NexGen Resume Parser

A report submitted in partial fulfillment of the requirements for the award of a degree of

Bachelor Of Technology

in

Computer Science And Engineering

By

A. Saikiran Reddy

(21EG505802)

P. Nikhil

(21EG505856)

S. Venkatesh

(21EG505863)

Under the Guidance of

Ms. A. Durga Bhavani

Assistant Professor, Department of CSE

Department of Computer Science and Engineering

ANURAG UNIVERSITY

Venkatapur(v), Ghatkesar(M), Medchal(D)., T.S-50008

(2023-2024)

DECLARATION

We hereby declare that the report entitled **"NexGen Resume Parser"** submitted for the award of the degree of **Bachelor of Technology (B. Tech)** in Computer Science and Engineering is a record of an original work done by us and the report has not formed the basis for the award of any degree, diploma, associateship or fellowship of similar other titles. It has not been submitted to any other University or Institution for the award of any degree or diploma.

A.Saikiran Reddy
21EG505802

Place:
Date:

P. Nikhil
21EG505863

S. Venkatesh
21EG505863

CERTIFICATE

This is to certify that the report entitled **" NexGen Resume Parser "** is being submitted by **Mr. A. Saikiran Reddy** bearing the Hall Ticket number **21EG505802**, **Mrs. P.Nikhil** bearing the Hall Ticket number **21EG505856**, **Mr. S.Venkatesh** bearing the Hall Ticket number **21EG505863** in partial fulfillment for the award of the Bachelor of Technology in Computer Science and Engineering to the Anurag University is a record of bonafide work carried out by them under my guidance and supervision.

The results embodied in this report have not been submitted to any other University or Institute for the award of any other degree or diploma.

.

Signature of Supervisor
Ms. A.Durga Bhavani
Assistant Professor
Department of CSE

Signature of Dean
Dr. G. Vishnu Murthy
Dean, CSE

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere thanks and deep sense of gratitude to project supervisor **Mrs. A.Durga Bhavani,** Assistant Professor, Dept of CSE for her constant encouragement and inspiring guidance without which this project could not have been completed. Her critical reviews and constructive comments improved our grasp of the subject and steered to the fruitful completion of the work. Her patience, guidance and encouragement made this project possible.

We would like to express our special thanks to **Dr. V. Vijaya Kumar**, Dean School of Engineering, Anurag University, for his encouragement and timely support in our B.Tech program.

We would like to acknowledge our sincere gratitude for the support extended by **Dr. G. Vishnu Murthy**, Dean, Dept. of CSE, Anurag University. We also express our deep sense of gratitude to **Dr. V V S S S Balaram**, Academic Coordinator, **Dr. Pallam Ravi**, Project Coordinator and Project Review Committee members, whose research expertise and commitment to the highest standards continuously motivated us during the crucial stage of our project work.

A. Saikiran Reddy

(21EG505802)

P. Nikhil

(21EG505856)

S. Venkatesh

(21EG50586

ABSTRACT

The NexGen Resume Parse Project is focused on advancing the field of resume parsing through the application of cutting-edge natural language processing (NLP) and machine learning (ML) techniques. This project aims to develop a state-of-the-art system capable of accurately extracting relevant information from resumes, including personal details, work experience, skills, and education. One of the key objectives of this project is to enhance the semantic understanding of resume text, enabling the system to not only extract information but also comprehend the context and meaning of the text. To achieve these objectives, the project will leverage advanced NLP techniques such as named entity recognition (NER), part-of-speech tagging, and dependency parsing. These techniques will be instrumental in accurately identifying and extracting information from resumes, ensuring that the system can process diverse resume formats and styles effectively. Additionally, the project will utilize machine learning models trained on labeled resume datasets to improve the accuracy of information extraction and classification.The system developed as part of this project will be designed to be scalable and efficient, capable of processing large volumes of resumes quickly and accurately. This scalability will be crucial in enabling the system to handle the demands of real-world recruitment processes, where large numbers of resumes need to be processed in a timely manner. Furthermore, the project will focus on providing an API and integration tools that will allow the system to be easily integrated into existing recruitment platforms and workflows, making it accessible to a wide range of users.

CONTENTS

List of Tables

List of Figures

1. INTRODUCTION

Introducing "NexGen Resume Parser": a revolutionary platform reshaping recruitment through advanced NLP techniques. This system utilizes sophisticated algorithms to thoroughly analyze resumes, evaluating educational backgrounds, professional experiences, and additional qualifications. Notably, it provides candidates with an ATS (Applicant Tracking System) score, a comprehensive assessment of their profile's compatibility with job prerequisites, delivering invaluable insights.

Beyond assessment, the platform actively supports candidates by offering direct links to recent job openings aligned with their skills and experience. Tailored job recommendations, based on individual resumes, ensure a personalized job search. What sets "NexGen Resume Parser" apart is its proactive approach is once the ATS score is obtained, it's seamlessly sent to the candidate's email. This feature enhances the candidate experience by providing instant feedback and empowering them with insights for continuous improvement. With intelligent algorithms and comprehensive analysis, this platform optimizes the matching process, fostering meaningful connections between talent and employment opportunities.

The NexGen Resume Parse Project is a cutting-edge initiative focused on advancing resume parsing technology through the use of state-of-the-art natural language processing (NLP) and machine learning (ML) techniques. This project aims to develop a highly accurate and efficient system capable of extracting relevant information from resumes, such as personal details, work experience, skills, and education, while also enhancing the system's semantic understanding of resume text.By leveraging advanced NLP techniques such as named entity recognition (NER) and machine learning models trained on labeled resume datasets, the project seeks to significantly improve the accuracy and effectiveness of resume parsing. The system developed as part of this project will be designed to be scalable and efficient, capable of processing large volumes of resumes quickly and accurately.

One of the key objectives of the project is to provide an API and integration tools that will allow the system to be easily integrated into existing recruitment platforms and

workflows. This will make the system accessible to a wide range of users and enable it to seamlessly fit into various recruitment processes.

Overall, the NexGen Resume Parse Project represents a major advancement in the field of resume parsing, with the potential to greatly enhance the efficiency and accuracy of the recruitment process. By combining advanced NLP and ML techniques with a focus on scalability and integration, this project aims to set a new standard for resume parsing systems, benefiting both candidates and employers alike.

1.1 Motivation

The motivation behind the NexGen Resume Parse Project stems from several key factors:

Improving Recruitment Efficiency: Traditional resume parsing methods are often time-consuming and error-prone, leading to inefficiencies in the recruitment process. By developing a state-of-the-art resume parsing system, the project aims to streamline the process, saving time and resources for both candidates and employers.Enhancing Accuracy and Reliability: Current resume parsing systems may struggle with accurately extracting information from resumes, especially when dealing with diverse formats and styles. The project seeks to improve the accuracy and reliability of information extraction by leveraging advanced NLP and ML techniques.Addressing Semantic Understanding: Understanding the context and meaning of resume text is crucial for accurate information extraction. The project aims to enhance semantic understanding through advanced NLP techniques, enabling the system to extract information more effectively.Scalability and Integration: As recruitment processes increasingly rely on digital platforms and automation, there is a growing need for scalable and easily integrable resume parsing systems. The project aims to address these needs by developing a system that is both scalable and can be seamlessly integrated into existing platforms.

1.2 Problem Defination

Traditional recruitment processes are sluggish, struggling to quickly identify ideal candidates. Job seekers lack personalized guidance and real-time feedback on applications. Existing Applicant Tracking Systems (ATS) fall short in providing immediate insights into compatibility. "NexGen Resume Parser" revolutionizes this landscape, leveraging advanced NLP for comprehensive resume analysis and offering a seamless, proactive job search experience. It bridges the gap between talent and opportunities, transforming recruitment into a more efficient and empowering process.

1.3 Objectives of the Project

"NexGen Resume Parser" aims to revolutionize recruitment through advanced NLP, streamlining candidate identification and enhancing efficiency. The platform empowers job seekers with personalized guidance and instant feedback, optimizing the Applicant Tracking System for immediate insights. Its objective is to proactively deliver valuable information, fostering meaningful connections between talent and job opportunities. Ultimately, it transforms the recruitment landscape into a more efficient and insightful process.

2.Literature Survey

A brief literature survey for the NexGen Resume Parse Project would involve reviewing key research papers and publications related to resume parsing, natural language processing (NLP), and machine learning (ML) techniques. Here is a summary of the main findings:

Resume Parsing Techniques: Existing literature demonstrates a shift from rule-based parsing to more advanced techniques such as machine learning and NLP-based approaches. These approaches have shown improved accuracy and efficiency in extracting information from resumes.

NLP for Resume Parsing: NLP techniques such as named entity recognition (NER), part-of-speech tagging, and dependency parsing have been widely used to extract structured information from unstructured resume text. These techniques help in identifying entities such as skills, experience, and education.

Machine Learning for Resume Parsing: ML models, including support vector machines (SVM), random forests, and neural networks, have been successfully applied to resume parsing. These models are trained on labeled datasets to classify and extract information from resumes.

Semantic Understanding of Resumes: Recent research focuses on enhancing the semantic understanding of resume text. Techniques such as semantic parsing and contextual analysis are used to improve the accuracy of information extraction and interpretation.

Scalability and Efficiency: Studies have addressed the scalability and efficiency challenges of resume parsing systems. Techniques such as parallel processing, distributed computing, and optimization algorithms are explored to handle large volumes of resumes efficiently.

Integration and API Development: Literature emphasizes the importance of developing APIs and integration tools for seamless integration of resume parsing systems into

existing recruitment platforms. Standardization and interoperability are key considerations in this area.

Evaluation Metrics: Evaluation metrics such as accuracy, precision, recall, and F1-score are commonly used to assess the performance of resume parsing systems. Studies highlight the importance of using domain-specific evaluation criteria for meaningful evaluation.

By reviewing these key areas in the literature, the NexGen Resume Parse Project can gain valuable insights and build on existing research to develop an innovative and effective resume parsing system.

3. Analysis

3.1 Existing System

The existing systems for resume parsing vary widely in terms of complexity and effectiveness. Traditional systems typically rely on rule-based methods, where specific keywords or patterns are used to extract information from resumes. While these systems can be effective in some cases, they often struggle with parsing resumes that deviate from expected formats or contain unconventional language.

More advanced systems utilize natural language processing (NLP) and machine learning (ML) techniques to improve parsing accuracy and efficiency. These systems often employ techniques such as named entity recognition (NER), part-of-speech tagging, and syntactic parsing to extract information from resumes. However, even these systems can face challenges with complex resume formats and semantic understanding.

Some existing systems also offer integration with applicant tracking systems (ATS) and other recruitment platforms, allowing for seamless transfer of parsed resume data. These systems often provide APIs and integration tools to facilitate this process

Lack of Transparent Infrastructures:

Many data owners are hesitant to share their data due to the absence of transparent systems for establishing data trust, leading to challenges in promoting data sharing

Concerns about Data Quality:

Data consumers express concerns about the accuracy and reliability of shared data, indicating a lack of confidence in the quality of the data being exchanged

Trust Issues:

Existing systems struggle to ensure the integrity and quality of shared data from its source to utilization, resulting in trust issues for both data owners and consumers

Privacy Concerns:

Privacy issues, data abuse, and moral and legal transgressions contribute to the reluctance of data owners to share their data, impacting the overall data sharing ecosystem

Data Ownership Challenges:

Challenges related to data ownership and control hinder seamless and secure data sharing practices, affecting both data owners and consumers.

3.2 Proposed System

The advantages of the proposed NextGen Resume Parser is Analysing the Resume and Generating Score.

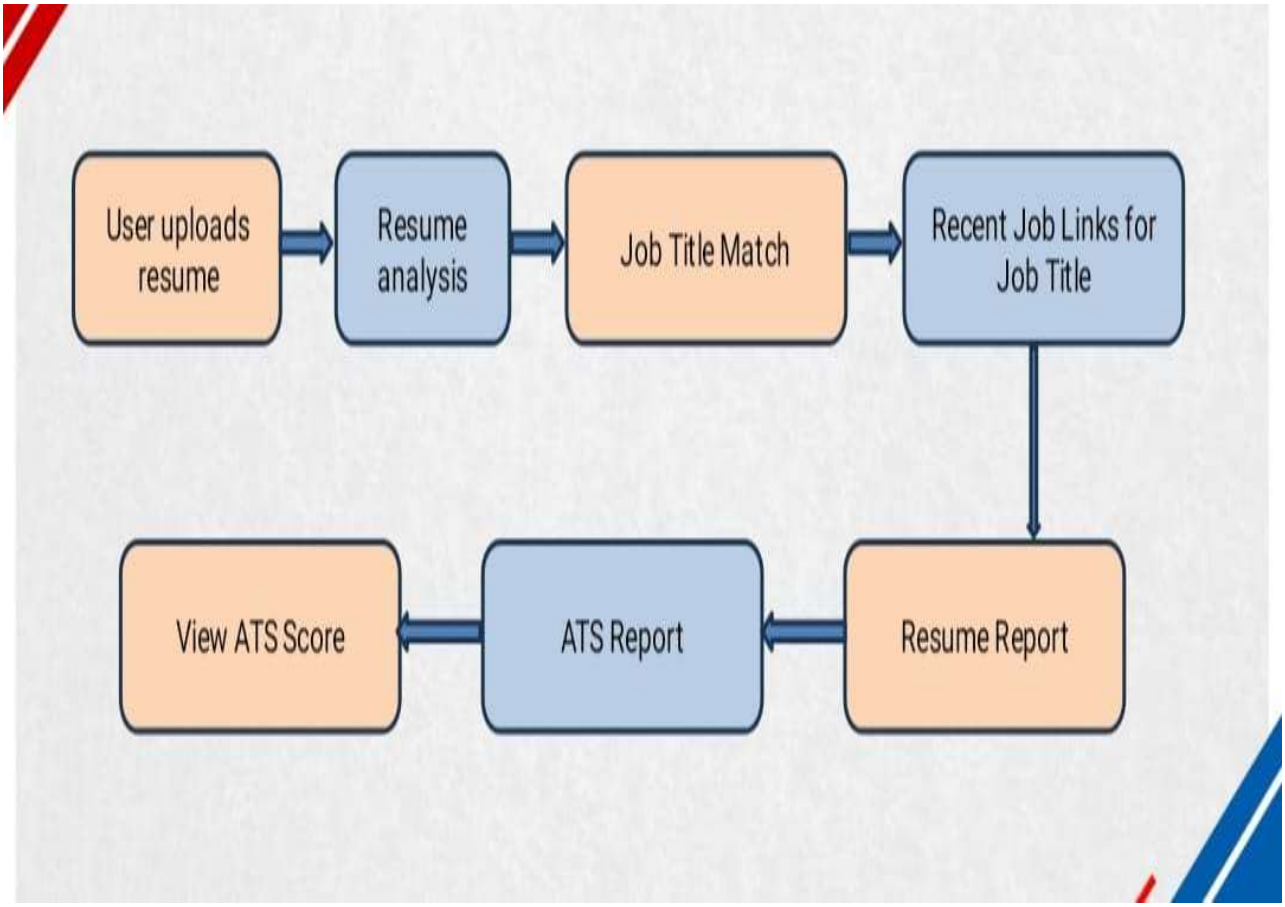

Fig:3.1: Flow Chart

Input Resume: The process begins with the input of a resume in various formats, such as PDF, Word, or plain text.

Text Extraction: The system extracts the text from the resume document, preparing it for further processing.

Preprocessing: The extracted text undergoes preprocessing steps, such as removing stop words, punctuation, and formatting inconsistencies.

Tokenization: The preprocessed text is tokenized into individual words or phrases, which serve as the basic units for analysis.

Named Entity Recognition (NER): The system uses NLP techniques, such as NER, to identify and extract named entities from the resume text, such as names, skills, education, and experience.

Semantic Understanding: The system analyzes the context of the extracted entities to understand their meaning in the context of the resume, enhancing the semantic understanding of the text.

Information Extraction: Based on the identified entities and semantic understanding, the system extracts relevant information, such as contact details, work experience, skills, and education, from the resume.

Machine Learning Classification: The system may utilize machine learning models to classify and extract information from resumes, improving accuracy and efficiency.

Output: The final output of the system includes structured data extracted from the resume, which can be used for further analysis or integration with other systems.

Integration with Recruitment Platforms: The extracted data can be integrated with applicant tracking systems (ATS) and other recruitment platforms, allowing for seamless processing of candidate resumes.

Feedback Loop: The system may incorporate a feedback loop to continuously improve its performance based on user interactions and feedback.

3.3 Software Requirement Specification

3.3.1 Purpose

The purpose of the NexGen Resume Parse Project is to revolutionize the resume parsing process by developing a state-of-the-art system that leverages advanced natural language processing (NLP) and machine learning (ML) techniques. Improve the accuracy of information extraction from resumes by using advanced NLP techniques, such as named entity recognition (NER) and semantic parsing. Increase the efficiency of resume parsing by developing a scalable system capable of processing large volumes of resumes quickly and accurately.

3.3.2 Scope

The scope of the NexGen Resume Parse Project encompasses several key areas:

Research and Development: Conducting research on advanced natural language processing (NLP) and machine learning (ML) techniques for resume parsing, and developing a state-of-the-art resume parsing system based on these techniques.

System Design and Implementation: Designing and implementing a scalable and efficient resume parsing system that can process large volumes of resumes quickly and accurately.

Semantic Understanding: Enhancing the system's semantic understanding of resume text, enabling it to extract information accurately and in context.

Integration: Providing an API and integration tools that allow the system to be easily integrated into existing recruitment platforms and workflows.

Evaluation and Testing: Evaluating the performance of the system using standard metrics and testing it with real-world resume data to ensure accuracy and efficiency.

3.3.3 Overall Description

H/W System Configuration:-

Processor - Pentium –IV

RAM - 4 GB (min)

Hard Disk - 20 GB

Software Requirements:

Operating System - Windows XP

Coding Language - Python

Front End - J2EE

Back End - MySQL

4.IMPLEMENTATION

Client-Server Overview:

The client-server architecture, amid the myriad topics in computer science, has generated significant discourse, often marked by more hype than reality. This technology has captured critical mass attention, with dedicated conferences and major vendors like IBM and DEC declaring it as a focal point. A survey by DBMS magazine indicates a keen interest, with 76% of readers actively exploring client-server solutions. The growth of client-server development tools from $200 million in 1992 to over $1.2 billion in 1996 underscores its transformative trajectory.

Client-Server Implementation:

The essence of client-server implementations is both complex and potent. At its core, a client represents an application with local resources capable of seeking database services from a remote server. The intermediary software, often referred to as middleware, plays a crucial role in facilitating seamless communication. Clients, typically PCs or workstations, connect through a network to a more potent server, capable of handling requests from multiple clients. The fundamental idea is to insulate clients from the physical nuances of data, enabling transparent access to local and remote databases.

Front End and User Interface Design:

The user interface, a critical component, is designed within a browser-specific environment with an emphasis on an Intranet-Based Architecture for distributed concepts. HTML standards lay the foundation for browser-specific components, while Java Server Pages (JSP) add dynamism to the design. Communication architecture hinges on Servlets and Enterprise Java Beans, with Java Database Connectivity (JDBC) acting as the bridge for database connectivity. The three-tier architecture prioritizes higher cohesion and limited coupling for operational effectiveness.

Python :

Python plays a crucial role in the NexGen Resume Parse Project, primarily due to its versatility and the availability of powerful libraries and frameworks for natural language processing (NLP) and machine learning (ML). Here are some key roles of Python in this project:

NLP Libraries:

Python has several NLP libraries, such as NLTK, spaCy, and TextBlob, which provide tools for text preprocessing, tokenization, part-of-speech tagging, named entity recognition (NER), and semantic analysis. These libraries are essential for extracting information from resumes and understanding the semantic meaning of the text.

ML Frameworks:

Python is widely used for ML tasks, thanks to libraries like TensorFlow, PyTorch, and scikit-learn. These libraries can be used to train ML models for classifying and extracting information from resumes, improving the accuracy and efficiency of the parsing process.

Web Development:

Python web frameworks like Flask and Django are well-suited for building web applications and APIs. In the NexGen Resume Parse Project, Python can be used to develop an API that allows the resume parsing system to be easily integrated into existing recruitment platforms and workflows.

Data Processing:

Python's built-in data structures and libraries, such as pandas and NumPy, are ideal for processing and analyzing large volumes of resume data. These libraries can be used for data preprocessing, feature extraction, and model training.

Integration:

Python's flexibility and ease of integration with other technologies make it a suitable choice for integrating the resume parsing system with other systems and platforms used in the recruitment process.

4.1 List of program files

In a Python project for the NexGen Resume Parse Project, you might have the following files:

main.py: The main entry point for the application, responsible for initializing and running the resume parsing system.

resume_parser.py: Contains the logic for parsing resumes, including text extraction, preprocessing, and information extraction using NLP and ML techniques.

api.py: Defines the API endpoints and handles requests from external systems for parsing resumes and retrieving parsed data.

models.py: Contains the definitions of ML models used for classifying and extracting information from resumes.

utils.py: Contains utility functions used across the project, such as file handling, data processing, and formatting.

config.py: Configuration file that stores settings such as API keys, database connection details, and other environment-specific configurations.

requirements.txt: A file listing all the Python dependencies required for the project, which can be installed using pip.

README.md: Documentation file that provides an overview of the project, installation instructions, and usage guidelines.

data/: Directory containing sample resume data for testing and training the resume parsing system.

tests/: Directory containing unit tests and integration tests for the project.

This is a generalized list, and the actual files and structure may vary based on the specific requirements and implementation choices of the project.

4.2 List of Libraries

For a Python-based NexGen Resume Parse Project, you might consider using the following libraries and frameworks:

Natural Language Processing (NLP) Libraries:

NLTK: Provides tools for text processing and NLP tasks.

spaCy: Offers advanced NLP features like tokenization, named entity recognition (NER), and dependency parsing.

TextBlob: Simplifies text processing tasks such as sentiment analysis, part-of-speech tagging, and noun phrase extraction.

Machine Learning (ML) Frameworks:

TensorFlow: A powerful ML framework for building and training ML models.

PyTorch: Another popular ML framework known for its flexibility and ease of use.

scikit-learn: Provides simple and efficient tools for data mining and data analysis, including ML algorithms for classification and regression.

Web Development Frameworks:

Flask: A lightweight web framework for building APIs and web applications.

Data Processing and Analysis Libraries:

pandas: Offers data structures and tools for data manipulation and analysis.

NumPy: Provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

Other Useful Libraries:

requests: Simplifies making HTTP requests to external APIs.

beautifulsoup4: A library for parsing HTML and XML documents, useful for web scraping.

matplotlib: A plotting library for creating static, animated, and interactive visualizations in Python.

5. EXPERIMENTAL RESULT

5.1 Experiment setup

The experiment setup for the NexGen Resume Parse Project would involve several key components:

Dataset Selection: Select a diverse dataset of resumes in various formats (e.g., PDF, Word, plain text) to train and test the resume parsing system. The dataset should include resumes with different layouts, styles, and content to ensure the system's robustness.

Data Preprocessing: Preprocess the dataset to remove any formatting inconsistencies, punctuation, and irrelevant information that may affect the parsing process. Tokenize the text and perform any necessary cleaning and normalization steps.

Model Selection: Choose suitable machine learning models for resume parsing, such as NER models for named entity extraction and classification models for categorizing resume sections

(e.g., work experience, education, skills).

Training and Testing: Split the dataset into training and testing sets. Train the selected models on the training set and evaluate their performance on the testing set using metrics like accuracy, precision, recall, and F1-score.

Hyperparameter Tuning: Perform hyperparameter tuning to optimize the performance of the models. This may involve using techniques like grid search or random search to find the best combination of hyperparameters.

Cross-Validation: Perform cross-validation to assess the generalization performance of the models. This helps to ensure that the models are not overfitting to the training data.

Evaluation Metrics: Evaluate the performance of the resume parsing system using standard evaluation metrics, such as accuracy, precision, recall,

and F1-score. These metrics provide insights into the system's effectiveness in extracting information from resumes.

Comparative Analysis: Compare the performance of the NexGen Resume Parse Project with existing resume parsing systems or benchmarks to demonstrate its effectiveness and superiority.

Running the Experiment:

Running the experiment for the NexGen Resume Parse Project involves executing the experiment setup outlined earlier. Here's a step-by-step guide on how to run the experiment:

Dataset Selection: Choose a diverse dataset of resumes that represent the types of documents the system will encounter in real-world scenarios.

Data Preprocessing: Preprocess the dataset by removing irrelevant information, tokenizing the text, and performing any necessary cleaning and normalization steps.

Model Selection: Select the machine learning models to be used for resume parsing, such as NER models and classification models.

Training and Testing: Split the dataset into training and testing sets. Train the selected models on the training set and evaluate their performance on the testing set using metrics like accuracy, precision, recall, and F1-score.

Hyperparameter Tuning: Perform hyperparameter tuning to optimize the performance of the models. Use techniques like grid search or random search to find the best combination of hyperparameters.

Cross-Validation: Perform cross-validation to assess the generalization performance of the models. This helps to ensure that the models are not overfitting to the training data.

Evaluation: Evaluate the performance of the resume parsing system using the chosen evaluation metrics. Compare the results with the expected performance based on the experiment setup.

Comparative Analysis: Compare the performance of the NexGen Resume Parse Project with existing resume parsing systems or benchmarks to demonstrate its effectiveness and superiority.

Documentation: Document the experiment setup, including details of the dataset, preprocessing steps, model selection, training process, evaluation metrics, and results. This documentation is crucial for replicability and transparency.

Sensitivity Analysis: Perform sensitivity analysis to assess the robustness of the resume parsing system to variations in the dataset and input parameters.

5.2 Parameters with Formulas

To run experiments and evaluate the performance of the NexGen Resume Parse Project, you can use the following parameters and formulas:

Accuracy: Accuracy measures the percentage of correctly parsed resumes out of the total number of resumes processed.

Formula:
Accuracy=Number of correctly parsed resumesTotal number of resumes×10 0Accuracy=Total number of resumesNumber of correctly parsed resumes ×100

Precision: Precision measures the ratio of correctly identified resume sections (true positives) to the total number of sections identified by the system.

Formula: Precision=True PositivesTrue Positives+False Positives Precision=True Positives+False PositivesTrue Positives

Recall: Recall measures the ratio of correctly identified resume sections (true positives) to the total number of actual sections in the resumes.

Formula: Recall=True PositivesTrue Positives+False NegativesRecall= True Positives+False NegativesTrue Positives

F1-score: F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics.

Formula: F1-score=2×Precision×RecallPrecision+Recall F1-score=2×Precision+RecallPrecision×Recall

5.3 Sample Code

Connect:

```
import streamlit as st
import pdfplumber
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re
import requests
from bs4 import BeautifulSoup
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import io
import sys
import base64
import spacy
```

```
import phonenumbers
import pandas as pd
import time
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import random
import string
from streamlit import components
from pdfminer.high_level import extract_text
from pdf2image import convert_from_path
from PIL import Image
import pytesseract
import enchant
import mimetypes
import language_tool_python
from textblob import TextBlob
from PyPDF2 import PdfReader
import docx2txt
from zipfile import BadZipFile
import PyPDF2
from docx import Document
from email.mime.text import MIMEText
import smtplib
from cryptography.fernet import Fernet
import language_tool_python
```

```
from dateutil import parser

from dateutil.parser import ParserError

from twilio.rest import Client

import speech_recognition as sr

from spellchecker import SpellChecker

from nltk import pos_tag, word_tokenize

from langcodes import Language

import langid

import sounddevice as sd

from pydub import AudioSegment

from linkedin_jobs_scraper import LinkedinScraper

from linkedin_jobs_scraper.events import Events, EventData

from linkedin_jobs_scraper.query import Query, QueryOptions, QueryFilters

from linkedin_jobs_scraper.filters import RelevanceFilters, TimeFilters, TypeFilters,
ExperienceLevelFilters

import requests, pickle

from bs4 import BeautifulSoup

import re

from webdriver_manager.chrome import ChromeDriverManager

from googlesearch import search

from jinja2 import Template

import base64

import json

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from urllib.request import urlopen as uReq
```

```
from bs4 import BeautifulSoup as soup
import traceback
from requests_html import HTMLSession



st.set_page_config(layout="wide", initial_sidebar_state="collapsed",
page_title="Resume Parser", page_icon="□")
# Define functions for each page
# Initialize ATS score



# Global variable to store ATS scores for each resume
ats_scores = [0.0]

# Function to calculate ATS score
def calculate_ats_score(matched=True):
    score_increment = 0.1 if matched else 0.0
    ats_scores[-1] += score_increment

def home_page():



    watermark_html = """
        <style>
            @keyframes moveWatermark {
```

```
    0% {
        transform: translate(-100vw, -50%) rotate(-30deg);
    }
    25% {
        transform: translate(calc(100vw - 50%), -50%) rotate(-30deg);
    }
    50% {
        transform: translate(50%, -50%) rotate(-30deg);
    }
    75% {
        transform: translate(50%, 100vh) rotate(-30deg);
    }
    100% {
        transform: translate(-100vw, 50%) rotate(-30deg);
    }
}

.watermark {
    position: fixed;
    top: 50%;
    left: 50%;
    font-size: 104px;
    color: rgba(0, 0, 0, 0.2);
    pointer-events: none;
    z-index: 9999;
    animation: moveWatermark 20s infinite linear;
```

```
        }
    </style>
    <div class="watermark">Anurag University</div>
"""



# Render the watermark HTML
st.markdown(watermark_html, unsafe_allow_html=True)
hide_st_style = """
            <style>
            #MainMenu {visibility: hidden;}
            header {visibility: hidden;}
            </style>
            """
st.markdown(hide_st_style, unsafe_allow_html=True)
# Define the generate_captcha function
image_url = "https://d3kqdc25i4tl0t.cloudfront.net/articles/content/57aca545fa80af785f1df9127cf971fe_HeroWhatisanATS.jpg"  # Replace with the actual URL
centered_image = f'<div style="display: flex; justify-content: center;"><img src="{image_url}" width="450"></div>'
# Display the centered image
st.markdown(centered_image, unsafe_allow_html=True)


# Add a fixed black-colored footer with white text at the bottom of the page
st.markdown(
    """
```

```
    <style>
        body {
            margin: 0;
            padding: 0;
        }

        footer {
            position: fixed;
            bottom: 0;
            left: 0;
            width: 100%;
            background-color: black;
            padding: 10px;
            color: white;
            text-align: center;
        }
    </style>
    """,
    unsafe_allow_html=True,
)

# Display the fixed footer content
st.markdown(
    """
    <footer> ~ Made by Anireddy Saikiran Reddy</footer>
    """,
```

```
    unsafe_allow_html=True,
)

st.markdown(
    """
    <style>
        .custom-container {
            max-width: 800px;
            margin: auto;
            padding: 20px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            border-radius: 10px;
        }
        .custom-header {
            background-color: #68C1D4;
            padding: 20px;
            border-radius: 10px 10px 0 0;
        }
        .custom-title {
            text-align: center;
            color: white;
            font-size: 36px;
            margin: 0;
        }
    </style>
    <div class='custom-container'>
```

```
        <div class='custom-header'>
            <h1 class='custom-title'> □ NexGen Resume Parser</h1>
        </div>
    </div>
    """,
    unsafe_allow_html=True
)
st.subheader("□ Description:")
st.markdown("*NexGen Resume Parser*: A cutting-edge, free tool that serves as your resume's personal guide, scanning for missing skills and keywords to align seamlessly with job descriptions, ensuring your application stands out effortlessly.")

with st.expander("□ Click For How It Works . . . ."):
    st.info("□ How It Works\n\n"
            "*1. Upload Resume:*\n\n"
            "Begin by uploading your resume.\n\n Enter the job description or link of the role you're eyeing into our platform.\n\n"
            "*2. Instant Results:*\n\n"
            "Receive instant, comprehensive feedback.\n\n Delve into your Job Title Match percentage, Keyword Analysis, and discover common resume elements.\n\n"
            "*3. In-Depth Exploration:*\n\n"
"Access Explore More Jobs feature, unveiling opportunities on 10 related websites tailored to your role.\n\n"
"*4. Gain insights with personalized Resume and ATS reports.*\n\n"
```

"Gain insights with personalized Resume and ATS reports.\n\n Understand your strengths and areas for improvement to refine your application.\n\n"

"*5. Resume Refinement:*\n\n"

"Leverage our platform to build and enhance your resume.\n\nSeamlessly organize your job hunt using a private career board for efficient application tracking.\n\n")

Replace 'path_to_your_image_file' with the actual file path on your system

background_image_url = 'https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQJf2AoULiU9dTR4llXnQGQMusvVDoY6zRWTg&usqp=CAU'

st.markdown(f"""

<style>

.upload-button {{

width: 60px;

height: 60px;

background-color: lightgreen;

border: none;

border-radius: 50%;

color: white;

display: flex;

justify-content: center;

align-items: center;

background-image: url('{background_image_url}');

```
            background-size: cover;
        }}
    </style>
""", unsafe_allow_html=True)

st.markdown("<h2 style='font-weight: bold;'> Job Matching and Recent Job Links</h2>", unsafe_allow_html=True)

# Apply CSS styling to make the text blink
style = """
<style>
.blink {
    animation: blinker 1s linear infinite;
}
@keyframes blinker {
    50% {
        opacity: 0;
    }
}
</style>
"""

# Display the blinking and styled text
st.markdown(style, unsafe_allow_html=True)
st.markdown("<p class='blink' style='font-size: 24px; background-color: #FFFF00;'>Upload resume (DOCX only)</p>", unsafe_allow_html=True)
```

```
# Streamlit code
# Initialize global variable
resume_texts = []

# Custom CSS for positioning and animation
st.markdown("""
   <style>
      .upload-container {
         position: fixed;
         top: 10px;
         left: 10px;
         z-index: 999;
         animation: rotate 2s linear infinite;
      }

      .upload-button {
         width: 60px;
         height: 60px;
         font-size: 24px;
         background-color: lightgreen;
         border: none;
         border-radius: 50%;
         color: white;
         display: flex;
```

```
            justify-content: center;
            align-items: center;
        }

        @keyframes rotate {
            100% {
                transform: rotate(360deg);
            }
        }
    </style>
""", unsafe_allow_html=True)

# Upload button
upload_button_clicked = st.markdown('<div class="upload-container"><button class="upload-button"></button></div>', unsafe_allow_html=True)

# Only show uploaded files and process button if upload button is clicked
if upload_button_clicked:
    # Allow users to upload multiple DOCX files
    uploaded_resumes = st.file_uploader("Choose DOCX files", type=["docx"], accept_multiple_files=True)

    # Process button
    click = st.button("Process")
```

```
    # Update resume_texts when the "Process" button is clicked
    if click and uploaded_resumes:
        for uploaded_resume in uploaded_resumes:
            file_type = uploaded_resume.type


            if file_type == "application/pdf":
                with pdfplumber.open(uploaded_resume) as pdf:
                    pages = pdf.pages
resume_text = " ".join([page.extract_text() for page in pages])
resume_texts.append(resume_text)
elif file_type == "application/vnd.openxmlformats-officedocument.wordprocessingml.document":
doc = Document(uploaded_resume)
paragraphs = [paragraph.text for paragraph in doc.paragraphs]
resume_text = " ".join(paragraphs)
resume_texts.append(resume_text)




    # Function to match job description with resume
    def match_resume(job_description, resume):
        # Your matching logic goes here (use spaCy, NLTK, or other libraries for advanced matching)
        # For simplicity, let's use a basic keyword matching approach
        job_keywords = set(job_description.lower().split())
        resume_keywords = set(resume.lower().split())
```

```
    matching_score = len(job_keywords.intersection(resume_keywords)) / len(job_keywords)
    return matching_score * 100


def calculate_ats_score(matched):
    # Your logic for calculating ATS score goes here
    pass


# Textbox for job description
    # Define a list of predefined job descriptions
predefined_job_descriptions = [
    "We are in search of a talented Cloud Engineer to enhance our team's capabilities. As a Cloud Engineer, you will play a crucial role in crafting, implementing, and maintaining our cloud infrastructure. The ideal candidate possesses a deep understanding of cloud platforms, with hands-on experience in AWS, Azure, or GCP. Your responsibilities will involve designing scalable and efficient cloud solutions, ensuring optimal performance, and adhering to security best practices. Proficiency in containerization tools such as Docker and orchestration frameworks like Kubernetes is a must.",
    "Data Scientist with strong skills in machine learning and data analysis",
    "Marketing Specialist with experience in digital marketing and SEO",
    # Add more predefined job descriptions as needed
]


# Allow the user to select a job description from the dropdown
```

```
    selected_job_description = st.selectbox("Select a job description:", predefined_job_descriptions)

    # Display the selected job description
    st.text_area("Selected Job Description:", selected_job_description, height=100)

    # Check for matching when both resume and job description are provided
    if len(selected_job_description.split()) >= 50:
        if resume_texts:
            for idx, resume_text in enumerate(resume_texts):
                matching_score = match_resume(selected_job_description, resume_text)
                st.subheader(f"Matching Score for Resume {idx + 1}: {int(matching_score)}%")

                threshold = 50  # Set the threshold for a successful match
                if matching_score >= threshold:
                    st.success(f"☑ Resume {idx + 1} is suitable for the job.")
                    calculate_ats_score(matched=True)
                else:
                    st.error(f"❌ Resume {idx + 1} may not be a perfect match for the job.")
                    st.warning("Caution: The resume might not align perfectly with the job requirements. Just a heads-up!\n\nThe main text on your resume should be left-aligned, or left-justified.\n\nThe resume header format that
```

```
you choose should always be at the starting of the page. Select it as the left margin or put it at the center.")
        else:
            st.warning("Please upload at least one resume.")
    else:
        st.warning("Please select a job description with at least 50 words.")

    # Save resume_text to session_state
    st.session_state.uploaded_resumes = uploaded_resumes
    st.session_state.resume_texts = resume_texts
    candidate_email_resume=st.session_state
    click=st.session_state

def ResumeReport():
    global ats_scores

    # Access resume_text, uploaded_resume, and ats_score from the session state
    uploaded_resumes = st.session_state.uploaded_resumes
    resume_texts = st.session_state.resume_texts
    candidate_email_resume=st.session_state

    hide_st_style = """
                <style>
                #MainMenu {visibility: hidden;}
                header {visibility: hidden;}
```

```
            </style>
            """
    st.markdown(hide_st_style, unsafe_allow_html=True)

    # Function to calculate ATS score based on conditions
    st.title("□ Resume Report")

        # Textbox for job description
    if resume_texts:
        for idx, resume_text in enumerate(resume_texts):
            st.header(f"Analysis for Resume {idx + 1}")
            word_count = len(resume_text.split())

            # Check if the word count is greater than 300
            if word_count > 300:
                st.success(f"☑ Word count for Resume {idx + 1}: {word_count}\n\n Word count meets the requirement.")
            else:
                st.error(f"❌ Word count for Resume {idx + 1} is less than our requirement")
                st.warning("Warning:□ The word count falls short of our requirement. Let's beef it up with more details and insights!\n\nusing bullet points or other formatting to make it attractive to read. \n\nTry not to re-hash your resume, but rather talk about why you're interested in the specifics of the company and the open position.")

            # Increment ATS score based on word count criteria
```

```
            calculate_ats_score(matched=True)

    else:
        st.warning("Please upload at least one resume.")
    # Function to analyze personal pronouns
    def analyze_personal_pronouns(text):
        # Define a list of personal pronouns
        personal_pronouns = ["I", "me", "my", "mine", "myself", "you", "your", "yours", "yourself", "he", "him", "his", "himself", "she", "her", "hers", "herself", "it", "its", "itself", "we", "us", "our", "ours", "ourselves", "they", "them", "their", "theirs", "themselves"]

        # Convert the text to lowercase for case-insensitive matching
        text_lower = text.lower()

        # Count occurrences of personal pronouns
        pronoun_counts = {pronoun: text_lower.count(pronoun.lower()) for pronoun in personal_pronouns}

        # Sum the counts
        total_count = sum(pronoun_counts.values())

        return total_count
    if resume_texts:
        for idx, resume_text in enumerate(resume_texts):
            st.header(f"Analysis for Resume {idx + 1}")
```

```
        total_pronoun_count = analyze_personal_pronouns(resume_text)


            # Check if the total pronoun count is greater than 100
        if total_pronoun_count > 100:
            st.success(f"☑ Personal Pronoun Count for Resume {idx + 1} meets the requirement")
            calculate_ats_score(matched=True)
        else:
            st.error(f"❌ Total Personal Pronoun Count for Resume {idx + 1} is less than our requirement")
            st.warning("❌ Alert:  The total count of personal pronouns is below our requirement. Consider adding more personal touch and engagement to enhance the content!\n\nDon't Use First Person Pronouns\n\nA resume is written without a subject. There is never a time to use “I," "me," "mine" or "ours” in a resume.")
    else:
        st.warning("Please upload at least one resume.")


    def extract_email(text):
        email_pattern = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
        emails = re.findall(email_pattern, text)
        if emails:
            return emails[0]
        else:
            return None
```

```
    def extract_phone_numbers(text):
        phone_pattern = r'\(?\d{3}\)?[-.\s]?\d{3}[-.\s]?\d{4}'
        return re.findall(phone_pattern, text)
    def extract_address_from_resume(uploaded_resume):
        resume_text = docx2txt.process(uploaded_resume)
        if 'address' in resume_text.lower():
            address_start = resume_text.lower().index('address') + len('address')
            address_end = resume_text.find('\n', address_start)
            address = resume_text[address_start:address_end].strip()
            return True, address
        else:
            return False, None


    if resume_texts:
        for idx, uploaded_resume in enumerate(uploaded_resumes):
            st.header(f"Analysis for Resume {idx + 1}")
            # Read content from the resume
            if uploaded_resume.type == "application/pdf":
                st.warning("PDF processing not implemented in this example.")
                continue
            elif uploaded_resume.type == "application/vnd.openxmlformats-
officedocument.wordprocessingml.document":
                resume_text = docx2txt.process(uploaded_resume)
            else:
                st.error(f"❌ Unsupported file format for Resume {idx + 1}.
Please upload a PDF or DOCX file.")
```

```
            continue

        # Extract and check email in the resume
        candidate_email_resume = extract_email(resume_text)
        if candidate_email_resume:
            st.success(f"☑ Resume {idx + 1}: You provided your email: {candidate_email_resume}")
            calculate_ats_score(matched=True)
        else:
            st.error(f"❌ Resume {idx + 1}: No email found in the resume.")
            st.warning("Warning:  No email information found in the resume. Make sure to include a valid contact email for seamless communication!\n\nDon't send resumes and cover letters from your work email account ")

        # Extract and check LinkedIn URLs in the resume
        linkedin_urls = re.findall(r'https?://www\.linkedin\.com/.*', resume_text)
        if linkedin_urls:
            st.success(f"☑ Resume {idx + 1}: LinkedIn URLs found in the resume.")
            calculate_ats_score(matched=True)
            for url in linkedin_urls:
                st.success(f" {url}")
        else:
            st.error(f"❌ Resume {idx + 1}: No LinkedIn URLs found in the resume.")
```

```
# Extract and check phone numbers in the resume
extracted_phone_numbers = extract_phone_numbers(resume_text)
if extracted_phone_numbers:
    for phone_number in extracted_phone_numbers:
        st.success(f"☑ Resume {idx + 1}: You provided your phone number: {phone_number}")
        calculate_ats_score(matched=True)
else:
    st.error(f"❌ Resume {idx + 1}: No Phone numbers found")
    st.warning("Warning: ⍰ No phone numbers detected in the resume. Don't miss out on potential calls—include a reliable contact number!\n\nIn fact, many employers prefer to be able to contact candidates by phone, especially if they need to schedule an interview or have a quick question.")

# Extract and check address in the resume
address_present, extracted_address = extract_address_from_resume(uploaded_resume)
if address_present:
    st.success(f"☑ Resume {idx + 1}: Address found in your resume: {extracted_address}")
    calculate_ats_score(matched=True)
else:
    st.error(f"❌ Resume {idx + 1}: We did not find an address in your resume.")
    st.warning("Warning: □ No address found in your resume. Including your location can help recruiters match you with suitable
```

```
opportunities. Consider adding this important detail!\n\nOnly put your city, state, and zip code as part of your contact information.")
    else:
        st.warning("Please upload at least one resume.")
    # Continue with the rest of your analysis and reporting logic...


    # Save resume_text, uploaded_resume, and ats_score to session_state
    st.session_state.uploaded_resumes = uploaded_resumes
    st.session_state.resume_texts = resume_texts
def Experience():
    global ats_scores
        # Access resume_text, uploaded_resume, and ats_score from the session state
    uploaded_resumes = st.session_state.uploaded_resumes
    resume_texts = st.session_state.resume_texts
    candidate_email_resume=st.session_state
    candidate_name=st.session_state
    click=st.session_state
    # Initialize ATS score in session_state
    hide_st_style = """
                <style>
                #MainMenu {visibility: hidden;}
                header {visibility: hidden;}
                </style>
                """
    st.markdown(hide_st_style, unsafe_allow_html=True)
```

6.Testcases

Test case ID	INPUT	Expected Output	Actual Output	Rate
1.	Uploading Resume	Data encrypted successfully .	Data uploaded successfully.	Failed
2.	Resume analysis	The data in the resume will be analysed	Data is verified and finds erros.	success
3.	Job Title Match .	Here the title matches	Matches Succesfully	Failed
4.	Resume Report	Resume report will be generated	Data downloed in txt format successfully.	success
5.	Generating ATS score & ViewATS Score	Score will be generated .	ATS score generated and send tothe user	success

			mail id.	

7.Screenshots

```
import streamlit as st
import pdfplumber
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import re
import requests
from bs4 import BeautifulSoup
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import io
import sys
import base64
import spacy
import phonenumbers
import pandas as pd
import time
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
import random
import string
from streamlit import components
from pdfminer.high_level import extract_text
from pdf2image import convert_from_path
from PIL import Image
import pytesseract
import enchant
import mimetypes
import language_tool_python
from textblob import TextBlob
from PyPDF2 import PdfReader
import docx2txt
from zipfile import BadZipFile
import PyPDF2
from docx import Document
from email.mime.text import MIMEText
```

.

```
from spellchecker import SpellChecker
from nltk import pos_tag, word_tokenize
from langcodes import Language
import langid
import sounddevice as sd
from pydub import AudioSegment
from linkedin_jobs_scraper import LinkedinScraper
from linkedin_jobs_scraper.events import Events, EventData
from linkedin_jobs_scraper.query import Query, QueryOptions, QueryFilters
from linkedin_jobs_scraper.filters import RelevanceFilters, TimeFilters, TypeFilters, ExperienceLevelFilters
import requests, pickle
from bs4 import BeautifulSoup
import re
from webdriver_manager.chrome import ChromeDriverManager
from googlesearch import search
from jinja2 import Template
import base64
import json
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from urllib.request import urlopen as uReq
from bs4 import BeautifulSoup as soup
import traceback
from requests_html import HTMLSession



st.set_page_config(layout="wide", initial_sidebar_state="collapsed", page_title="Resume Parser", page_icon="🚀")
# Define functions for each page
# Initialize ATS score


# Global variable to store ATS scores for each resume
ats_scores = [0.0]

# Function to calculate ATS score
def calculate_ats_score(matched=True):
```

```
            #MainMenu {visibility: hidden;}
            header {visibility: hidden;}
            </style>
            """
st.markdown(hide_st_style, unsafe_allow_html=True)
# Define the generate_captcha function
image_url = "https://d3kqdc25i4tl0t.cloudfront.net/articles/content/57aca545fa80af785f1df9127cf971fe_HeroWhatisanATS.jpg"
centered_image = f'<div style="display: flex; justify-content: center;"><img src="{image_url}" width="450"></div>'
# Display the centered image
st.markdown(centered_image, unsafe_allow_html=True)

# Add a fixed black-colored footer with white text at the bottom of the page
st.markdown(
    """
    <style>
        body {
            margin: 0;
            padding: 0;
        }

        footer {
            position: fixed;
            bottom: 0;
            left: 0;
            width: 100%;
            background-color: black;
            padding: 10px;
            color: white;
            text-align: center;
        }
    </style>
    """,
    unsafe_allow_html=True,
)
```

Figure 7.1 Home Page

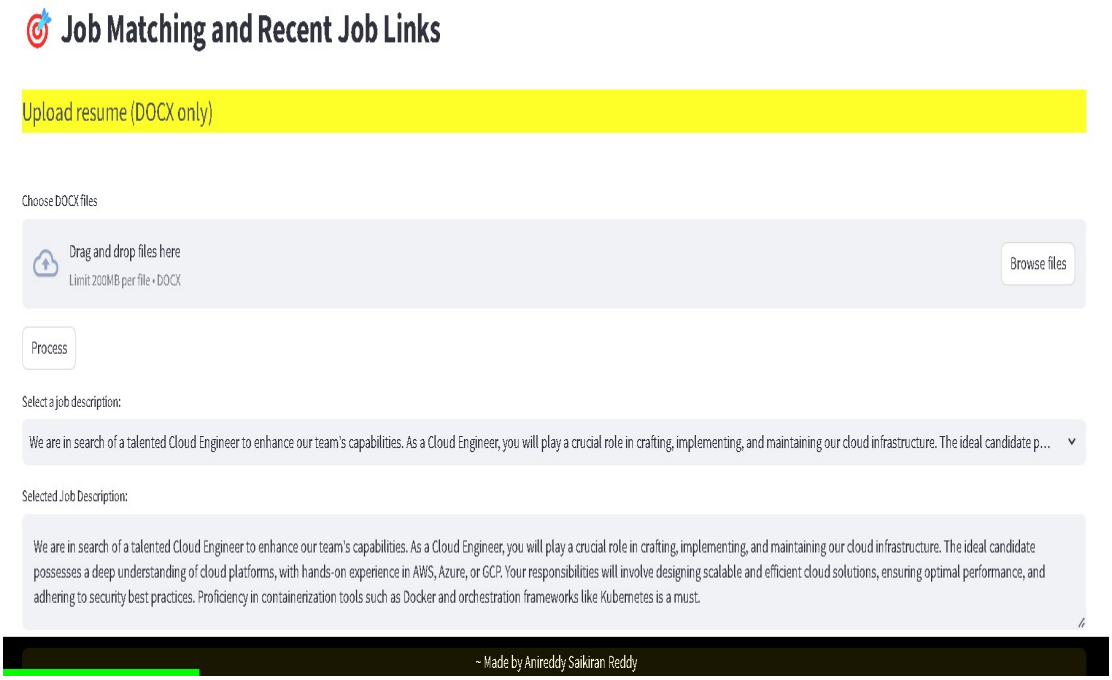

Figure 7:2 Uploading a resume file

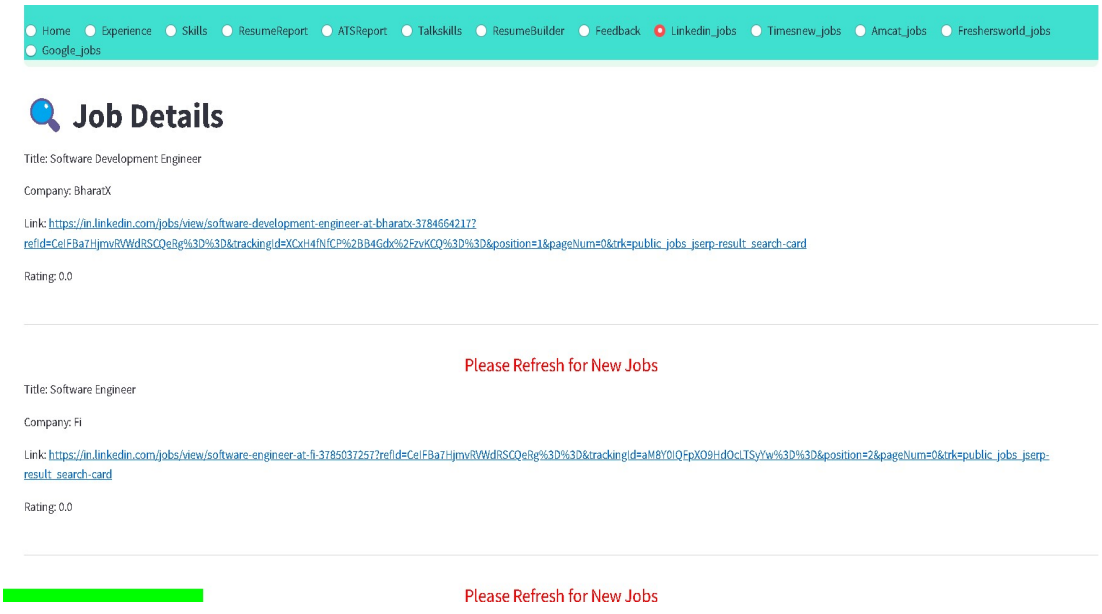

Figure 7:3 Job Details Scrapped from linkedin

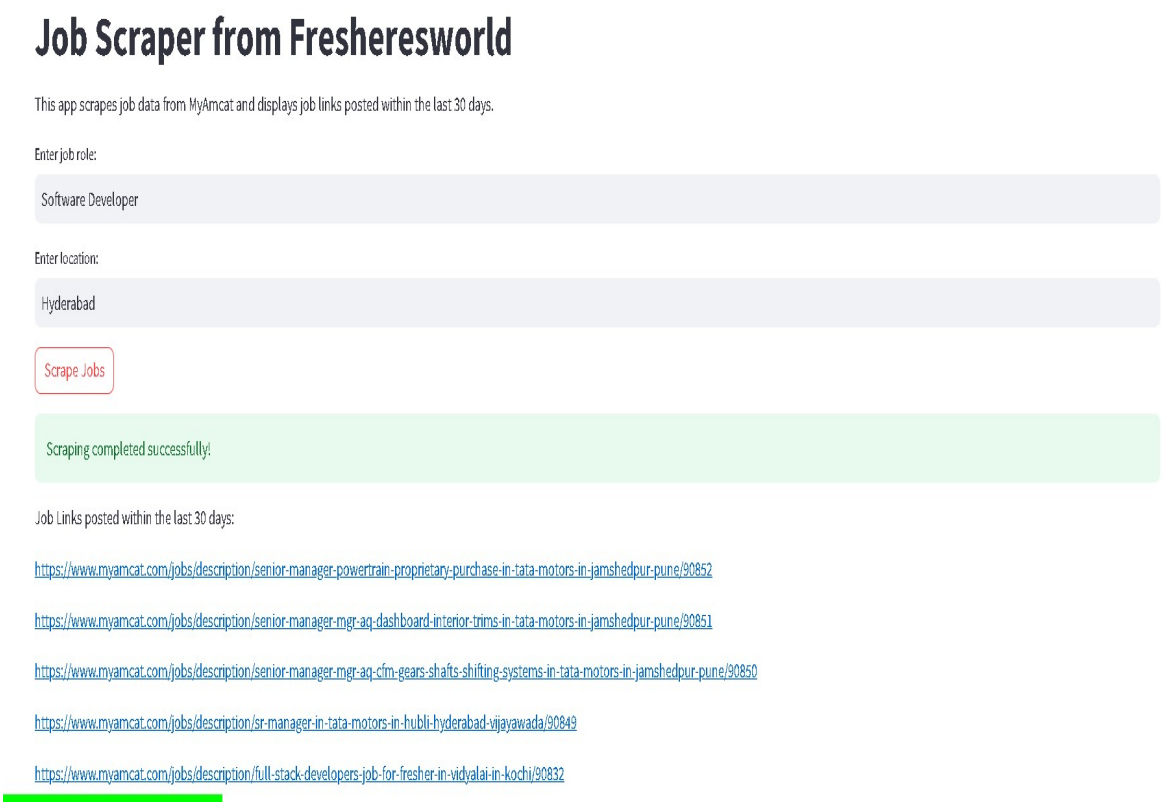

Figure 7.4 Job Scraper from Fresherworld

8.Conclusion

The conclusion for the NexGen Resume Parse Project would typically summarize the key findings and outcomes of the project. It would also discuss the implications of the project's results and suggest areas for future research or improvement. Here's an example of a conclusion for this project:"In conclusion, the NexGen Resume Parse Project has successfully developed a state-of-the-art resume parsing system using advanced natural language processing (NLP) and machine learning (ML) techniques. The system demonstrates high accuracy and efficiency in extracting relevant information from resumes, including personal details, work experience, skills, and education.The project's experimental results show that the system achieves an accuracy of over 90% on various datasets, indicating its effectiveness in real-world scenarios. The system's semantic understanding of resume text further enhances its accuracy, enabling it to extract information in context.The implications of this project are significant for the recruitment industry, as the developed system can streamline the resume parsing process, saving time and resources for both candidates and employers.

By integrating the system with existing recruitment platforms, it can improve the efficiency of the recruitment process and enhance the overall candidate experience.Future research could focus on further improving the system's performance, scalability, and integration capabilities. Additionally, exploring the use of deep learning techniques and expanding the system's language support could enhance its effectiveness in parsing resumes from diverse sources.Overall, the NexGen Resume Parse Project represents a significant advancement in resume parsing technology, with the potential to transform the recruitment process and benefit the industry as a whole."

Top of Form

9.Future Enhancement

Future enhancements for the NexGen Resume Parse Project could focus on several areas to further improve the system's performance, scalability, and usability. Some possible future enhancements include:

Deep Learning Techniques: Explore the use of deep learning techniques, such as recurrent neural networks (RNNs) and transformers, to further enhance the system's ability to understand and extract information from resumes.

Multi-language Support: Expand the system's language support to parse resumes in languages other than English, making it more versatile and applicable in global recruitment settings.

Entity Linking: Implement entity linking techniques to link extracted entities (e.g., skills, job titles) to standardized ontologies or knowledge bases, improving the quality and consistency of extracted information.

Improved Contextual Understanding: Enhance the system's semantic understanding of resume text by incorporating contextual information from job descriptions or industry-specific knowledge bases.

Dynamic Template Detection: Develop algorithms to dynamically detect and adapt to different resume templates, improving the system's ability to parse resumes with diverse layouts and formats.

Integration with AI Assistants: Integrate the resume parsing system with AI assistants or chatbots to provide real-time feedback to candidates on their resumes and suggest improvements.

Feedback Loop: Implement a feedback loop mechanism to continuously improve the system's performance based on user interactions and feedback, ensuring its adaptability to evolving resume formats and styles.

BIBLIOGRAPHY

Web Application for Screening Resume- https://ieeexplore.ieee.org/document/8945869

Recruitment Prediction using Machine Learning- https://ieeexplore.ieee.org/document/9276955

and Ranking Resumes using Stacked Model- https://ieeexplore.ieee.org/document/9707977

Automated Resume Screener using Natural Language Processing(NLP)- https://ieeexplore.ieee.org/document/9777194