Learning Approach to Predictive Maintenance in Manufacturing Industries**
4. **Renewable Energy Solutions for Sustainable Urban Development**
5. **Designing an Educational Platform for Personalized Learning Using Adaptive Algorithms**

Crafting a Title for the Provided Example:

If we consider the earlier example of the smart irrigation system, a suitable title could be:

"IoT-Based Smart Irrigation System for Optimized Water Usage in Sustainable Agriculture"

This title clearly mentions:

- The technology used (IoT-Based)
- The main focus (Smart Irrigation System)
- The goal (Optimized Water Usage)
- The application area (Sustainable Agriculture)

By following these guidelines, you can create a title that is informative, specific, and engaging for your project abstract.