Spring
2025

CS6320 Natural Language Processing

INSTRUCTOR: Tatiana Erekhinskaya

Job Match Genie Project Report

YouTube Link: https://youtu.be/gmSwrSxEqhE

GitHub Repo: https://github.com/IndraniBorra/JobMatchGenie/tree/main

REPORT BY,

INDRANI, BORRA IXB240007

KRISHNA TEJASWINI, PALETI KXP240029

VENKATA ABHIRAM, DACHARLA VXD240002

SAI CHARAN, PALVAI SXP240049

Table of Contents

Abstract

Job Match Genie is an intelligent job application assistant designed to automate and simplify the job search process using Natural Language Processing techniques. It features three integrated tools: an AI-powered resume parser that extracts essential details like contact information, skills, and a clean text version of the resume; a resume scoring module that evaluates how well a candidate's resume aligns with a given job description using semantic similarity; and a cover letter generator that creates personalized and professional letters tailored to each application. By combining these capabilities, the system empowers job seekers to apply more strategically, improve resume relevance, and generate application materials efficiently. The solution is scalable, user-friendly, and built to enhance job search outcomes through automation and AI.

System Overview

The objective of **Job Match Genie** is to create an AI-driven assistant that empowers job seekers by automating key steps in the job application process through Natural Language Processing (NLP). The system integrates three essential tools:

- A **resume parser** extracts structured information such as contact details and skills, performs job category prediction, and recommends relevant job roles based on the predicted category.
- A **resume scoring module** that analyzes the semantic similarity between a candidate's resume and a job description to assess alignment;
- A **cover letter generator** that produces customized, professional cover letters tailored to the specific job and candidate profile.

This project is designed to reduce the time, effort, and uncertainty involved in applying for jobs. It addresses common challenges faced by applicants—such as aligning their resume with job requirements and crafting effective cover letters—by providing actionable insights and automation. Ultimately, Job Match Genie aims to improve the quality of applications, boost candidate confidence, and support better job-market outcomes through intelligent NLP solutions.

Methodology

Resume Parser

This component of the project focuses on applying Natural Language Processing (NLP) techniques to automate the analysis and understanding of resumes. It performs three core tasks: resume text extraction, job category prediction, and personalized job recommendation. These tasks were implemented through a structured pipeline involving information extraction, vectorization, classification, and deployment using a Flask-based web interface.

Dataset Description

The Resume Parser utilizes two structured datasets sourced from Kaggle's publicly hosted Resume Datasets repository. These datasets were selected to support both resume categorization and job-role recommendation through various NLP techniques such as classification, information extraction, and similarity analysis.

clean_resume_data.csv

- **Source: Resume Dataset For Job Categorization**
- This dataset contains textual resumes and their associated job category labels. Each row represents a single candidate's resume along with a pre-labeled job field, making it suitable for supervised learning tasks.
- **Structure:**
    - Resume: Contains the raw text extracted from individual resumes.
    - Category: Labels indicating the job category or field associated with each resume (e.g., Data Science, HR, Software Engineering).**Usage in NLP Tasks:**
- **Text Classification:** The category labels serve as targets for supervised learning models to classify resumes into predefined job categories.
- **Skill Extraction:** The resume text is processed to identify and extract relevant skills using techniques like keyword matching and Named Entity Recognition (NER).
- **Contact Information Extraction:** Regular expressions are applied to extract contact details such as email addresses and phone numbers from the resume text.

jobs_dataset_with_features.csv

Source: **Resume Dataset for Job Recommendation**

Description:
This dataset consists of job roles along with detailed textual descriptions that capture the overall nature, responsibilities, and expectations associated with each role. Each entry provides a job

title and a corresponding feature string, which summarizes important role-specific characteristics in free-text format.

Structure:

- ➢ **Role**: The job title or domain (e.g., Software Engineer, Data Scientist).
- ➢ **Feature:** A single textual description representing the overall characteristics, requirements, and expectations of that role.

Usage in the System:

- ➢ **Job Role Profiling**: Acts as a repository of job role definitions that the system can refer to when recommending jobs.
- ➢ **Resume–Role Matching**: After extracting structured features from resumes, the feature text of each job role is compared using **TF-IDF vectorization** and **cosine similarity**.
- ➢ **Personalized Recommendations**: Helps in ranking job roles for a candidate based on how closely their resume aligns with the job's described features.

NLP Pipeline and Tasks

Named Entity and Information Extraction

- ➢ **PDF Resume Parsing:** PyMuPDF (fitz) was used to extract textual data from resumes in PDF format, preserving layout and structure better than traditional parsers.
- ➢ **Text Cleaning & Normalization:**
  - Lowercasing
  - Removal of punctuation and special characters
  - Stopword removal using NLTK
  - Tokenization for word-level processing
- ➢ **Named Entity Extraction (NER):**
  - **Skills:** Extracted using domain-specific keyword matching from a curated skill set
  - **Contact Info:** Email and phone numbers extracted using regular expressions

These steps convert unstructured resumes into structured data suitable for downstream NLP tasks.

Resume Categorization—Text Classification

This task predicts the category or field of a resume (e.g., HR, Data Science) using supervised text classification, a core NLP task. The process includes:

- ➢ **Feature Extraction:**
  - TF-IDF vectorization to transform resumes into numerical feature vectors

    - Consideration of unigrams and bigrams to capture context and phrase patterns
- ➢ **Model:** RandomForestClassifier from Scikit-learn, trained on TF-IDF vectors

This classification task enables automated resume labeling — a foundational application in resume screening systems.

Job Recommendation—Multi-Class Mapping via Text Similarity

To personalize user experience, a second classification pipeline was built to recommend specific job roles based on resume content. Although structured as a classification task, it aligns conceptually with information retrieval and similarity-based matching in NLP.

- ➢ **Vectorizer:** TF-IDF with different feature extraction configuration
- ➢ **Model:** Another Random Forest model trained on labeled resume–job title pairs
- ➢ **Objective:** Predict the most relevant job title from historical labeled data

This setup simulates intelligent job matching by interpreting the content of a resume using learned textual patterns.

Flask Integration and User Interface

The NLP models and feature extractors were deployed via a Flask application. Key features include:

- ➢ PDF resume upload and asynchronous processing
- ➢ Real-time parsing, skill detection, and classification
- ➢ Display of:
    - Extracted skills
    - Email and phone
    - Predicted job category
    - Job role recommendation
- ➢ A responsive frontend (resume_parser.html) offering a clean user experience with dynamic content rendering using JavaScript

Challenges and Fixes

Inconsistent PDF Structures

Resumes came in various formats with differing layouts and text encoding, which caused unreliable extraction using basic libraries.

Fix: Switched to **PyMuPDF (fitz)**, which provided more consistent and accurate parsing of resume content, preserving structural flow and text order.

Noisy Skill Detection
Initial keyword matching led to irrelevant or duplicated skill extractions due to overlaps in generic words and formatting noise.
Fix: Built a **refined NLP-based skill extractor**, using tokenization, filtering, and a curated skill list to improve accuracy and reduce false positives.

Token Length Overflow
Long resumes caused dimensional issues in TF-IDF vectorization, leading to memory inefficiencies and model degradation.
Fix: Applied **max token limits and vector dimensionality control** (e.g., limiting TF-IDF features), ensuring smooth processing and consistent input sizes.

Generalization on Rare Classes
Some job categories had very few examples, which led to overfitting and poor prediction on underrepresented roles.
Fix: Addressed class imbalance by **using stratified sampling, pruning rare classes**, and validating model generalization through **cross-validation**.

Deployment Issues (Pickle Loading Errors)
In early stages, the system failed to load models correctly due to inconsistent path handling and partial saves.
Fix: Ensured **model and vectorizer were saved and loaded together**, with consistent paths and tested using pickle and joblib integrity checks.

Unstructured Contact Information Extraction
Emails and phone numbers appeared in varied formats, making it difficult to extract consistently.
Fix: Used **robust regular expressions** that could adapt to multiple formats and clean noisy text before pattern matching.

Frontend Integration and User Feedback Delay
The UI initially had delayed or non-responsive interactions during resume uploads.
Fix: Implemented **asynchronous JavaScript** and improved result display flow for faster and more interactive parsing experience.

Resume Score

The Resume Score feature assesses the alignment of a candidate's resume with a job description using both keyword-based and semantic similarity techniques. This empowers users to identify areas for improvement and tailor their applications for enhanced success rates.

The pipeline extracts text from both the resume and the job description. It then computes:

- ➢ **TF-IDF Match Score** (keyword overlap).
- ➢ Sentence Embedding Similarity (semantic similarity using sentence-transformers/all-MiniLM-L6-v2).
- ➢ **Skill Match %** using dynamically extracted skills from the resume and required job skills.
- ➢ **Match Category** based on computed thresholds (Good Fit, Partial Fit, Poor Fit).

The feature also presents missing skills as recommendations.

Key Functions:

- ➢ compute_similarity_from_text() → TF-IDF similarity.
- ➢ compute_embedding_similarity() → Embedding similarity.
- ➢ extract_skills() → Skills identified using a small custom spaCy NER, backed by POS-based noun extraction.
- ➢ **Dynamic fallback logic** ensures that when certain models fail (e.g., no embedding available), other simpler methods still return a valid score.

Dataset & Preprocessing

Primary Dataset

We used the **cnamuangtoun/resume-job-description-fit** and **facehuggerapoorv/resume-jd-match** dataset from HuggingFace. This dataset provided labelled resume-job description pairs, which allowed initial training of machine learning models for match prediction, then dynamically evaluates similarity using user-uploaded resume and job description pairs.

Preprocessing Steps

- PDF or text input parsing using PyPDF2.
- Text cleaning, lowercasing, tokenization, and lemmatization.
- Sentence segmentation (implicitly by models).
- Embedding generation (all-MiniLM-L6-v2).
- Skill extraction (static CSV method initially, later improved to dynamic NLP extraction).

Trials on Pre-trained Models

We attempted training with:

- **distilbert-base-uncased**: Failed due to high memory requirements even after freezing layers.
- **bert-base-uncased**: Dropped early because of compute limits.
- **LoRA fine-tuned model** (shashu2325/resume-job-matcher-lora): Trained successfully but too resource-heavy for deployment.

Model Trails and Errors

Throughout the development of the Resume Score feature, I systematically explored multiple modelling approaches, balancing accuracy, scalability, and resource constraints. This iterative process involved both successes and failures, each contributing valuable insights to the final design.

1. TF-IDF Vectorizer (Sklearn): The initial baseline approach employed a **TF-IDF vectorizer** to measure keyword overlap between resume and job description text.
*Outcome***:** This method was computationally efficient, simple to implement, and provided clear, interpretable results. However, it lacked the ability to capture semantic similarity between terms, limiting its effectiveness when wording varied between documents.
*Decision***:** Retained as one component of the final scoring system for keyword-based similarity.

2. Sentence Embeddings (all-MiniLM-L6-v2): To address the limitations of TF-IDF, semantic similarity was introduced using the all-MiniLM-L6-v2 model from the Sentence Transformers library. This approach computes dense vector embeddings to capture meaning-level similarity between text pairs.
Outcome: The model performed well in capturing the semantic relationship between resumes and job descriptions, even when the wording differed significantly. It also offered an excellent trade-off between accuracy and computational speed.
Decision: Adopted as the primary method for semantic similarity in the production system.

3. LoRA Fine-Tuned Model (shashu2325/resume-job-matcher-lora): A Low-Rank Adaptation (LoRA) fine-tuned transformer model was trained using the ***cnamuangtoun/resume-job-description-fit*** dataset to optimize the resume-job matching task.
Outcome: The model demonstrated promising accuracy during testing and aligned well with the project's objectives.
Limitation: The model's resource requirements were excessive for real-time deployment in the web application. It was impractical given the available hardware and latency requirements.

Decision: Excluded from deployment. The experimentation was valuable and informed future directions.

4. DistilBERT (distilbert-base-uncased): Fine-tuning of the DistilBERT model was attempted as a compromise between the smaller sentence transformer models and full-scale BERT models.
Outcome: Multiple training attempts failed due to memory limitations, even after strategies such as freezing layers to reduce computational load.
Decision: Abandoned in favor of lighter, more scalable methods.

5. BERT Fine-Tuning: Direct fine-tuning of the bert-base-uncased model was also explored.
Outcome: Preliminary tests indicated strong performance potential. However, the model's computational demands exceeded the available resources.
Decision: Abandoned early in the experimentation phase.

6. SpaCy PhraseMatcher for Skill Extraction: The initial approach to skill extraction used SpaCy's PhraseMatcher in combination with a static CSV list of technical skills.
Outcome: The method was straightforward to implement and offered reasonable accuracy for known skills.
Limitation: The static nature of the skill list made it inflexible and incapable of identifying novel or multi-word skills.
Decision: Transitioned to dynamic extraction methods.

7. Static CSV Skills: A CSV-based skills matching system was implemented during early development phases.
Outcome: Enabled rapid prototyping and initial testing.
Limitation: Lacked coverage for dynamic skills and required manual updates.
Decision: Replaced with NLP-based dynamic extraction techniques.

8. SkillNER Library: Integration of the SkillNER library was attempted to provide more sophisticated and flexible skill recognition.
Outcome: Incompatibility issues arose due to version mismatches between SkillNER and the latest versions of spaCy. This resulted in critical errors that prevented stable operation.
Decision: Abandoned in favor of custom NLP-based skill extraction solutions.

9. SpaCy Custom NER (Small Model): A small, domain-specific SpaCy Named Entity Recognition (NER) model was trained to detect skills.
Outcome: The model provided effective recognition of technical skills while remaining lightweight enough for real-time use.
Decision: Adopted as the primary skill extraction method.

10. Part-of-Speech (POS) Tagging (Noun/Proper Noun Extraction): As a fallback, POS tagging was used to extract nouns and proper nouns when no skills were detected by other methods.
Outcome: Provided a reliable backup method that ensured the system could always produce some skill extraction results, even in edge cases.
Decision: Integrated into the final pipeline as a fallback mechanism.

Attempt	Method / Model	Why We Tried It	What Happened	Final Outcome
TF-IDF Vectorizer	sklearn	Baseline, interpretable	Worked well	Kept
Sentence Embeddings (all-MiniLM-L6-v2)	Semantic similarity	Captured meaning beyond keywords	Worked well	Kept
LoRAFine-tuned (shashu2325/resume-job-matcher-lora)	Specialized transformer	High accuracy	Too heavy for deployment	Dropped
DistilBERT (distilbert-base-uncased)	Smaller transformer	Expected lightweight option	Memory errors during training	Dropped
BERT Fine-tuning	Deep learning	Expected superior semantic performance	Resource limits stopped training	Dropped
SpaCy PhraseMatcher	Rule-based skill extractor	Easy to set up	Inflexible for dynamic skills	Dropped
Static CSV Skills	Hardcoded list	Quick startup	Not scalable or dynamic	Dropped
SkillNER	Dynamic skill extractor	Expected cutting-edge skill detection	spaCy version compatibility issues	Dropped
SpaCy Custom NER	Trained skill detector	Lightweight, domain-specific	Performed well	Kept
POS Tagging	Fallback extraction	Guaranteed minimal skill detection	Robust fallback	Kept

Table representation

Final evaluation metrics

The final Resume Score output was designed to be user-friendly and actionable. We reported five metrics: TF-IDF Match Score, Embedding Match Score, Skill Match%, Match Category, and Missing Skills. These allow both semantic and keyword-level evaluation, along with clear

feedback on which skills to improve in the resume. This multi-metric approach is inspired by real-world ATS (Applicant Tracking System) scoring models.

Metric	What it measures	Why it's useful
TF-IDF Match Score	Keyword overlap	Like ATS systems (Applicant Tracking Systems)
Embedding Match Score	Deep semantic similarity	Captures meaning even if wording is different
Skill Match %	% of required skills found	Very intuitive for users
Match Category	Overall evaluation (Good/Moderate/Poor)	Easy to interpret
Missing Skills	What's lacking	Gives actionable feedback

Flask Integration and User Interface

The Resume Score feature was integrated using Flask with a modular Blueprint (resume_score_bp) structure. Three key API routes powered the feature:

- /resume-score: Rendered the user interface.
- /extract-text: Extracted text from uploaded PDF files.
- /compare-text: Processed inputs to compute TF-IDF similarity, embedding similarity (using *all-MiniLM-L6-v2*), skill match percentage, and missing skills, returning JSON responses.

The frontend (resume_score.html) allowed users to input or upload resume and job description data. Results were dynamically displayed using AJAX, including similarity scores, skill match %, match category, and recommendations. The design prioritized ease of use and real-time feedback.

Cover Letter Generator

This component of the project automates the generation of basic cover letters by fine-tuning a T5-base transformer model on a curated dataset of resumes, job descriptions, and human-written cover letters. The approach followed a structured pipeline consisting of dataset preparation, input formatting, model fine-tuning, and deployment.

Dataset Description

The dataset used for this task is the publicly available ShashiVish/cover-letter-dataset hosted on Hugging Face. This dataset includes structured information about applicants and jobs, with the following columns:

- Job Title
- Preferred Qualifications
- Hiring Company
- Applicant Name
- Past Working Experience
- Current Working Experience
- Skillsets
- Qualifications
- Cover Letter (target output)

Each row represents a complete data instance with structured input features and a corresponding cover letter written in natural language.

Prompt Construction

To prepare the data for training the T5 model, each instance was converted into a natural language prompt and paired with its target cover letter. The input prompt concatenated key fields from the dataset as follows:

Generate cover letter:

Job Title: <job title>.
Preferred Qualifications: <preferred qualifications>.
Company: <hiring company>.
Past Experience: <past experience>.
Current Experience: <current experience>.
Skills: <skillsets>.
Qualifications: <qualifications>.

This approach allowed the model to learn a meaningful mapping from structured information to unstructured cover letter text.

Tokenization and Preprocessing

We used Hugging Face's T5Tokenizer to preprocess both the input prompts and the target cover letters. Each was tokenized with truncation and padding to a maximum sequence length of 512 tokens to stay within the model's limits. A custom tokenization function was applied to map each example into the format expected by the model:

- input_ids and attention_mask from the prompt
- labels from the target cover letter

These were converted into Hugging Face Dataset objects for both training and validation.

Model Fine-Tuning

The model used for training was t5-base, a text-to-text transformer. Fine-tuning was performed using Hugging Face's Trainer API with the following training configuration:

- Learning Rate: 1e-4
- Batch Size: 4
- Epochs: 5
- Max Length: 512
- Mixed Precision: Enabled (fp16=True)
- Evaluation Strategy: Every 500 steps

The model was trained on Google Colab with GPU acceleration. Mixed precision training was employed to improve performance and reduce memory consumption. After training, the model and tokenizer were saved locally under the directory t5_cover_letter_model/. The directory was then zipped and downloaded for use in the web application.

Flask Integration and Deployment

The trained model was integrated into a Flask-based web application. The backend was implemented using T5Tokenizer and T5ForConditionalGeneration from the Transformers library. Key features of the Flask app include:

- Upload or paste input support for resumes and job descriptions
- Tone selection: professional, academic, confident, experienced, or talented
- Real-time inference with model sampling (top_p=0.95, top_k=50, temperature=0.9)
- Clean HTML/CSS frontend with asynchronous PDF parsing

The app dynamically generates a tone-specific prompt based on user input and feeds it to the model. The output is then displayed directly in the browser.

Challenges and Fixes

During development, several challenges were encountered and addressed:

- **Input Truncation:** Long inputs caused token overflows. To handle this, we used truncation=True and max_length=512 in tokenization to safely clip content without losing structure.
- **Unstructured PDF Text Extraction:** Many resumes and JDs were uploaded as PDFs. Using PyMuPDF (fitz), we implemented robust server-side extraction that retained basic formatting.
- **Tone-Specific Prompt Crafting:** To support multiple tones in generation, tone-specific system prompts were manually written and injected into the input before passing to the model.
- **Training Instability on Colab:** GPU timeouts and resource limits in Colab were resolved by using smaller batch sizes and fewer epochs while monitoring GPU usage.
- **Download and Integration Errors:** After training, saving and reloading the model was prone to path issues. This was resolved by explicitly saving both the tokenizer and model together and verifying the load with from_pretrained().

Innovation or Creativity

Job Match Genie stands out through its integration of modern NLP techniques into a unified, user-centric platform. One of the most innovative aspects is the use of the T5 Transformer model for dynamic cover letter generation, enabling personalized and contextually accurate outputs based on both resumes and job descriptions.

Additionally, the project replaces traditional keyword-based matching with semantic similarity scoring using SBERT embeddings, allowing for a deeper understanding of candidate–job relevance. This shift from surface-level to context-aware matching significantly improves alignment accuracy.

Other creative contributions include

- ➢ A modular architecture separating parsing, matching, and generation logic for better scalability
- ➢ Regex-driven extraction pipelines tailored to handle diverse resume formats
- ➢ Custom thresholding logic to interpret similarity scores as intuitive labels like “Good Fit,” “Potential Fit,” and “Bad Fit,” enhancing user interpretability
- ➢ These innovations transform a routine task into an intelligent and automated experience, offering practical benefits to job seekers and showcasing the power of applied NLP.

Complexity Highlights

Several components of Job Match Genie presented noteworthy technical complexity:

- ➢ **Multi-Functional Resume Parser:** Designing a single parser that could not only extract text and contact information but also accurately predict job categories and recommend roles required integrating rule-based extraction with supervised machine learning models.
- ➢ **Semantic Similarity Matching:** Implementing Sentence-BERT for resume–job description comparison involved handling variable-length textual data, optimizing embedding performance, and designing a reliable thresholding system for classification (Good Fit, Potential Fit, Bad Fit).
- ➢ **Cover Letter Generation using T5:** Integrating a generative transformer model introduced challenges in input formatting, prompt engineering, and ensuring coherent output tailored to both resume and job description contexts.
- ➢ **Data Imbalance Handling:** Resume categorization required balancing highly skewed class distributions, which was addressed using resampling techniques to improve model generalization.
- ➢ **Pipeline Modularity:** Architecting the project into separate, yet connected, modules (extraction, classification, matching, generation) demanded careful input/output standardization and testing across stages.

These complexities highlight both the depth of technical implementation and the system design efforts that went into building a fully functional and intelligent job-matching assistant.

Limitations & Future Work

Limitations

- **Rule-Based Skill Extraction**
  The current system uses keyword matching and regular expressions for skill and contact information extraction. While effective, this approach may miss domain-specific or multi-word skills and may not generalize well to varied resume formats.
- **TF-IDF and Static Similarity**
  Job recommendation and similarity Scoring relies partly on TF-IDF and cosine similarity, which consider only surface-level term overlap and lack semantic understanding.
- **Limited Multilingual Support**
  The parser and generation modules currently support only English-language resumes and job descriptions, limiting global applicability.
- **Model Resource Constraints**
  Advanced transformer models (e.g., DistilBERT, full BERT, LoRA fine-tuned) were

tested but could not be deployed due to memory and runtime limitations on Colab or local environments.

- **Semantic Thresholds**
  Resume—JD semantic similarity scoring thresholds are currently heuristic and not fine-tuned based on labeled validation data, which may affect edge-case scoring accuracy.

Future Work

- **Contextual Skill and Entity Extraction**
  Replace rule-based extraction with transformer-based Named Entity Recognition (e.g., BERT + SpaCy) to identify technical and soft skills with greater contextual accuracy.
- **Layout-Aware Resume Parsing**
  Integrate layout-aware models like **LayoutLM** or **Donut** to better interpret the spatial structure of complex resumes, especially multi-column or visually rich formats.
- **Multilingual & Global Resume Support**
  Add translation pipelines or use multilingual models (e.g., XLM-RoBERTa) to parse resumes and generate content in different languages.
- **Semantic Matching with Fine-Tuned Models**
  Expand semantic similarity using fine-tuned models like SBERT or LoRA-adapted transformers, and incorporate section weighting (e.g., prioritizing experience or skills) during scoring.

- **Modular Model Selection in UI**
  Allow users to toggle between different generation models (e.g., T5, template-based, BERT-based) or resume scorers through the frontend for experimentation.

External Testing and User Feedback

To ensure the system's usability and performance beyond internal development, we conducted preliminary testing with **five users outside the project team**. These users interacted with the Resume Parser, Resume Score, and Cover Letter Generator modules and provided feedback on various aspects, including

- ➢ Clarity and responsiveness of the user interface
- ➢ Accuracy and relevance of skill extraction
- ➢ Interpretability of resume–JD match scores
- ➢ Tone and structure of generated cover letters

Based on their input, several improvements were made to the frontend, including better real-time output rendering and improved input validation. The feedback also influenced prompt refinement and the presentation of match results.

This external validation helped improve overall user experience and provided early insights into real-world expectations. Future development will expand this testing into more structured usability studies and A/B testing cycles.

Contributions & Self Scoring

Indrani Borra

She was solely responsible for developing the resume score component of the project. Her contributions spanned data preprocessing, NLP pipeline implementation, model integration, and front-end development.

Key Contributions:

- ➢ Designed and implemented the Resume Score component, integrating similarity scoring, skill matching, and recommendations.
- ➢ Preprocessed and analyzed the cnamuangtoun/resume-job-description-fit dataset.
- ➢ Experimented with and fine-tuned multiple models:
  - TF-IDF for keyword overlap.
  - all-MiniLM-L6-v2 for semantic similarity.
    Fine-tuned LoRA model (*shashu2325/resume-job-matcher-lora*).
  - Attempts with DistilBERT and BERT for deeper contextual matching.
- ➢ Developed the skill matching pipeline:
  - Transitioned from static CSV lists to dynamic spaCy custom NER.
  - Integrated fallback methods using POS tagging.
- ➢ Integrated model and scoring logic into a Flask Blueprint (resume_score.py)
- ➢ Developed resume_score.html with responsive design and dynamic result rendering.
- ➢ Explored SkillNER and fell back gracefully when compatibility became a blocker.
- ➢ Evaluated performance trade-offs between models and optimized for speed and scalability.
- ➢ Collected feedback from peers and iteratively improved user experience.

Self Scoring

Category	Score	Justification
Significant exploration beyond baseline	80	Integrated multiple NLP approaches, experimented with advanced models and overcame deployment constraints.
Innovation or Creativity	30	Created a hybrid scoring pipeline combining TF-IDF, embeddings, and dynamic skill matching with semantic fallback mechanisms.
Highlighted complexity (data/architecture/optimization)	10	Addressed embedding similarity, PDF parsing, text preprocessing challenges, and model scalability issues.
Lessons learned and potential improvements	10	Gained deep insights into real-world NLP deployment, model trade-offs, and efficient API design.
Exceptional visualization/diagrams/repo	10	Built a clean UI with clear output display and modular, production-ready backend code.
Testing outside the team (on 5 people)	10	Conducted usability testing and refined the feature based on feedback.
Earned money with the project	0	Purely academic and exploratory.

Dacharla Venkata Abhiram

He was solely responsible for designing and implementing the cover letter generation component of the project. His contributions included

- Selecting and analyzing the structured cover letter dataset from Hugging Face.
- Constructing multi-field natural language prompts suitable for T5 fine-tuning.
- Fine-tuning the t5-base model using Hugging Face's Trainer API with mixed precision on Google Colab.
- Developing preprocessing pipelines for input tokenization and formatting.
- Building the Flask web application and integrating the trained model.
- Implementing front-end functionality for resume and job description upload, text extraction, and tone selection.
- Creating dynamic, tone-specific prompts and ensuring output generation was accurate and stylistically consistent.

Self Score

Significant exploration beyond baseline	80	Fine-tuned a T5 model and deployed it in a real-world web app.
Innovation or Creativity	30	Added tone-based prompts and flexible PDF/text input handling.
Highlighted complexity (data/architecture/optimization)	10	Addressed truncation, prompt crafting, and model loading issues.
Lessons learned and potential improvements	10	Gained insights into deployment, inference tuning, and UX issues.
Exceptional visualization/diagrams/repo	10	Clean interface and well-organized GitHub repository.
Testing outside the team (on 5 people)	10	Collected feedback from 5 users to improve usability.
Earned money with the project	0	Not a commercial project.

Krishna Tejaswini Paleti

She was solely responsible for developing the Resume Parser component of the project. Her contributions spanned data preprocessing, NLP pipeline implementation, model integration, and front-end development.

Key Contributions:

- Preprocessed and analyzed two structured datasets (clean_resume_data.csv and jobs_dataset_with_features.csv) for classification and recommendation tasks.
- Designed and implemented skill and contact information extraction using token-based filters and regular expressions.
- Built a TF-IDF + Random Forest pipeline for job category classification and resume-based job role recommendation.
- Integrated both classification and recommendation models into a Flask web application.
- Developed a responsive front end using HTML/CSS and JavaScript for asynchronous resume upload and results display.
- Ensured model deployment stability by resolving loading issues and optimizing TF-IDF dimensions.
- Evaluated the model's performance and implemented improvements in user interaction flow based on testing.

Category	Score	Justification
Significant exploration beyond baseline	80	Built an end-to-end resume analysis system with classification and recommendation from scratch using NLP techniques.
Innovation or Creativity	30	Combined multiple NLP tasks in one pipeline, including TF-IDF based job matching and dynamic skill extraction.
Highlighted complexity (data/architecture/optimization)	10	Handled PDF variability, unstructured input, and optimized TF-IDF for long-form resumes.

Lessons learned and potential improvements	10	Gained experience in handling noisy real-world data, model integration, and enhancing user interaction.
Exceptional visualization/diagrams/repo	10	Developed a clean, interactive front end and a well-structured back end with modular code.
Testing outside the team (on 5 people)	10	Collected usability feedback from peers and refined front-end behavior based on suggestions.
Earned money with the project	0	Academic and exploratory; not monetized.

Sai Charan Palvai

Sai Charan Palvai contributed across both the NLP implementation and frontend development aspects of the project. While his primary responsibility was building an intuitive and responsive frontend interface, he also played an important role in experimenting with various NLP models during the development phase.

Key Contributions:

- ➢ Designed and developed the frontend of the application using HTML, CSS, and JavaScript, ensuring a responsive, clean, and interactive user experience across all modules.
- ➢ Implemented core UI elements, including file uploads, tone selection, and dynamic output rendering, fully integrated with Flask APIs.
- ➢ Focused on real-time responsiveness and user-friendly design, enhancing interactivity using AJAX and asynchronous workflows.
- ➢ Explored multiple NLP model architectures for text generation beyond the T5 model, contributing to model evaluation and comparison during early prototyping.
- ➢ Supported prompt formatting and integration of model outputs into the interface, ensuring consistency between user inputs and generated outputs.

- ➢ Assisted in validating model behavior and ensuring smooth backend–frontend communication.

Self Score

Category	Score	Justification
Significant exploration beyond baseline	80	Balanced frontend ownership with meaningful NLP contributions through model testing and integration.
Innovation or Creativity	30	Created a highly usable and responsive interface while exploring alternative approaches for text generation.
Highlighted complexity (data/optimization)	10	Navigated frontend–backend communication and handled UI-driven model inference display.
Lessons learned and potential improvements	10	Improved skills in both modern frontend design and NLP model experimentation.
Exceptional visualization/diagrams/repo	10	Produced a clean, interactive frontend and helped maintain organized project structure.
Testing outside the team	10	Conducted hands-on user testing to refine the interface based on real feedback.
Earned money with the project	0	The project was developed solely for academic and learning purposes.

Conclusion

This project successfully integrates NLP techniques to automate resume analysis, job matching, and cover letter generation. The system demonstrates practical real-world value by combining model fine-tuning, semantic similarity, and user-friendly deployment.

References

[1] Hugging Face, “Transformers Documentation.” [Online]. Available: https://huggingface.co/docs/transformers
[2] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” 2019. [Online]. Available: https://www.sbert.net/
[3] Flask, “Flask Documentation.” [Online]. Available: https://flask.palletsprojects.com/
[4] Scikit-learn Developers, “scikit-learn: Machine Learning in Python.” [Online]. Available: https://scikit-learn.org/stable/
[5] spaCy, “Industrial-Strength Natural Language Processing in Python.” [Online]. Available: https://spacy.io/
[6] PyMuPDF Developers, “PyMuPDF Documentation.” [Online]. Available: https://pymupdf.readthedocs.io/
[7] ShashiVish, “Cover Letter Dataset,” Hugging Face Datasets. [Online]. Available: https://huggingface.co/datasets/ShashiVish/cover-letter-dataset