THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Natural Language Processing
Final Project
October 15, 2024
Amir Jafari
Due: *Check the Syllabus*

Resume-Align

(a solution to align resume with jobs)

for the course

DATS 6312 ‘Natural Language Processing’

Fall 2024

Team Members:

Aman Jaglan

Aravinda

Harsha

Shrihan

I. INTRODUCTION:

Project Overview:

When we first started working on this project, our goal was to build a system that could help align candidate resumes with job descriptions and give feedback on how to improve it. Over time, the scope expanded significantly, and we became involved in designing multiple features: analyzing similarity between resumes and job descriptions, generating resumes, providing visualization tools, ranking resumes based on their suitability for a given job, recommending relevant jobs to candidates, and evaluating grammar and formatting in resumes. The final product integrated all these components into a Streamlit application, creating a comprehensive platform where users can upload their resumes, paste job descriptions, and then explore a host of functionalities such as similarity scoring, skill gap analysis, resume generation, and other visualization techniques.

Initially, the project was simply about finding a good model for similarity and feedback with grammar checks and exploring how to compare a resume to a job description. But as we began adding more capabilities—like generating formatted resumes based on instructions and identifying missing skills in a candidate's profile—we realized that each component influenced and complemented the others. For instance, the similarity score and feedback from skill gap analysis could guide improvements to the user to add missing skills, help highlight what needed to be added or improved in the resume. This interplay eventually led us to develop a more holistic system rather than just a single model or script.

II. PROJECT DATASETS:

Resume Dataset

Source and Composition

- **Dataset Name:** "InferencePrince555/Resume-Dataset"
- **Platform:** Hugging Face
- **Total Entries:** Over 30,000 unique resumes
- **Significance:** Provides a comprehensive and diverse collection of professional resume samples

Key Characteristics

- Represents a wide range of:
    - Professional backgrounds
    - Industries
    - Career levels
    - Skill sets
- Offers a robust foundation for training AI models in resume analysis and processing

Job Description Dataset

Overview

- **Total Job Descriptions:** More than 1.6 million entries
- **Scope:** Covers multiple job fields and industries
- **Comprehensive Coverage:** Represents a broad spectrum of professional opportunities

Dataset Significance

- Enables comprehensive job matching
- Provides extensive context for skill mapping
- Supports advanced recommendation algorithms
- Allows for in-depth labor market analysis

Combined Dataset Potential

The combination of these two extensive datasets creates a powerful resource for:

- AI-driven resume parsing
- Intelligent job recommendation systems
- Skill gap analysis
- Career development insights

III. PROJECT OBJECTIVES

1. Resume Generation

Develop an advanced system capable of generating comprehensive and professional resumes tailored to individual user profiles and industry requirements.

2. AI-Powered Platform Development

Create an intelligent technological platform that leverages artificial intelligence to enhance the resume creation, evaluation, and job matching processes.

3. Resume Quality Evaluation

Implement sophisticated algorithms to assess and score resume quality, providing actionable insights to users for improving their professional documentation.

4. Resume Ranking Mechanism

Design a robust ranking system that can effectively compare and prioritize multiple resumes based on predefined criteria and job requirements.

5. Resume Parsing Optimization

Streamline the process of extracting and organizing critical information from resumes, ensuring efficient and accurate data interpretation.

6. Similarity Measurement

Develop advanced techniques to measure and analyze the similarity between resumes, enabling more precise job matching and skill assessment.

7. Intelligent Job Recommendations

Create a recommendation engine that provides personalized job suggestions based on individual resume content, skills, and professional background.

8. Skill Gap Analysis

Implement a comprehensive analysis tool that identifies and highlights potential skill gaps, offering users guidance for professional development and career growth.

Strategic Significance

These objectives collectively aim to transform the traditional approach to resume management and job searching, leveraging cutting-edge AI technology to provide more intelligent, efficient, and user-centric solutions.

IV. DATA PREPROCESSING AND PARSING

Pipeline for Data Extraction

The data extraction pipeline is responsible for the end-to-end process of extracting, parsing, and storing data, such as resumes, from various sources. The key components of this pipeline include:

The **DataExtractorProject** folder, which serves as the main project directory and contains all the necessary settings and configurations for the data extraction process. This includes references to the IDE (Integrated Development Environment) used for the project, the virtual environment for managing dependencies, and various executables and configuration files required for the virtual environment.

The **Data** section is where the extracted and processed data is stored. This includes the **Processed** folder, which holds the final processed data, such as the extracted resumes. The **Resumes** folder contains the processed resumes, while the **dataextractor** folder includes the core extraction modules. The **parsers** folder holds the utilities responsible for parsing job descriptions and resumes, and the **utils** folder contains various utility modules used throughout the project.

Pipeline for Resume and Jobscore

The pipeline for resume and jobscore processing is designed to handle the end-to-end workflow of processing resumes and job descriptions, as well as generating job scores. The key components of this pipeline include:

The **ResumeJobScore** folder, which serves as the main project directory for this workflow. Similar to the data extraction pipeline, it includes references to the IDE, virtual environment, and various configuration files.

The **data** folder, which stores the raw data, such as job descriptions and processed resumes. The **job_descriptions**subfolder holds the raw job descriptions, while the **processed** subfolder contains the parsed and processed resumes.

The **logs** folder, which stores log files related to the preprocessing tasks.

The **src** folder, which holds the source code for the preprocessing modules. This includes scripts for processing job descriptions (**job_desc_processor.py**), processing resumes (**resume_processor.py**), tokenizing text data (**tokenizer.py**), and utility modules (**utils**). Additionally, the **model_training.py** script handles the training of machine learning models, and the **pipeline.py** script executes the full preprocessing pipeline.

The **main.py** script is the entry point for running the overall preprocessing pipeline.

Parsing job description data:

In the initial phase of our resume recommendation project, we developed a robust data preprocessing system designed to transform raw job description data into a structured, machine-readable format. The primary challenge was to convert a complex CSV file containing job listings into a clean, consistent JSON structure that could be easily utilized by subsequent project components.

Technical Approach

The parsing solution was implemented using a modular Python approach, centered around a JobParser class that encapsulates multiple data transformation techniques. The parsing process involves several key steps:

1. Data Validation and Column Checking The parser first validates the input CSV file, ensuring the presence of essential columns such as 'Job Id', 'Experience', 'Job Title', and 'Skills'. This validation prevents processing incomplete or incompatible datasets and provides immediate feedback about data quality.
2. Experience Parsing A dedicated method was created to extract and standardize experience requirements. Using regular expression matching, the parser can intelligently interpret experience ranges like "5 to 15 Years", converting them into a structured dictionary with minimum and maximum years of experience.
3. Skill Extraction The skill parsing mechanism handles various input formats, converting comma-separated skill strings into clean, standardized lists. This ensures consistency in skill representation across different job descriptions.
4. Description Enrichment Beyond simple text storage, the parser adds value by computing additional metadata for each job description, such as:
    - Total text length
    - Extraction of key terms using predefined pattern matching
    - Preserving the full original text for potential future analysis

Error Handling and Robustness

Recognizing the variability and potential inconsistencies in real-world job description data, we implemented comprehensive error handling:

- Try-except blocks to gracefully manage parsing failures
- Logging of specific parsing errors without halting the entire process
- Flexible parsing that can handle missing or malformed data

Output and Transformation

The final output is a JSON file containing a list of structured job descriptions. Each job entry includes normalized fields like:

- Unique job identifier
- Job title
- Experience requirements
- Required skills
- Detailed responsibilities

Parsing user input data:

The initial approach to resume parsing relied on traditional regular expression-based techniques, which quickly revealed significant limitations. Resume formats are inherently diverse and unstructured, presenting a complex parsing challenge:

- Format Variability: Resumes come in countless layouts, fonts, and structures
- Inconsistent Information Placement: Skills, experiences, and projects can appear in unpredictable locations
- Semantic Nuances: Capturing the contextual meaning of professional experiences requires more than pattern matching

The regular expression method suffered from critical drawbacks:

- High false-negative rates

- Inability to understand context
- Brittle parsing that broke with minor format changes
- Difficulty handling complex, multi-line text sections
- Poor performance across different resume styles (academic, professional, creative)

Solution: Parsing using Google Gemini API

To overcome these limitations, we developed a sophisticated parsing solution leveraging Google's Gemini generative AI model. This approach transforms resume parsing from a rigid pattern-matching problem to an intelligent, context-aware extraction process.

Key Technical Components

1. Robust Text Extraction
    - Implemented multi-layered text extraction using PyPDF2
    - Added encoding detection with chardet to handle various file formats
    - Created a fallback mechanism for text reading to ensure maximum compatibility
2. Advanced Parsing Strategy
    - Designed detailed, structured prompts for the Gemini API
    - Created a standardized JSON output format with predefined keys
    - Implemented flexible parsing for different resume sections
3. Error Handling and Reliability
    - Added comprehensive exception handling
    - Implemented regex-based JSON extraction from AI responses
    - Provided fallback mechanisms for parsing failures

Parsing Workflow

1. Document Preprocessing
    - Text cleaning to remove unnecessary whitespace
    - Normalization of text content
    - Removal of non-ASCII characters for consistency
2. AI-Powered Extraction
    - Send resume text to Gemini with a structured parsing prompt
    - Request JSON-formatted output with specific, predefined keys
    - Handle various resume sections dynamically
3. Structured Output The parsing generates a comprehensive JSON with sections like:
    - Personal Information
    - Skills
    - Work Experience

    ○ Projects
    ○ Education

Technical Innovations

The Gemini-powered approach offers several advantages:

- Contextual Understanding: AI comprehends semantic meaning beyond simple pattern matching
- Adaptive Parsing: Works across diverse resume formats
- Structured Output: Consistent JSON format for downstream processing
- Scalability: Easily extensible to handle more complex parsing requirements

Future Integration

This parsing mechanism is strategically designed for future components of our project:

- Skill gap analysis
- Resume scoring using BERT
- Personalized job recommendations

V. RESUME QUALITY CHECK: GRAMMAR AND FORMATTING MODEL

Introduction

The Grammar and Formatting Check Model is a robust solution that leverages AWS Bedrock's advanced AI capabilities to analyze and improve the quality of resumes. The model extracts text from PDF resumes, identifies significant grammatical and formatting issues, and generates actionable recommendations to enhance professionalism and clarity. By integrating text extraction, AI-driven analysis, and structured output parsing, the model ensures a comprehensive evaluation of resume quality.

Pipeline Workflow

1. **Initialization**
   The **BedrockResumeAnalyzer** class is initialized using AWS credentials, which include access keys, session tokens (if applicable), and the AWS region. The initialization also configures logging to track execution flow and debug errors, ensuring a seamless operation of the pipeline.
2. **Text Extraction**
    - **Method**: extract_text_from_pdf
      This step uses the PyPDF2 library to extract and clean text from PDF resumes. Text from all pages of the document is combined, and errors—such as missing files or issues during text extraction—are logged for transparency and debugging.
3. **Resume Analysis**
    - **Method**: analyze_resume_text
      A detailed prompt is constructed to analyze the extracted text. This step focuses on identifying:
    - The top grammatical errors.
    - Key formatting inconsistencies.

    - Suggestions for improvement.
    - Numerical scores (ranging from 0 to 100) for grammar and formatting. The prompt is sent to AWS Bedrock's language model (e.g., Claude) for analysis, with robust error handling for reliable performance.

4. **Output Parsing**
    - **Method**: _parse_resume_analysis
      The AI model's response is parsed using regular expressions to extract:
    - Grammar and formatting scores.
    - Lists of grammatical errors and formatting issues.
    - Constructive recommendations.
      A helper function, _extract_section, ensures the structured parsing of specific sections for clarity and organization.

5. **Main Function**
    - **Function**: analyze_resume
      This orchestrates the entire pipeline by:
    - Initializing the analyzer with AWS credentials.
    - Extracting text from the resume file.
    - Analyzing the text for grammar and formatting issues.
    - Returning structured results, including scores and improvement suggestions.

Key Features

- **Comprehensive Analysis**:
  Identifies and quantifies grammatical errors and formatting inconsistencies, providing numerical scores to assess resume quality.
- **Actionable Insights**:
  Offers clear and concise recommendations to improve the presentation and professionalism of resumes.
- **Error Handling**:
  Employs robust exception handling mechanisms for file-related and processing errors, ensuring pipeline reliability.
- **AWS Bedrock Integration**:
  Utilizes state-of-the-art AI technology (Claude) to deliver accurate, detailed, and focused analysis.

Example Output

- **Scores**:
    - Grammar Score: 85/100
    - Formatting Score: 90/100
- **Top Grammatical Errors**:
    - Incorrect verb tense in the work experience section.
    - Missing punctuation in project descriptions.

- **Formatting Issues**:
  - Inconsistent use of bullet points.
  - Uneven spacing between sections.
- **Recommendations**:
  - Use consistent formatting for bullets and spacing.
  - Revise verb tense for clarity and accuracy.

VI. JOB RECOMMENDATION SYSTEM

Introduction

The development of an intelligent job recommendation system involved overcoming technical limitations and refining approaches to better capture the semantic relationships between job descriptions and resumes. Initial methods relying on traditional TF-IDF modeling revealed significant shortcomings in understanding contextual and semantic meaning, prompting a shift toward more advanced techniques.

Limitations of Initial Approaches

1. TF-IDF Shortcomings

- Based primarily on word frequency, making it inadequate for capturing contextual meaning.
- Treated words as discrete, independent entities, ignoring relationships between terms.
- Unable to identify or model semantic connections, limiting the accuracy and relevance of recommendations.

Transition to Word2Vec: A Semantic Breakthrough

To address the limitations of TF-IDF, the system transitioned to Word2Vec, an advanced word embedding technique that represents words as dense vector spaces. This pivot enabled the model to better understand semantic relationships and align resumes with job descriptions more effectively.

Key Technical Innovations

1. Advanced Text Preprocessing

- Implemented sophisticated text cleaning techniques, including:
  - Removal of stop words and special characters.
  - Standardization of text representation.
  - Consistent tokenization across job descriptions.
- Ensured clean and uniform input data for downstream processing.

2. Semantic Embedding Strategy

- Utilized the skip-gram model to learn efficient word embeddings.
- Created semantic embeddings from various job-related text sources, including:
  - Job titles.
  - Required skills.

    - Qualifications.
    - Responsibilities.

3. Optimization Techniques

- Leveraged multi-core processing to handle computational demands.
- Reduced training epochs to accelerate computation.
- Developed caching mechanisms to store embeddings and job vectors for reuse.

Performance Optimization Challenges

The initial implementation of Word2Vec faced significant performance bottlenecks, including:

- Recommendation times of 5–6 minutes per job match.
- High computational overhead during vector computations.
- Inefficient processes for embedding generation and similarity calculations.

Caching and Performance Improvements

To address these challenges, a multi-layered optimization strategy was implemented:

1. Embedding Caching

- Saved pre-computed Word2Vec models for reuse.
- Allowed embeddings to be shared across multiple recommendation runs.
- Significantly reduced initialization time.

2. Job Vector Precomputation

- Used joblib for efficient serialization of job vectors.
- Precomputed and cached job vectors to minimize redundant computations.
- Reduced recommendation times from several minutes to mere seconds.

3. Vectorization Strategy

- Implemented document vectorization by averaging word vectors.
- Handled out-of-vocabulary words gracefully with fallback mechanisms.
- Ensured robust handling of unseen tokens using zero-vector defaults.

Recommendation Algorithm

The recommendation system combines multiple scoring mechanisms for comprehensive matching:

1. Semantic Similarity

- Calculated cosine similarity between resume and job vectors.
- Captured deep semantic relationships between the text representations.

2. Skill Match Ratio

- Compared skills listed in the resume with job requirements.

- Provided a quantitative metric for skill alignment.

3. Hybrid Scoring

- Combined semantic similarity and skill match ratio using weighted scoring.
- Ensured recommendations were both contextually relevant and skill-aligned.

VII. RESUME-JOB SIMILARITY CHECKER

In today's competitive job market, efficiently matching resumes to job descriptions is crucial for both recruiters and job seekers. To address this need, we've developed a script that enhances resume-job description matching and to improve semantic comparison between resumes and job descriptions, we used section-based parsing, semantic embeddings from Sentence Transformers, and weighted similarity scoring to produce a nuanced and meaningful similarity score between a candidate's resume and a job posting.

Initially, the process involved extracting and preprocessing text from PDF documents using the PyPDF2 library. The raw text was standardized through a cleaning function that converted it to lowercase and removed non-alphabetic characters. Embeddings were generated using a pretrained BERT model from the Hugging Face Transformers library, and cosine similarity was calculated between the embeddings of the entire resume and job description. However, this approach had several drawbacks. It was computationally expensive due to the heavy BERT model, lacked scalability by treating entire documents as monolithic texts, and didn't allow for weighing different sections based on their importance.

To overcome these limitations, the we refined it to include enhanced parsing and section-based analysis. We used a parser that deconstructs documents into structured fields. These fields include skills, responsibilities, and qualifications, which are treated independently. This section-based analysis addresses the issue of unequal importance across different parts of the documents. The module converts resume and job description files into JSON format. Each section is clearly delineated to enable targeted comparison. This modular approach enhances reusability and reduces redundant code when integrating updates or additional fields.

Focus on Specific Fields: Each section (e.g., "skills") contains domain-relevant entities and phrases, ensuring that the matching algorithm captures specific requirements without being influenced by irrelevant text.

Parsing and Structuring Documents

```
from resume_job_description_parser import process_documents
from sentence_transformers import SentenceTransformer, util
import numpy as np
model = SentenceTransformer('all-MiniLM-L6-v2')

```

To encode the parsed sections, we utilized the all-MiniLM-L6-v2 model from the Sentence Transformers library. This choice was motivated by the need for a model that balances efficiency and semantic richness. Cosine similarity is calculated between the embeddings of corresponding sections from resumes and job descriptions. A weighted scoring mechanism is used to reflect the importance of each section. This model provides lightweight embeddings that retain the necessary semantic information, significantly improving computational performance compared to the bulky BERT model.

The core of the script is function, which computes a weighted similarity score between the parsed resume and job description:

```
def calculate_weighted_similarity(resume, job_description, weights):
    # Extract and concatenate text from each relevant section
    resume_skills = " ".join(resume.get('skills', []))
    resume_responsibilities = " ".join(
        [" ".join(exp.get('responsibilities', [])) for exp in resume.get('work_experience', [])]
    )
    resume_qualifications = " ".join([edu.get('degree', '') for edu in resume.get('education', [])])

    job_skills = " ".join(job_description.get('required_skills', []))
     job_responsibilities = " ".join(job_description.get('responsibilities', []))
```

```
job_qualifications = " ".join(job_description.get('qualifications', []))

# Generate embeddings for each section
resume_embeddings = {
    'skills': model.encode(resume_skills, convert_to_tensor=True),
    'responsibilities': model.encode(resume_responsibilities, convert_to_tensor=True),
    'qualifications': model.encode(resume_qualifications, convert_to_tensor=True)
}
job_embeddings = {
    'skills': model.encode(job_skills, convert_to_tensor=True),
    'responsibilities': model.encode(job_responsibilities, convert_to_tensor=True),
    'qualifications': model.encode(job_qualifications, convert_to_tensor=True)
}

# Calculate cosine similarities between corresponding sections
similarities = {
    field: util.cos_sim(resume_embeddings[field], job_embeddings[field]).item()
    for field in weights.keys()
}

# Compute the weighted average similarity score
weighted_similarity = sum(similarities[field] * weights[field] for field in weights)
return weighted_similarity

```

The above method performs the following-

- **Extraction of Relevant Sections**: It extracts text from relevant sections of both the resume and job description. By focusing on specific fields like skills, responsibilities, and qualifications, we ensure that the comparison is both targeted and meaningful.
- **Generating Embeddings**: For each section, the function generates semantic embeddings using the Sentence Transformer model. These embeddings are numerical representations that capture the semantic essence of the text.
- **Calculating Similarities**: It computes the cosine similarity between the embeddings of corresponding sections from the resume and job description. Cosine similarity measures how similar two vectors are, with a value closer to 1 indicating high similarity.
- **Weighted Averaging**: Recognizing that different sections have varying levels of importance, the function applies predefined weights to each similarity score. The final similarity score is a weighted average, reflecting the overall match according to the specified priorities.

The process involves several key steps. First, it extracts text from relevant sections of both the resume and job description, focusing on specific fields like skills, responsibilities, and qualifications to ensure that the comparison is both targeted and meaningful. Next, for each section, the function generates semantic embeddings using the Sentence Transformer model. These embeddings serve as numerical representations that capture the semantic essence of the text. Following this, it computes the cosine similarity between the embeddings of corresponding sections from the resume and job description. Cosine similarity measures how similar two vectors are, with a value closer to 1 indicating high similarity. The similarities between corresponding sections are calculated using cosine similarity, which measures the cosine of the angle between two vectors in a multidimensional space. This metric is effective for determining how similar two pieces of text are in terms of their content and context. The script then applies a weighted scoring mechanism to these similarities. Weights are assigned to each section based on their importance to the job role—for instance, skills might be given a weight of 0.5, responsibilities 0.4, and qualifications 0.1. The final similarity score is computed as the weighted sum of these individual similarities.

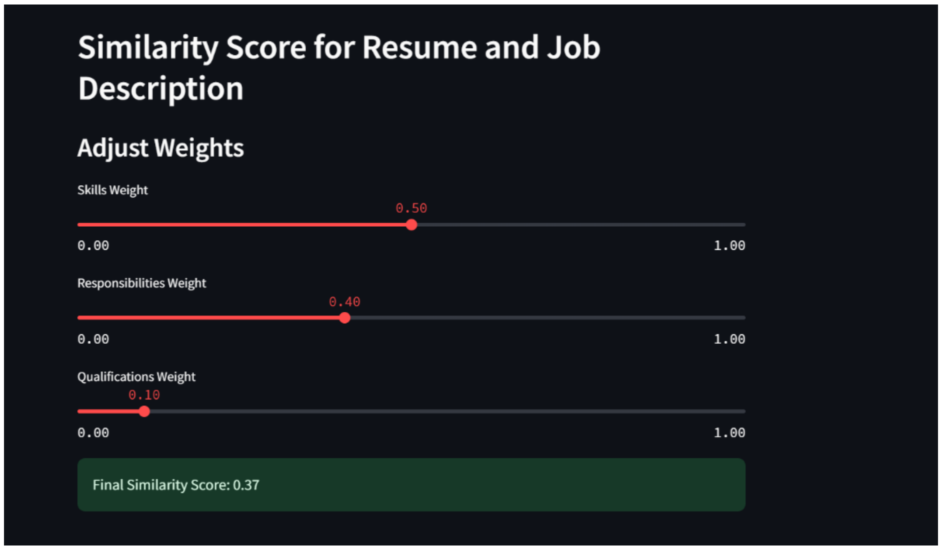
The enhancements made in the refined solution address the limitations of the initial approach. By parsing documents into sections, the script allows for a more nuanced analysis that considers the varying importance of different parts of the resume and job description. The use of the lightweight Sentence Transformer model improves computational efficiency, making the process faster and more scalable. The weighted similarity scoring mechanism provides flexibility in emphasizing certain sections over others based on the specific requirements of the job, thereby yielding more meaningful results.

In terms of the experimental setup, the system processes real-world resume and job description data without the need for explicit model training, as it utilizes pre-trained embeddings. Performance is evaluated based on the similarity scores computed, which can be used to rank candidates according to their fit for a given job description. The simplicity of the model and the independent processing of each resume-job description pair help prevent overfitting—a common issue in supervised learning setups.

Hyperparameter tuning is minimal, with the primary parameters being the weights assigned to each section. Adjusting these weights allows stakeholders to tailor the system according to the importance of various sections for different roles. For instance, a technical position might prioritize skills more heavily, while an academic role might value qualifications.

The modular and reusable design of the script enhances maintainability and scalability. By decoupling the parsing logic and the similarity calculation, each component can be updated or replaced independently. This design facilitates integration with other systems and allows for easy extension to include additional sections or different types of documents.

Overall, the refined script offers a robust solution for resume-job description matching by combining advanced NLP techniques with practical design considerations. It delivers improved accuracy and efficiency, making it a valuable tool for recruiters and job seekers alike in navigating the complexities of the job market.



Similarity Score for Resume and Job Description

Adjust Weights

Skills Weight: 0.50 (0.00 – 1.00)

Responsibilities Weight: 0.40 (0.00 – 1.00)

Qualifications Weight: 0.10 (0.00 – 1.00)

Final Similarity Score: 0.37

Fig 1. Similarity score result

VIII. SKILL GAP ANALYZER

The Skill Gap Analysis tool was designed to compare resumes against job descriptions and provide actionable feedback on missing skills, work experience, and educational qualifications. This tool uses a combination of semantic similarity and direct matching to identify gaps, leveraging NLP techniques powered by spaCy and pre-trained language models for natural language understanding.

Initially, we tried a very straightforward similarity approach to identify skill gaps, using simple cosine similarity between TF-IDF or embedding vectors of the resume and the job description. However, we soon realized that a more semantic approach was needed. The cosine similarity approach was too literal and did not capture nuanced relationships between words. For example, “Python development experience” in the job description might not match exactly with “skilled in Python for web applications” in the resume due to differing phrasing, even though semantically they mean something similar.

The tool employs **spaCy's en_core_web_md model**, which provides pre-trained word embeddings and similarity scoring. These embeddings map textual inputs to a dense vector space, allowing for the comparison of semantic similarity between phrases.

```
spacy.cli.download("en_core_web_md")
nlp = spacy.load("en_core_web_md")

```

The similarity scoring uses cosine similarity between vectors. If the cosine similarity exceeds a threshold (e.g., 0.8), the text is considered a match. This method is particularly effective for identifying semantically similar phrases, even when exact wording differs.

The core algorithm parses JSON data for skills, work_experience, and education fields, comparing them between the resume and job description. For skills, a simple list comparison identifies missing entries. For experience and education, semantic similarity is calculated using spaCy.

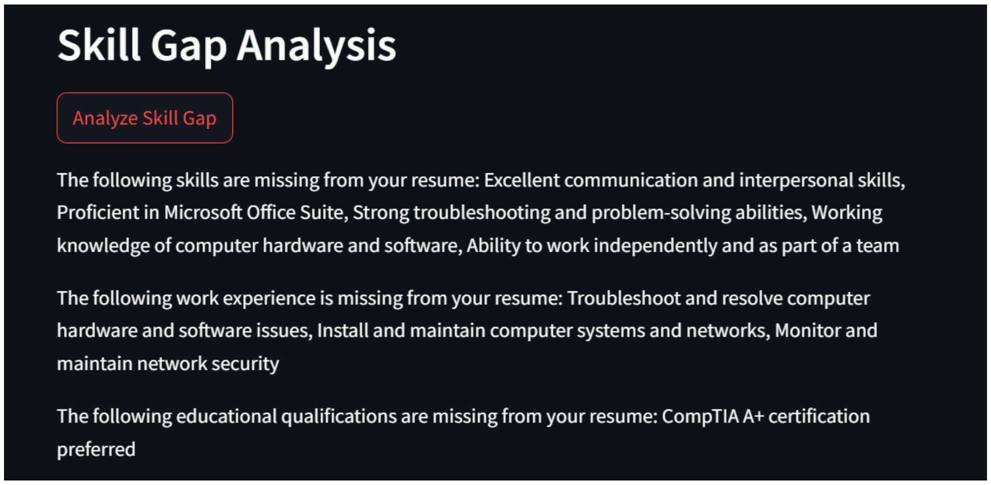
```
missing_skills = [skill for skill in job_skills if skill not in resume_skills]

experience_gap = []
for job_exp in job_experience:
    job_exp_doc = nlp(job_exp)
    if not any(job_exp_doc.similarity(nlp(exp)) > 0.8 for exp in resume_experience):
        experience_gap.append(job_exp)

```

For hyper parameters we adjusted,

- **Similarity Threshold**: Tuned at 0.8 after experimentation to balance precision and recall. Higher thresholds were too strict, missing semantically similar phrases; lower thresholds were overly permissive.
- **Experience Matching**: The model checks for phrase similarity using embeddings. Adjusting the threshold dynamically based on responsibilities' importance could improve results.

This tool demonstrates the power of NLP in personalized, context-aware feedback, paving the way for more sophisticated applications in recruitment and career development. The Skill Gap Analysis tool effectively utilizes NLP techniques to bridge the gap between candidate resumes and job descriptions. By leveraging spaCy's pre-trained embeddings and semantic similarity scoring, the tool identifies missing skills, work experience, and educational qualifications with high accuracy. This enables users to receive actionable feedback on how to tailor their resumes for specific job roles.

Future improvements could involve integrating advanced contextual embeddings from models like BERT or Sentence-BERT, which could provide deeper insights into complex job descriptions. Additionally, adding visual aids and assigning weights to critical requirements would further enhance the tool's usability and effectiveness. Overall, the Skill Gap Analysis tool exemplifies the potential of NLP in streamlining recruitment processes and empowering job seekers to present themselves more effectively in competitive job markets.



Skill Gap Analysis

The following skills are missing from your resume: Excellent communication and interpersonal skills, Proficient in Microsoft Office Suite, Strong troubleshooting and problem-solving abilities, Working knowledge of computer hardware and software, Ability to work independently and as part of a team

The following work experience is missing from your resume: Troubleshoot and resolve computer hardware and software issues, Install and maintain computer systems and networks, Monitor and maintain network security

The following educational qualifications are missing from your resume: CompTIA A+ certification preferred

Fig 2. Skill gap analyzer example

IX. VISUALIZATIONS

To bridge the gap between raw technical output and actionable insights, a suite of advanced visualizations was developed to make complex textual data relationships more intuitive and accessible. These visualizations were instrumental in fostering informed decision-making for aligning resumes with job descriptions, providing a clear and impactful representation of the data. The methodologies, rationale, and implications of each visualization type are outlined below.

Word Clouds: Thematic Representation of Prominent TermsWord clouds were utilized to highlight the most significant terms in resumes and job descriptions, ranked by their TF-IDF (Term Frequency-Inverse Document Frequency) scores. TF-IDF, a widely used statistical measure in NLP, assesses the importance of a term in a document relative to its frequency in the corpus. This visualization provided an intuitive overview of thematic alignment between resumes and job descriptions. For instance, the terms were displayed with varying font sizes and color-coded to reflect their relative importance, offering immediate visual cues about overlapping themes or gaps. This insight proved invaluable for identifying key areas of alignment or divergence, guiding both candidates and recruiters in prioritizing essential skills and attributes.

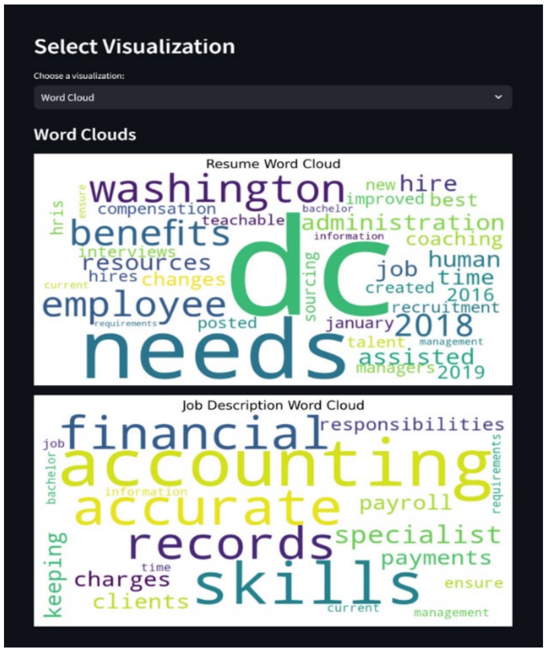
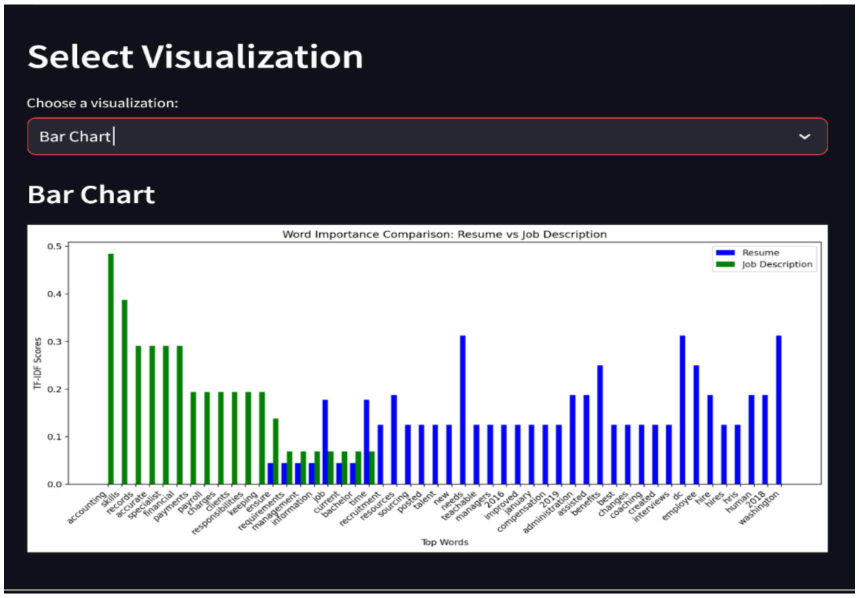

Select Visualization
Choose a visualization:
Word Cloud
Word Clouds
Resume Word Cloud
washington dc needs benefits employee resources administration hire human job time 2018 2016 2019 compensation teachable improved best bachelor coaching information interviews hires changes current requirements posted sourcing created recruitment january talent management assisted managers hris ensure new
Job Description Word Cloud
financial accounting accurate records skills responsibilities payroll specialist payments ensure charges clients keeping time job bachelor information requirements current management

Fig 3. Word Cloud

Bar Charts: Comparative Analysis of Term Importance

Bar charts were employed to facilitate a direct comparison of term relevance between resumes and job descriptions. Using TF-IDF scores, the charts prioritized critical terms, enabling a granular view of the most relevant features. These visualizations highlighted terms that appeared more frequently in job descriptions compared to resumes, effectively identifying areas where resumes might require enhancement. This quantitative comparison not only made the analysis actionable but also ensured stakeholders could focus on bridging the most significant gaps.



Select Visualization

Choose a visualization: Bar Chart

Bar Chart

Fig 4. Bar Charts

Heatmaps: Quantifying Sectional Similarity

Heatmaps visualized the cosine similarity scores between corresponding sections of resumes and job descriptions. This approach quantified alignment across critical areas such as skills, work experience, and education. The color-coded representation, with scores ranging from 0 (no alignment) to 1 (perfect alignment), provided an intuitive understanding of thematic matches. Higher scores indicated strong alignment, while lower scores pinpointed areas needing improvement. Heatmaps were particularly impactful for comparing multiple resumes against a single job description or vice versa, making them a powerful tool for recruiters and candidates to identify strengths and weaknesses.

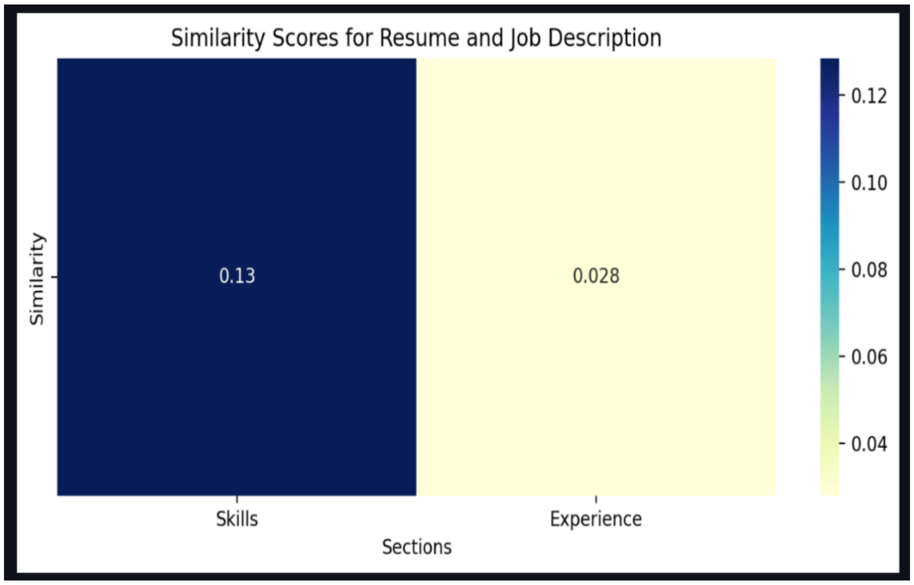


Fig 5. heatmap

Venn Diagrams: Mapping Common and Unique Skills

Venn diagrams offered a high-level perspective on the overlap between skills listed in resumes and job descriptions. This visualization mapped common skills, unique skills in job descriptions (highlighting areas for improvement), and unique skills in resumes (showcasing additional strengths). The clear demarcation of these areas provided actionable insights, enabling candidates to focus on acquiring missing skills and recruiters to recognize unique value propositions. Venn diagrams were particularly effective for fostering focused discussions on alignment strategies, making them an essential visualization for this analysis.

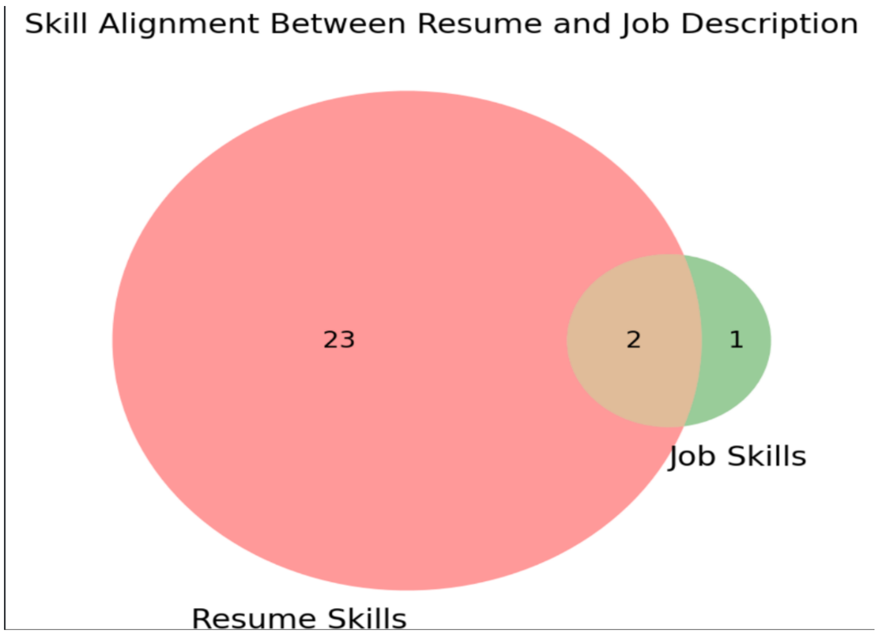


Fig 6. Venn Diagram

The visualizations were created using a combination of Python libraries and frameworks to ensure both aesthetic appeal and functional robustness. Word clouds and Venn diagrams were generated using the WordCloud and Matplotlib's Venn module, respectively. Bar charts and heatmaps were implemented with Matplotlib, Seaborn, and Plotly, while Scikit-learn facilitated TF-IDF vectorization and cosine similarity computations. These tools enabled interactivity, enhancing user engagement and making the insights more actionable.

The integration of visualizations significantly enhanced the interpretability of analytical outputs. By pinpointing specific areas for improvement, candidates could tailor their resumes more effectively, addressing skill gaps and emphasizing relevant expertise. Recruiters benefited from data-driven hiring decisions, focusing on candidates with strong thematic alignment. Overall, the visualizations bridged the gap between raw analytical data and actionable insights, empowering stakeholders with a clear and impactful understanding of the alignment between resumes and job descriptions. This approach demonstrated the potential of combining NLP-driven analytical tools with advanced visualizations for real-world applications.

X. RESUME RANKING

This feature enhances the recruitment process by providing a smart and efficient way to identify and rank the most relevant resumes against a given job description. By transforming traditional keyword-based methods into a context-aware semantic matching system, it ensures that each candidate’s suitability is judged holistically, capturing nuances that simple text matching might miss.

The first implementation used a vanilla BERT model to generate embeddings for entire resumes and job descriptions. While this represented a significant advancement over static embeddings like Word2Vec or GloVe, it introduced complexity in handling long documents. Additionally, the results, though improved over keyword matching, did not fully leverage domain-specific signals such as job-role relevance scores.

To improve ranking accuracy, label-based relevance scores were introduced and fed into an XGBoost model. This model learned from examples of “good” and “poor” resume-job matches, providing more nuanced rankings. In parallel, chunking strategies were employed to handle long resumes exceeding BERT’s token limit, ensuring no valuable content was lost. These changes improved accuracy but increased complexity and introduced dependencies. Overfitting risks arose if the amount of labeled training data was limited, and the pipeline began to feel cumbersome.

Next, a more sophisticated text splitting technique using LangChain's RecursiveCharacterTextSplitter was integrated. By splitting resumes into overlapping chunks, more contextually cohesive embeddings were produced, resulting in even richer semantic representations. However, this approach increased processing time and complexity, making the system less agile for real-time use cases. Each iteration pushed accuracy higher, but the pipeline was becoming resource-intensive and complex.

The underlying model uses a BERT-based language representation to generate embeddings for both job descriptions and resumes. In contrast to earlier word-level embedding models like Word2Vec or GloVe, BERT embeddings are contextually aware, capturing the meaning of words in relation to their surrounding text. Once the embeddings are generated, the system uses cosine similarity to measure semantic alignment between a resume’s vector and the job description’s vector.

To ensure high-quality input, the resumes are first converted from their original PDF format into raw text and then cleaned to remove extraneous characters, ensuring that the textual input to the model is noise-free. The normalization and embedding generation process produces a uniform representation of textual content, enabling reliable comparisons.

In operational terms, this feature:

- **Data Input**: The system expects inputs in PDF format for resumes. The extract_text_from_pdf function reads and extracts text from PDFs, ensuring that the downstream NLP pipeline receives raw textual data.
- **Preprocessing**: The clean_text function prepares the extracted text for embedding by lowercasing and removing non-alphanumeric characters. This reduces noise and ensures more stable embedding generation.
- **Embedding Generation**: The embed_texts function applies the tokenizer and model to the cleaned text. It processes each document, generates embeddings using BERT, and stores them as NumPy arrays. This step transforms unstructured text into numeric vectors, enabling mathematical comparison.
- **Normalization and Similarity Computation**: Although the code includes a normalize_vectors function (not directly used in the given snippet), normalization is a common step to ensure each embedding vector has unit length. The rank_resumes function then applies cosine_similarity to measure the closeness of each resume’s embedding to the job embedding.
- **Ranking**: After computing similarity scores, the system sorts the resumes by their similarity values in descending order. The rank_resumes function returns a list of resumes along with their similarity percentages, ranks, filenames, and content previews. This allows users to quickly identify the top candidates that align most closely with the job description’s requirements.

As a result, recruiters receive a concise and actionable list of ranked candidates, saving time and improving the decision-making process. This semantic approach to resume ranking yields higher accuracy than traditional keyword-matching techniques, allowing organizations to focus on the most qualified candidates from the outset.

XI. RESUME GENERATION

The resume generation feature aims to create realistic, structured resumes based on user instructions, effectively turning a general-purpose text generation model into a specialized resume drafting tool. By leveraging state-of-the-art Transformer models and a carefully curated dataset, it produces coherent, contextually relevant resumes aligned with a given job role or prompt.

This feature relies on a dataset sourced from Hugging Face’s InferencePrince555/Resume-Dataset. The dataset pairs instructions (e.g., “Generate a Resume for a Software Engineer Job”) with corresponding resume texts. Each instruction corresponds to a particular job role, providing the model with explicit guidance on the type of resume to generate. The dataset includes roughly 52 distinct prompts (job types), and some roles, like Software Engineer, are overrepresented with thousands of examples, while others (e.g., BPO roles) are more sparse. On average, the resumes are about 900 words long, offering substantial context for the model to learn domain-specific terminology, common resume structures, and stylistic elements.

The initial attempts at resume generation using T5 models (T5-small and T5-base) faced several challenges. Firstly, there was a token length limitation, as both T5-small and T5-base have a maximum input length of about 512 tokens. This often resulted in resumes exceeding the limit, leading to truncation and the loss of important contextual details, which made the generated resumes incomplete or incoherent. Secondly, the outputs were frequently repetitive and unstructured. Without careful fine-tuning, T5 produced repetitive phrases and struggled with proper resume formatting, failing to maintain headings, bullet points, or other structural cues over longer sequences. Lastly, the quality of the generated content did not meet the desired standards for professionalism or clarity. The outputs felt generic and repetitive, lacking the ability to effectively reflect nuanced roles and skill sets.

To overcome these shortcomings, the approach shifted to **BART-base**, a model known for its effectiveness in text generation tasks and its ability to handle up to 1024 tokens. The increased token limit allowed feeding more contextual information into the model, crucial for resumes that frequently exceed a few hundred words. However, simply switching models was not enough. Initially, the model model_name = "facebook/bart-base" was fine-tuned using a basic pipeline with default parameters, as shown below:

```
tokenizer = BartTokenizer.from_pretrained(model_name)
model = BartForConditionalGeneration.from_pretrained(model_name)

training_args = TrainingArguments(
    output_dir="./resume_generator_bart",
    evaluation_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=5,
    save_strategy="epoch",
    logging_steps=50,
    save_total_limit=2
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer
)

```

With this initial setup, the model began to produce longer and more coherent resumes compared to T5, but they still displayed noticeable repetition and formatting issues. In response, a series of adjustments were made:

1. **Increased Training Epochs and Adjusted Learning Rate**: By raising the number of training epochs from 5 to up to 15 or even 20, the model had more exposure to the training data, allowing it to better learn the mapping from instructions (like "Generate a Resume for a Software Engineer") to properly structured output. Additionally, experimenting with a higher learning rate (e.g., 5e-4) sometimes improved convergence and the diversity of generated text. Mixed-precision training (fp16=True) was introduced to speed up training and potentially improve performance by fitting more data into GPU memory without sacrificing training stability.

2. **Data Preprocessing and Cleanup**: The original dataset contained raw HTML tags, invalid spans, and excessive whitespace. Such noise can confuse the model and lead to poorer output quality. A custom preprocessing pipeline was implemented:

   - **HTML & Tag Removal**: Regular expressions stripped out unwanted HTML tags.

   - **Whitespace Normalization**: Multiple consecutive spaces were reduced to single spaces.

- **Section Extraction**: Common headers like "Professional Summary," "Experience," and "Skills" were identified and used to segment the resume into meaningful sections. This helped the model learn the structural patterns of resumes, improving the logical flow and readability of generated content.

3. **Inference-Time Decoding Strategies**: Adjusting the generation parameters at inference time significantly influenced the style and coherence of the resumes. For example:

```
outputs = model.generate(
    inputs["input_ids"],
    max_length=512,
    top_k=50,         # Allows sampling from the top 50 candidates
    top_p=0.95,       # Nucleus sampling for diversity
    temperature=5.0, # High temperature for more variability
    repetition_penalty=3.0, # Penalize repetitive phrases
    num_beams=4,      # Beam search for more coherent sequences
    no_repeat_ngram_size=3, # Avoid repeating certain n-grams
     early_stopping=True
)

```

top_k & top_p sampling: Introduced diversity in the generated text, preventing the model from always choosing the top 1 most likely next token, which could lead to bland, repetitive output.

Temperature: A higher temperature encouraged creativity but required careful balancing; too high and the text became chaotic.

Repetition Penalty & no_repeat_ngram_size: These parameters specifically targeted repetitive lines and phrases, a common issue in early generations.

num_beams: Beam search evaluated multiple candidate sequences in parallel, improving coherence and reducing nonsensical segments.

Postprocessing:
After generation, a final pass on the output included:

- Removing duplicate lines.
- Reformatting section headers for consistency.
- Adding bullet points to lists of skills or responsibilities. These steps further improved the professional appearance and clarity of the resumes.

Results and Performance

Through these iterative refinements, the resume generator's outputs improved significantly. Early attempts with T5 yielded nearly unusable text, while the initial BART-base runs were marginally better but still lacked refinement. After careful tuning of hyperparameters, decoding strategies, and data preprocessing, the generated resumes became more coherent, better structured, and contained fewer repetitive segments.

Quantitative evaluation of the resume generator's performance relied, in part, on the BLEU (Bilingual Evaluation Understudy) score, a metric originally designed to assess the quality of machine-translated text by comparing it against one or more reference translations. In essence, BLEU measures how many n-grams (continuous sequences of words) in a generated text appear in a set of reference texts. Scores range from 0 to 1, with higher values indicating greater lexical similarity. A BLEU score closer to 1 suggests that the model's output closely matches the reference in terms of word choice and order, while a score near 0 means the text is largely dissimilar.

Although BLEU was not specifically created for tasks like resume generation—where creativity, readability, and logical structure matter more than a strict overlap of words—it still provides a useful, objective starting point. Early attempts yielded BLEU scores hovering between 0.1 and 0.2, which is close to random and signifies that the generated resumes bore little resemblance to the reference texts. Through iterative refinements—improved model selection, hyperparameter tuning, decoding adjustments, and thorough data preprocessing—the BLEU scores rose substantially to around 0.4–0.7. While still not perfect, this increase indicated that the generated resumes became more lexically similar to human-written references. In other words, even though BLEU does not capture all aspects of quality, the jump in scores showed that the model was learning to produce text that better aligned with the lexical style and vocabulary of the authentic resumes it was trained on.

Conclusions and Next Steps

By choosing a more suitable model (BART-base), extending the input length limit, performing thorough preprocessing, carefully tuning training and generation parameters, and applying postprocessing steps, the resume generation feature was transformed from a raw prototype into a functional component capable of producing human-readable, structured resumes. While the generated resumes may not always be perfect or entirely factual (the model is, after all, generating text rather than accessing a database), they now provide a strong foundation for further enhancements. Potential next steps include:

- Integrating domain-specific instructions or templates.
- Applying more robust evaluation metrics aligned with hiring manager feedback.
- Exploring more advanced models like Flan-T5 or instruction-tuned LLMs for even more natural language understanding and generation quality.

In summary, starting from initial attempts with T5 and facing challenges with truncation and repetition, the progression to a finely-tuned BART-based system demonstrated substantial improvements. The combination of preprocessing, fine-tuning, strategic inference parameters, and final formatting steps all contributed to a more sophisticated, coherent, and user-friendly resume generation tool.

XII. CONCLUSION

The journey from initial concept to a fully integrated, AI-driven platform exemplifies how iterative development, domain-specific tuning, and strategic integration of multiple NLP techniques can transform the recruitment and job search experience. Starting as a simple idea to compare a resume to a job description, the project evolved into a holistic ecosystem—**Resume-Align**—capable of generating polished resumes, identifying skill gaps, scoring semantic similarity, providing actionable visualizations, ranking candidates against job criteria, and even recommending relevant job opportunities.

Key achievements include:

- **Resume Generation:** Transitioning from T5-based methods to a refined, BART-based system significantly improved the structure, coherence, and readability of generated resumes. Thorough preprocessing, hyperparameter tuning, and strategic decoding parameters elevated BLEU scores and enhanced the naturalness of the output.
- **Similarity Scoring and Skill Gap Analysis:** By employing advanced embeddings (Sentence-BERT, spaCy) and semantic similarity metrics, the project moved beyond keyword matching to context-aware evaluations. This enabled precise identification of missing skills and experience, empowering candidates to strengthen their profiles while giving employers a more accurate measure of fit.
- **Resume Ranking and Job Recommendations:** Leveraging semantic embeddings and domain-specific models ensured that resumes are ranked with increased accuracy and relevance. Combined with caching and optimization strategies, the recommendation process became efficient and scalable. Candidates can now be matched to suitable roles more quickly, while recruiters can focus on top-tier candidates.

- **Visualizations and Insights:** The suite of visual tools—word clouds, bar charts, heatmaps, and Venn diagrams—translated complex textual relationships and analytical outputs into intuitive, actionable insights. These visualizations not only guided candidates in improving their resumes but also helped recruiters quickly assess thematic alignments and pinpoint areas needing attention.
- **Quality Checks and Formatting Evaluations:** Integration with AWS Bedrock and other grammar-checking services provided real-time assessments of resume quality. By highlighting grammatical errors, inconsistent formatting, and providing structured recommendations, the system ensured that final outputs were both professional and polished.

Ultimately, the synergy of these components yielded a comprehensive solution that addresses the needs of multiple stakeholders. Job seekers benefit from tailored feedback, improved resume quality, and personalized job suggestions. Recruiters and hiring managers gain efficient tools to rapidly identify top talent aligned with job requirements, supported by clear, data-driven evidence.

While each module—be it skill gap analysis, resume generation, or similarity scoring—achieved notable improvements on its own, the true value emerges from their integration into a unified platform. The project sets a strong foundation for future developments, such as incorporating instruction-tuned LLMs for even more naturalistic generation, employing more granular evaluation metrics aligned with recruiter feedback, and enhancing the user experience with interactive dashboards.

In conclusion, **Resume-Align** stands as a proof of concept for how natural language processing and machine learning can be harnessed to streamline and enrich the recruitment process. By focusing on semantic understanding, contextual alignment, and actionable insights, the project contributes meaningfully to a more efficient, equitable, and informed job market.

REFERENCES

1. https://github.com/611noorsaeed/Resume-Screening-App/tree/main
2. https://github.com/lil-Ribhav-Bhatt1012/Resume-Matchers/tree/master
3. https://github.com/OmkarPathak/pyresparser
4. https://github.com/srbhr/Resume-Matcher
5. https://www.analyticsvidhya.com/blog/2021/06/resume-screening-with-natural-language-processing-in-python/
6. https://kartikmadan11.medium.com/building-a-job-description-to-resume-matcher-using-natural-language-processing-5a4f5