AI FOR JOB SKILL GAP ANALYSIS AND RECOMMENDING RESKILLING RESOURCES

TABLE OF CONTENTS

List of Figures

Chapter 1

1. INTRODUCTION

1.1 BACKGROUND

In today's fast-changing job market, increasing demand for specialized skills has created a competitive environment in which professionals have to stay ahead of industry trends. This dynamic landscape has brought forth the necessity of identifying skill gaps and acquiring targeted learning to bridge those gaps effectively. The traditional methods of skill assessment and career guidance fail to provide precise insights in most cases, are not tailored to the person's needs, and often lead to inefficient learning paths, delayed career progress, and mismatched resources. Generalized solutions offered by platforms such as online course providers and certification programs usually do not meet the particular skills required in the career of the candidate. Such systems rarely provide holistic learning experiences that address individual preferences, pace, and career aspirations. The key to providing a structured, yet personalized professional development journey for professionals is to employ data-driven AI-driven profiling with curative recommendations.

The proposed system employs the latest Natural Language Processing techniques, namely Bidirectional Encoder Representations from Transformers (BERT), to determine skill gaps through resume and job description analysis.The system, using cosine similarity, compares the candidate's existing skills with industry expectations and identifies the most relevant areas for improvement. This project thus integrates these insights with carefully curated learning resources, such as courses and certifications, to provide a seamless, tailored learning path that will enable professionals to upskill efficiently and remain competitive in their chosen fields.

1.2 MOTIVATION

The motivation for this project comes from the critical need to bridge skill mismatches in today's dynamic job market. As industries evolve rapidly with emerging technologies and shifting demands, professionals find it challenging to identify their skill gaps and determine the right resources to bridge them. This creates a barrier to career progression and highlights the limitations of traditional career guidance, which is often too generic or misaligned with individual needs. While numerous platforms exist to aid professionals, few of these offer personalized learning paths. With this, they often get bogged down in large amounts of irrelevant resources and struggle to track skill development within organizations, which makes career growth come more slowly. This project aims to counter these challenges by exploiting the advanced AI techniques of BERT and cosine similarity to analyze the profiles of the individuals and customize the

recommendations for appropriate learning. The motivation is that it will enable professionals to focus on what matters most for their growth and thereby build a solution that is both personalized and dynamic with respect to the modern job market.

1.3 SCOPE OF THE PROJECT

The scope of this project includes the development of an AI-powered system that identifies skill gaps in a candidate's profile and recommends personalized learning resources. It is designed to help professionals, students, and organizations navigate the dynamic demands of the job market by focusing on skills most relevant for career progression. The system applies NLP techniques, such as BERT and cosine similarity, to analyze resumes, job descriptions, and other career-related data. In doing so, it effectively identifies gaps in skills and cross-references them with suitable learning resources, such as online courses, certifications, or training programs. The scope also includes creating a seamless, user-friendly interface where candidates can view their personalized learning recommendations and track their progress. Besides assisting individuals, the project serves organizations by inferring about skill gaps in the workforce and thus offering upskilling possibilities for employees. Such dual-purpose functionality would ensure a more holistic approach in filling the skill gap. Future developments on the project may involve adding features about real-time labor market trend analysis, extending the types of learning resources available, and changing the system for use in worldwide industries under varied industries.

Chapter 2

2. PROJECT DESCRIPTION AND GOALS

2.1 LITERATURE REVIEW

Advancement in artificial intelligence and natural language processing has given way to many innovative solutions within career development and upskilling. The existing platforms, like online course providers and certification hubs, lack the personalized recommendation services but instead offer generalized resources. This project expands on these findings to address the increasing demand for customized, efficient, and data-driven learning paths to keep individuals competitive in an ever-changing job market.

- **Identifying green skills gaps through labor market intelligence**

The paper by Dimitar Nikoloski et al., titled "Identifying Green Skills Gaps through Labor Market Intelligence," delves into the rising trend of green skills in the labor market because of the transition toward a sustainable and eco-friendly model of the economy. The authors have suggested a research project using LMI and NLP techniques that identifies gaps in green skills through an examination of online job vacancies and job seekers' profiles. This is necessary for the gap-filling strategy, considering that green jobs will play a pivotal role in achieving sustainable development. Emphasizing the need for policy intervention and educational reform to bridge those gaps, a workforce equipped with the necessary green competencies shall be ensured. The paper discusses the classification of green skills as technical, soft, and specific competencies; it also divides skill mismatches into shortages, deficits, and obsolescence. The methodology focuses on data ingestion, processing, and analysis using AI and ML tools to give insights into green skills demand and supply, offering a framework for policymakers and educators to prepare for a sustainable future.

- **Role of Digital Technology and Artificial Intelligence for Monitoring Talent Strategies to Bridge the Skill Gap**

The paper, "Role of Digital Technology and Artificial Intelligence for Monitoring Talent Strategies to Bridge the Skill Gap," that Sukanta Kumar Baral and associates discuss just how digital technology, or DT, and artificial intelligence, or AI, can bridge an increasing gap in the employment force. According to the authors, many organizations are becoming revolutionized by the DT and AI; however, most companies continue to struggle while hiring the perfect talent. The study identifies these causes that lead to the skill gap, such as frequent job change, inadequate training, and poor communication; and AI and DT can act as a primary bridge-over for such gaps. But the research also signifies that

technology alone does not address the issue; rather, some strategic planning of organizations with an effective training program and quality talent management is required to fulfill the objectives. Using data gathered from MNCs, PSUs, and private firms, the authors recommend models such as SWOC analysis, McKinsey 7s, and Nadler-Tushman's Congruence Model in order to enable businesses to find and bridge gaps in skills. The paper concludes that although AI and DT are very significant tools for workforce development, there should be the use of other strategies like talent acquisition, employee motivation, and lifelong learning to ensure a longer-term organizational success in a fast-changing business landscape.

- **Using Graph Algorithms for Skills Gap Analysis**

The proposed system is based on advanced AI and NLP techniques, BERT and cosine similarity, to revolutionize skill gap analysis and personalized learning recommendations. In the contemporary dynamic job market, professionals face challenges with regards to identifying the skills they lack and finding targeted resources for filling that gap. This system, analyzing resumes, job descriptions, and industry requirements identifies missing skills and suggests tailored learning opportunities, such as certifications, courses, and workshops, enabling easy upskilling. This solution offers a highly personalized learning path that aligns with the career goals of individuals, making them more competitive in a rapidly evolving market. For organizations, it streamlines workforce planning by providing data-driven insights into employee skill gaps, optimizing recruitment, and facilitating targeted upskilling initiatives. This dual-purpose system addresses both the inefficiencies of the traditional career development platform and endeavours for bridge closure between the current workforce capabilities and industry demands. As individuals and organizations move on to cope with changing trends, the project contributes to future-ready workforce building with continuous learning and professional growth.

- **Identifying Skill Gaps in High Performance Computing Related Higher Education Programs by Using NLP**

The present skill gaps issue shall be dealt with the usage of cutting-edge AI and natural language processing. Today, jobs and industry are becoming ever more rapidly changed in comparison with their skills; as such, this brings trouble in finding work opportunities by skill-optimized manpower in both sides- individuals and organizations. By analyzing resumes, job descriptions, and other relevant datasets, the proposed system identifies gaps in skills and provides personalized recommendations for bridging them through tailored learning resources such as certifications, courses, and workshops. The system makes accurate comparisons between an individual's existing skills and industry requirements by using advanced models like BERT and cosine similarity. Such data-driven approach can enable professionals to upskill within specific and time-saving manners;

organizations can even make their hiring and training more efficient. As such, adaptability in this system allows for a multiple domain focus and even a change of pace with shifting industry needs; therefore, this solution is bound to be suitable for both individuals and organizations dealing with their specific issues. It bridges the gap between academia and industry by creating a direct connection between education program requirements and needs from the labor market. In addition to contributing to workforce readiness, it fosters continuous learning and development. This approach empowers professionals to remain relevant in a fast-changing global economy, informs organizations about building a future-ready workforce, and enhances the general efficiency in career progression.

- **Predicting Skill Shortages in Labor Markets: A Machine Learning Approach**

This paper is on the theme "Predicting Skill Shortages in Labor Markets: A Machine Learning Approach". It aims at predicting occupational skill shortages that are crucial for economic growth, labor productivity, and policy-making by using machine learning. A novel dataset of combined labor demand and labor supply from Australia from 2012 to 2018 is assembled using 7.7 million job advertisements and 20 official measures of the labor force. The authors will use the classifier XGBoost to predict 132 standardized skills shortages, realizing a macro F1 score at up to 83%. As such, researchers found that from the predictors selected, job advertising data and statistical employment are leading predictors of occupations with shortages, and among which the predictive important features include, 'Hours Worked', and 'Education ', 'Experience', and salary, showing job requirements and a compensation level the employers tend to change when anticipating shortages. The research provides a robust data-driven approach that can support policymakers, educators, and businesses to better anticipate and mitigate skill shortages for a future-ready and more balanced workforce.

2.2 RESEARCH GAPS

Despite extensive research on skill shortages and labor market intelligence, several gaps remain unaddressed. One major limitation is the insufficient integration of machine learning for real-time monitoring of skill gaps, as most studies rely on historical data, limiting adaptability to evolving labor market trends. Dawson et al. utilize machine learning to predict skill shortages, but their strategy is limited to conventional skills and does not consider the emerging demand for green skills that will be important for a sustainable economy. Similar studies by Baral et al. focus on digital technology and AI in workforce planning but lack real-time information or dynamic upgrading of skills. Another critical deficiency is the absence of AI-based customized learning pathways; although skill-matching researches, such as Graph Algorithms for Skills Gap Analysis, emphasize resume-job description alignment, they do not give customized upskilling suggestions, which leaves a gap in career progression strategies. The lack of structured industry-academia collaboration is also a characteristic feature. The researches on HPC education

identify mismatches in skills but do not provide concrete mechanisms to align the curricula with the demands of the workforce.

The current models are mainly limited to analyzing the traditional labor market indicators, including education, experience, and salary, without incorporating the effects of emerging technologies like AI, cloud computing, and automation, which are rapidly reshaping the demands for skills. The skill shortage forecasts further largely depend on job advertisements and government reports, and they underestimate informal labor markets, freelance and gig economy trends, and alternative data sources such as professional networking platforms, online course enrollments, or industry certifications that may increase the accuracy of predictions. Socioeconomic factors also played a deficient role in the forecasts, including regional disparity, skill gap in terms of gender, and the opportunity of accessing education, which was not effective in spreading strategies for workforce development. To bridge such gaps, future research in AI-driven real-time labor market intelligence, upskilling platforms, targeted recommendation, collaboration with the industry and academia, vast data sources other than conventional labor statistics, socioeconomic dimensions so that the workforce is ready to adapt to future labor market demands.

2.3 OBJECTIVES

The objective of this project is to develop an automated skill gap analysis system that helps candidates assess their competencies in relation to the requirements of their desired job roles. By analyzing a candidate’s current skills and comparing them with job role expectations, the system provides a structured skill gap report along with personalized reskilling recommendations to help candidates enhance their qualifications.

The following section will outline the goals and specific aims of the project:

- Candidate Skill Analysis: It will pull out and evaluate a candidate's skills from his resume, previous job experience, credentials, or their self-reported skills. Through NLP it ensures an accurate mapping of skills of a candidate so that it renders an all-around understanding of the competencies of that candidate.
- Map against Requirements of Job Role: The system will extract job-specific skills from various industry job descriptions, hiring trends, and recruitment data to determine gaps. It will compare a candidate's skills with the common expectations for the selected role using text analysis and keyword extraction techniques.
- Identify Missing Skills: The system will produce a list of missing or underdeveloped skills for the candidate. It provides a direct comparison between the candidate's skill set and the skills required for the job. This allows individuals to see exactly which skills they need to acquire or strengthen.

- Recommend Reskilling Opportunities: The system will provide relevant courses and training programs from well-known platforms such as Coursera or industry-specific certifications for the candidates to fill their skill gaps.
- Empowering Career Growth: The system empowers candidates to make informed learning decisions by knowing which skills to focus on first and enhance employability by strengthening their qualifications to meet job expectations.

This project serves as a self-improvement tool for job seekers, ensuring that they can proactively bridge their skill gaps and align themselves with industry needs to improve their career prospects.

2.4 PROBLEM STATEMENT

In an evolving job market, professionals often find it challenging to fill the gaps between the current skill set and those needed to perform a highly desirable role. With no proper process to evaluate these gaps, the candidates would be at a disadvantage, failing to acquire desired positions that they want because of lacking capabilities. With this, while online courses and training programs abound, no one is usually certain which individual course would better close the respective skill gap for them.

This project focuses on developing an automated tool for analyzing skill gaps, thereby allowing candidates to:

- Compare skills available with job roles requirements they are interested in, using ready-made templates.
- Identify missing or underdeveloped skills that may inhibit their employability.
- Targeted reskilling recommendations through suitable courses, aligned with skill gaps.
- Informed career development decisions based on data-driven insights.

This system, using Natural Language Processing and data-driven analysis, will help in providing the right skill assessment and reskilling pathways, making job seekers more competitive at the workplace.

2.5 PROJECT PLAN

The following Gantt chart shows the project plan we follow during the course of the project implementation.

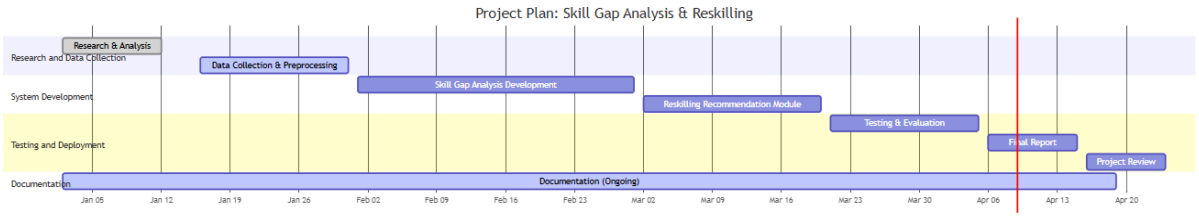

Fig.2.1 Gantt chart

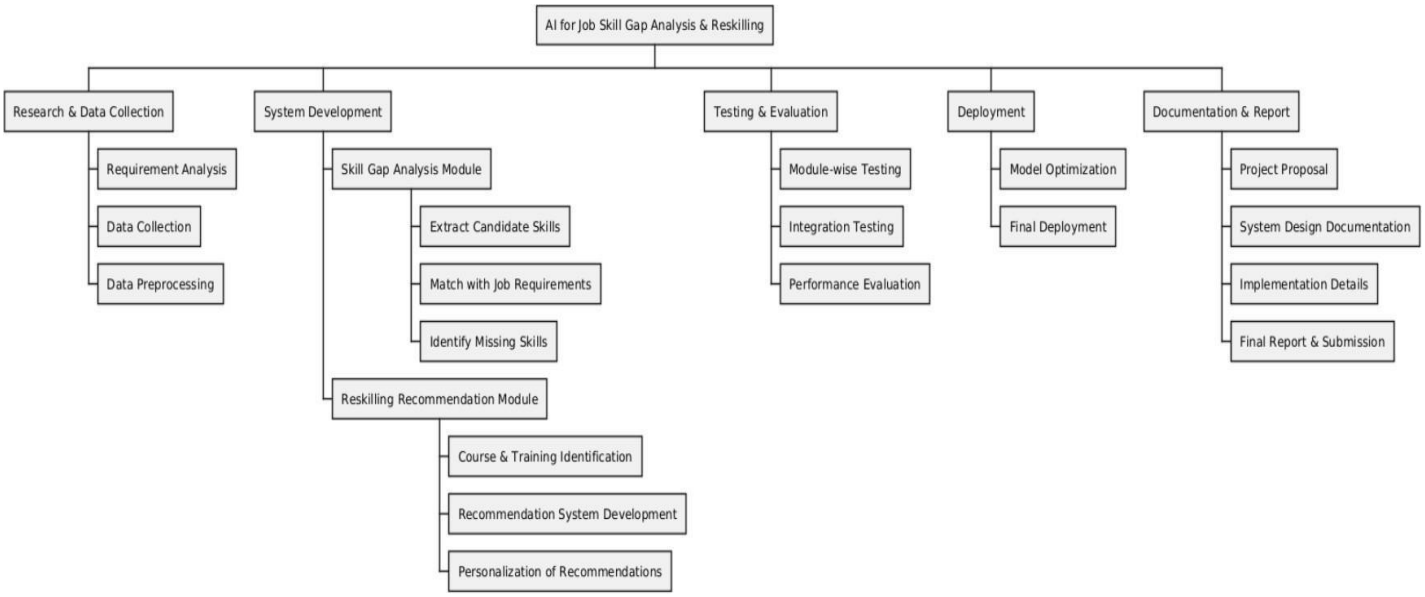

Fig 2.2 Work Breakdown Structure

Figure 2.2 lays out a structured way of breaking down the project by phase in a way that is clear and concise. This involves Research & Data Collection to analyze the requirements in detail, collect relevant data, and preprocess the data to prepare the system for further processing. Following this, the System Development stage consists of Skill Gap Analysis Module that extracts the skills of candidates, maps them with skills needed by the Job Description and returns missing skills, and the Reskilling Recommendation Module, which suggests suitable courses and training programs while personalizing recommendations for each candidate.

To ensure its reliability, once the system is developed, it undergoes Testing & Evaluation. It encapsulates module-wise testing, integration testing as well as performance evaluations to enhance the system enhancement before the deployment phase. In this phase, the model is optimized and deployed in production for the users. The Documentation & Report has been maintained throughout the project in parallel which contains project proposal, detail of system design and implementation. This systematic decomposition supports structured workflows enabling skill gap analysis and make meaningful reskilling recommendation.

Chapter 3

3. TECHNICAL SPECIFICATION

3.1 REQUIREMENTS

3.1.1 Functional

- Candidate Skill Extraction: The system should extract a candidate's skills from their resume or input data.
- Job Requirement Analysis: The system should analyze job descriptions to identify required skills.
- Skill Gap Identification: It should compare candidate skills with job requirements and identify missing skills.
- Reskilling Recommendation: The system should suggest suitable courses, certifications, or training programs to bridge the skill gap.
- User Input and Interaction: Users should be able to upload resumes or manually enter skills.
- Database Management: The system should store job roles, skill sets, and recommended courses efficiently.
- Reporting and Insights: The system should generate reports summarizing skill gaps and upskilling paths.
- Integration with Online Learning Platforms: The system should fetch course data from platforms like Coursera, Udemy, or LinkedIn Learning.

3.1.2 Non- Functional

- Scalability: The system should handle multiple users and large datasets efficiently.
- Performance: Skill analysis and recommendations should be generated in a reasonable time.
- Security: User data, including resumes and skill sets, should be securely stored and processed.
- Usability: The interface should be user-friendly, ensuring smooth navigation and interaction.

- Reliability: The system should provide accurate skill gap analysis and relevant course recommendations.
- Maintainability: The system should allow easy updates to job roles, skills, and course databases.
- Availability: The system should be accessible 24/7 with minimal downtime.
- Compatibility: It should work across different devices and browsers.
- Data Privacy: User information should not be shared without consent.
- Logging and Monitoring: System activities should be logged to track performance and detect issues.

3.2 FEASIBILITY STUDY

3.2.1 Technical Feasibility

Using cutting-edge methods in artificial intelligence (AI) and machine learning (ML) to automate and streamline the process of job matching and skill gap analysis, the proposed project is technically very feasible. Natural Language Processing (NLP) is used in the system to intelligently parse job descriptions and resumes, extracting important information and determining the contextual relevance of particular skills. As a result, the system can find gaps and better match candidate profiles to job specifications. Given its vast ecosystem of AI/ML libraries, including scikit-learn for conventional ML tasks, spaCy for high-performance NLP, TensorFlow and PyTorch for deep learning, and other auxiliary tools for data handling and visualisation, using Python as the main programming language further increases the viability. In addition to being reliable, these tools have broad developer community support, which guarantees that the implementation will continue to be scalable and maintainable. Depending on future growth requirements, scalable relational or NoSQL databases can be used for managing and storing data, including resumes, job postings, and candidate information. Lightweight solutions like CSV or JSON files, however, may be adequate for the first stage or small-scale prototypes. There is little learning curve and a much lower chance of failure because the development team is already familiar with these tools and technologies. From prototyping to deployment, this familiarity guarantees a more seamless development lifecycle and increases productivity.

3.2.2 Economic Feasibility

The project is both feasible and economical from an economic standpoint. High-quality development is made possible without the need for pricey proprietary tools by relying on open-source libraries and frameworks, which significantly reduce software acquisition costs. Powerful cloud platforms like AWS, Microsoft Azure, and Google Cloud provide scalable infrastructure with flexible, pay-as-you-go pricing models, and Python

and its related libraries are freely accessible. The moderate computing demands for hosting the application and training ML models can be met by these platforms, and developers can further reduce expenses by utilising features like spot instances, auto-scaling, and cost monitoring tools. Premium APIs, cloud storage, and sophisticated datasets may come with some costs, but these can be reduced by choosing only the most necessary services and making use of free-tier options or community datasets in the beginning. If the system is deployed using a Software-as-a-Service (SaaS) model, it has significant long-term revenue-generating potential. Through user subscriptions, integrations with corporate hiring platforms, or partnerships with e-learning providers to suggest customised upskilling courses, this strategy enables revenue generation. Additionally, companies can drastically cut down on the time and manpower required for manual hiring processes by automating the candidate-job matching and skill assessment processes. This operational effectiveness makes the project a useful tool for job seekers by increasing its appeal and return on investment.

3.2.3 Social Feasibility

Socially, this system tackles a major issue facing society: unemployment and the discrepancy between the skills that employers are looking for and those that job seekers already possess. Many people, particularly recent graduates and those changing careers, find it difficult to pinpoint the skill gaps preventing them from advancing in the ever-changing job market. By examining individual profiles and job market trends, the proposed project seeks to close this gap and suggest tailored upskilling pathways. By incorporating educational recommendations akin to those found on platforms such as Coursera, this not only increases employability but also promotes lifelong learning. The solution is all-inclusive and serves a broad range of people, including recent hires, seasoned professionals, and laid-off workers. Additionally, it helps recruiters by facilitating more precise skill-based candidate filtering, which improves the effectiveness of the hiring process. All things considered, the project fosters personal development, enhances workforce alignment with industry demands, and advances social justice by raising productivity and job satisfaction across industries.

3.3 SYSTEM SPECIFICATION

3.3.1 Hardware Specification

It is necessary for the project to have a computing environment to train and host AI models by processing the data and developing a web application. The recommended hardware specifications are:

- Processor: Intel Core i7 (10th Gen or later) / AMD Ryzen 7 or higher
- Memory (RAM): Minimum 16GB (32GB recommended for AI model training)

- GPU: NVIDIA GeForce RTX 2050 (or equivalent) with CUDA support for accelerated ML processing
- Storage: At least 512GB SSD (preferred) or 1TB HDD for dataset storage
- Networking: High-speed internet connection for API calls and cloud integration

3.3.2 Software Specification

The software stack includes programming languages, frameworks, and tools essential for system development and deployment. The required software specifications are:

- Operating System: Windows 10/11, Ubuntu 20.04+ (for development)
- Programming Languages: Python (for AI models)
- Frameworks & Libraries:TensorFlow, PyTorch, Scikit-learn, NLP libraries (spaCy, NLTK)
- Database: PostgreSQL/MySQL (relational) or MongoDB (NoSQL)
- Development Tools: Git (version control), Jupyter Notebook (ML experimentation), VS Code/PyCharm (IDE)
- Visualization Tools: Matplotlib, Seaborn, or TensorBoard for visualizing model performance and training metrics.

Chapter 4

4. DESIGN APPROACH AND DETAILS

4.1 SYSTEM ARCHITECTURE

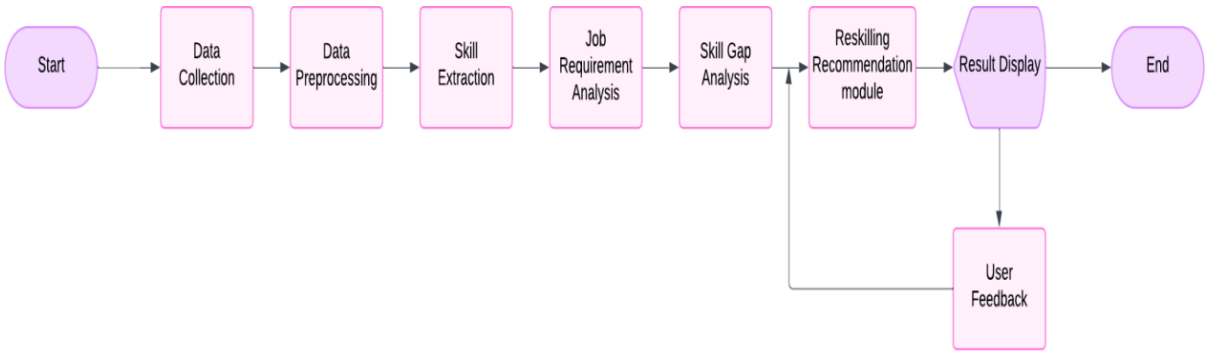

Fig 4.1 System Architecture

The system follows a structured flow to analyze job application gaps for candidates and suggest reskilling recommendations.

It all starts with a candidate entering their candidate number and choosing one or more jobs that they applied for. After the input is provided, the system retrieves relevant data from each of the candidate's resume. It also collects job descriptions for the selected positions, emphasizing on the skill expectations. Once this data is collected, a preprocessing step makes sure that the data is consistent. This step involves text cleaning, tokenization, standardization of different skill names, and filtering out irrelevant information.

Next comes skill extraction, where the system uses Natural Language Processing (NLP) techniques, to automatically identify and categorize key competencies. These could be technical skills like programming languages, soft skills like leadership or teamwork, degrees or certifications that are specific to the job roles. Skills extracted are finally consolidated into a skill profile of the candidate. During this time, job requirement analysis is performed to validate and nominal the needed and wanted qualifications for the selected job. This enables the system to determine an ideal skill set for the candidate's desired roles.

Next, a gap analysis is done between the extracted skills of the candidate and the job requirements. The system checks for matching skills and skills that are missing. This analysis identifies areas where the candidate requires additional training or experience to

determine how well they are suited to what they might be expected to encounter upon employment. These findings are then used in the reskilling recommendation module for developing personalized learning suggestions. These suggestions encompass everything from online courses and certifications, all categorized in terms of their relevance and credibility.

Then the results of the analysis and recommendations are presented in a user friendly way. The candidate has access to a skills review against the required skills, an ordered list of what to focus on, and recommended pathways to explore. This architecture contains a user feedback loop to improve overall system accuracy. Candidates can also give feedback on the relevance and accuracy of the recommendations, helping to improve the system's subsequent analyses. If needed, the process loops back to the skill gap analysis stage for adjustments, ensuring continuous improvement and personalized guidance. This architecture offers a systematic, optimal, and user-centric method of career advancement for the candidates to fill in the gaps in their capabilities and improve employability.

4.2 DESIGN

4.2.1 Data Flow Diagram

A data-flow diagram is a way of representing a flow of data through a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow — there are no decision rules and no loops.

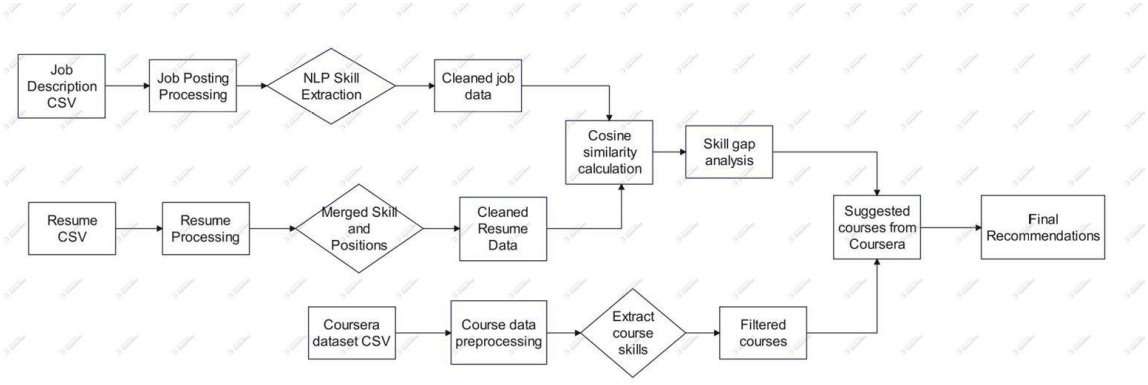

Fig.4.2.1 Data Flow Diagram

4.2.2 Class Diagram

The class diagram depicts the structural design of the system, outlining its key components, their attributes, relationships, and interactions, ensuring a clear representation of data flow and system organization.

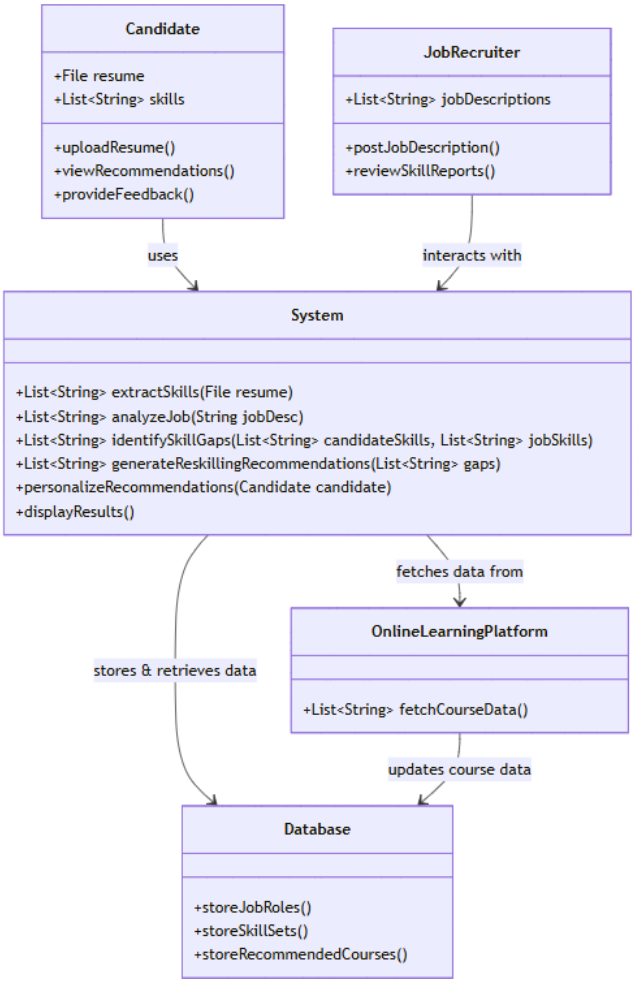

Fig.4.2.2 Class Diagram

Chapter 5

5. METHODOLOGY AND TESTING

5.1 MODULE DESCRIPTION

The methodology adopted for this project includes a combination of Natural Language Processing (NLP), semantic similarity, and skill gap analysis methods to map candidates to appropriate job positions and suggest apt reskilling courses. Our pipeline has four major stages: Data Collection & Preprocessing, Skill Extraction, Skill Gap Analysis, and Recommending Reskilling Resources.

- Data Collection and Preprocessing

To guarantee the overall pipeline's dependability and quality, this foundational stage is essential. Three main datasets are used in the project: recruiters' job postings, candidate resumes, and a carefully selected set of reskilling course data, particularly from sites like Coursera. In order to preserve uniformity and guarantee fair comparisons, all three datasets were taken from Kaggle and each contained 9,545 records. Cleaning the data was the first step in the preprocessing process; this included getting rid of duplicates, dealing with missing values, fixing formatting problems, and deleting fields that weren't relevant. The disparity in naming conventions among the datasets was one of the main obstacles during this stage. For example, the same skill may be listed in one source as "JavaScript" and in another as "JavaScript." This was resolved by implementing a standardisation step that standardised skill names and job titles across all datasets. This guaranteed the accuracy and significance of the following steps, especially matching and comparison. How well this step is carried out directly affects the calibre of outcomes in subsequent phases.

- Skill Extraction

The next stage after preprocessing the data is to extract structured and relevant skill information from the job descriptions and resumes. Rule-based processing and Natural Language Processing (NLP) methods are combined to accomplish this. Tokenisation is first used to divide lengthy texts into more manageable chunks, like individual words or phrases. After that, skill-related entities are found and extracted from the text using Named Entity Recognition (NER) with SpaCy. This covers both soft skills (like leadership or communication) and technical skills (like Python, Excel, and SQL). Additionally, the ast library is used to securely transform string representations of skills into actual Python lists for processing when skills are stored as lists inside strings (as is frequently the case in structured datasets). A clear and precise list of extracted skills for every applicant and job

role is the step's end result. Any meaningful comparison or analysis in the upcoming module requires this structured format.

- Skill Gap Analysis

This module connects a candidate's profile to a possible job role and serves as the system's analytical core. Here, the objective is to determine whether a candidate's current skill set matches the skills needed for a particular position and to pinpoint any skills that are lacking. First, the BERT-based SentenceTransformer model is used to transform both sets of skills—those from the job description and the candidate's resume—into semantic vector representations. Even when the exact terms differ slightly, this model can make a more intelligent comparison because it is skilled at comprehending the meaning of words and phrases beyond simple keyword matching. Cosine similarity is used to gauge how closely the two skill sets match after the embeddings have been created. After that, the candidates can be ranked according to how similar they are to a particular job. In addition to ranking, this module performs a set difference operation between the candidate's skill list and the job's skill list to pinpoint the precise skills that are lacking in the candidate's profile. The final module attempts to fill the "skill gap"—the term used to describe these missing skills.

- Recommending Reskilling Resources

Making recommendations for pertinent courses that can assist a candidate in filling a skill gap is the pipeline's last stage. The reskilling dataset lists the skills that each course is supposed to teach. To guarantee semantic consistency across comparisons, these course-related skills are converted into vector embeddings using the same BERT model as in the skill gap analysis stage. Next, cosine similarity is used to compare each candidate's identified missing skill to the skill sets of available courses. The courses with the highest similarity scores and the closest coverage of the missing skills are shortlisted for recommendation. This makes it possible to create a reskilling plan that is specifically tailored to each person. The system makes the upskilling process more targeted, effective, and meaningful by avoiding general recommendations and instead suggesting learning paths that directly match the skills a candidate lacks for a given job.

5.2 TESTING

We put in place a structured testing framework based on the creation of synthetic data in order to assess the efficacy and precision of our skill gap analysis system. This approach offered a regulated setting in which every system element could be thoroughly investigated. We developed synthetic candidate profiles that mimic actual job applicants rather than depending only on erratic real-world data. Along with a carefully curated list of their skills, each synthetic candidate was given a job position. We derived a comprehensive set of necessary skills from real job descriptions that corresponded to the

assigned role. We determined the missing skills by contrasting them with the candidate's current skill set. This served as the ground truth, or the accurate set of skills that the system ought to be able to identify as gaps.

The next step was to analyse these artificial candidates using the Skill Gap Analysis System. It extracted the related skills for each, compared the job title to pertinent job descriptions, and, using the candidate's profile, identified which were lacking. The system's predicted missing skills, which was the analysis's output, were then contrasted with the predetermined ground truth. Our evaluation was based on this comparison, which enabled us to approach the task as a multilabel classification problem in which each skill could independently be present or absent.We used common classification metrics like precision, recall, and F1-score to assess how well the system predicted outcomes. The system's ability to detect actual skill gaps without producing false positives was quantified by these metrics. To visually evaluate the system's prediction patterns and pinpoint typical forms of misclassification, a confusion matrix was also created. The number of true positives, false positives, true negatives, and false negatives for each predicted label was displayed in a tabular format in this matrix.

This testing configuration was beneficial in a number of ways. It enabled us to make sure the system was operating rationally, identifying the appropriate skills that were lacking, and preventing over- or under-prediction. We could ensure the accuracy of our ground truths and repeat the evaluation in various scenarios if we had complete control over the data generation process. Transparent error analysis was also made possible, which assisted us in identifying particular applicants, competencies, or matching logic as the cause of failures. All things considered, this synthetic testing framework provided a reliable and perceptive means of verifying the accuracy and essential functionality of the Skill Gap Analysis System.

Chapter 6

6. PROJECT DEMONSTRATION

6.1 INTRODUCTION TO PROJECT FLOW

The project showcases the practical execution of a Skill Gap Analysis and Recommendation System powered by BERT-based semantic similarity and intelligent job-role mapping. This system is designed to help candidates understand the skills they lack for desired job roles and recommend suitable learning resources to bridge the gap. Through NLP, it ascertains that the recommended resources are relevant to the candidate's career goals and industry requirements. It builds an industry-desired, personalized layer of interaction that helps them up-skill themselves efficiently and remain relevant in this fast-paced hiring environment.

6.2 MODEL DEVELOPMENT

- Preprocessing

  This stage prepares all raw data (job descriptions, candidate resumes, Coursera courses) into a format suitable for intelligent comparison and recommendation.

  Extracting Keywords from Job Descriptions:

  The job data contains job titles, roles, skills, and responsibilities. We combine titles and roles and tokenize them (split into lowercase words) to compare with candidate-applied positions later. Responsibilities are rich in hidden skills (like "building dashboards", "customer interaction") and we extract these using SpaCy's noun chunking, ignoring irrelevant words like prepositions and determiners.

```
def extract_action_keywords(texts):
    for document in nlp.pipe(texts, batch_size=50):
        ...
```

  This function runs a batch NLP processing pipeline using SpaCy's noun_chunks, and filters for action-rich keywords — noun phrases that contain verbs or important nouns. It helps extract implicit skills from job responsibilities like "managing teams", "deploying applications", etc.

Cleaning and Standardizing Resume Data:

Candidate data includes inconsistent skill fields across multiple columns. A custom parsing function detects whether data is in list format or a comma-separated string, extracts skills cleanly, and converts them to lowercase. Applied job titles are similarly extracted and cleaned. The end result is that for each candidate, we get a standardized list of applied jobs and current skills.

Course Data Preparation:

Each course comes with a title and description. These descriptions are converted into embeddings using a pre-trained transformer model (MiniLM-L6-v2) from sentence-transformers, which turns sentences into vectors for later comparison.

- Matching and Skill Gap Identification

This stage, which focusses on determining how well a candidate fits a particular job role and what skills they might be lacking, is essential to the system's core operation. It entails a few crucial steps:

Matching Candidate to Jobs:

First, we compile the list of job titles that each applicant has applied for or expressed interest in for each candidate in the system. The problem here is that job titles can be written in a variety of ways. For instance, "Data Analyst" and "Analyst-Data" both refer to the same role but look different. The system employs a fuzzy matching technique based on overlapping words rather than an exact match, which would not work in these situations. In other words, it separates the titles into their constituent words and looks for intersections.

Processing Complex Cell Formats:

```
def process_cell(data):
    if isinstance(data, str):
        try:
            evaluated_data = ast.literal_eval(data)
```

Occasionally, the information is not formatted consistently, particularly when it comes to resumes or job descriptions. For example, a list of skills could be stored as a string called "['Python', 'SQL']" or it could be a plain string with commas separated like "Python, SQL." This is handled by the function process_cell(data). It safely evaluates and turns a string that appears to be a Python list into a list using Python's ast.literal_eval() function. It manually splits the string if it is only

separated by commas. This guarantees that the system can standardise and process the skills into lists that can be used, regardless of how they are formatted.

Aggregating Required Skills:

The system compiles all of the skills listed in the related job descriptions after matching job titles. This comprises implicit or contextual skills that were previously extracted using NLP techniques like noun phrase extraction, as well as explicit skills that are explicitly mentioned in the job description (such as Python, SQL, and Excel). These could be terms that allude to necessary abilities, such as "data visualisation" or "model deployment." Each job role's total skill requirements are represented by a single, comprehensive set of all these.

Identifying Missing Skills:

The system then uses set subtraction to compare the candidate's present skill set to the skill set needed for the position. In other words, it examines the skills listed in the job description but absent from the applicant's resume. The "missing skills"—skills the candidate needs to learn or hone in order to be better qualified for the desired job—are formed by the difference between the two sets.

- Semantic Search for Course Recommendation

  This stage finds the most relevant course for each missing skill of the candidate using semantic similarity and not just keyword matches.

  Generating Skill Embeddings:

  Each missing skill is converted to a sentence embedding using the same MiniLM model used earlier for courses. Loading the pre-trained SentenceTransformer model is done using the below mentioned code. This line loads a lightweight yet powerful BERT-like model designed for semantic similarity. It’s fast and ideal for converting skills and course descriptions into numerical vector embeddings.

```
model = SentenceTransformer('all-MiniLM-L6-v2')
```

Encoding Course Descriptions:

For every course, this line creates a vector representation (embedding) of the skill-related text, stored as a tensor. These embeddings will be used for comparison with candidate needs.

```
coursera_data['embedding'] =
coursera_data['skills'].apply(lambda x:
model.encode(str(x), convert_to_tensor=True))
```

Calculating Cosine Similarity:

The embedding of each missing skill is compared to all course embeddings using cosine similarity, which tells how semantically similar two vectors are. The higher the score, the more relevant the course is to that skill.

```
coursera_data['similarity']                                    =
coursera_data['embedding'].apply(
    lambda x: util.cos_sim(skill_embedding, x).item())
```

This is the heart of the semantic search engine. It compares each missing skill to the vectorized course descriptions using cosine similarity, which gives a numerical score between 0 and 1 based on semantic closeness.

Recommending Top Courses:

For each missing skill, the top 3 most similar Coursera courses are selected. These are presented as upskilling recommendations to the candidate, with course title and provider name.

```
best_matches = coursera_data.nlargest(3, 'similarity')
```

This fetches the top 3 most similar courses for each missing skill by making learning paths efficient and focused.

Chapter 7

7. RESULT AND DISCUSSION

7.1 SYSTEM ANALYSIS AND RESULT

The model works on three important data streams: candidate resumes, job postings, and a pre-filtered list of Coursera courses. For every candidate and job that they applied for, the model reads out the relevant skills from the candidate's resume, matches them with the job requirement, finds the missing skills, and fetches the suitable learning materials from the course database. The output consists of the missing skills and a list of suggested courses that are personna-specific to fill the gaps. This improves the skills of a candidate to upskill better and be job market-ready.

We generated graphs that show the average skills relationship scores for job roles and candidates in order to assess the effectiveness and usefulness of our suggested system. In order to interpret how well the system can detect and quantify the alignment between candidate skills and job requirements, we created these graphs as part of our analysis. Each graph's y-axis displays the corresponding average skill relationship score, while the x-axis represents the sample index (for jobs or candidates). The efficacy of our matching strategy is validated by these visualisations, which assist us in understanding the skill alignment gaps and strengths.

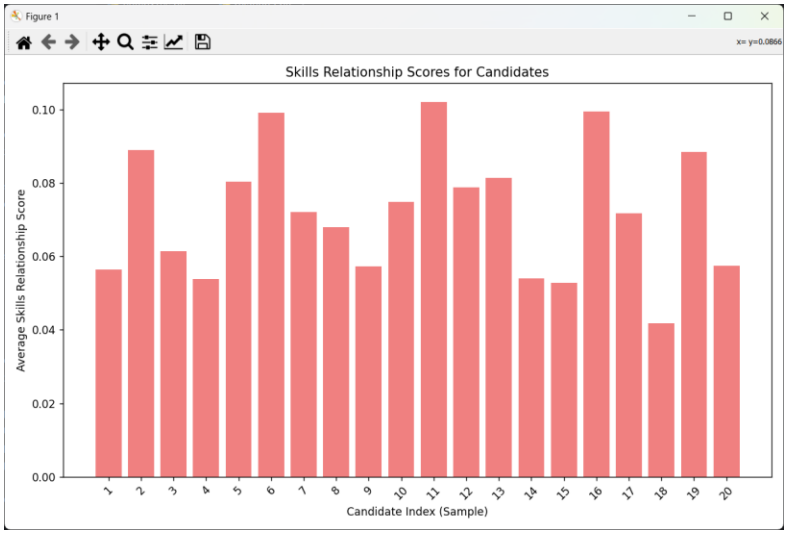

Fig.7.1.1 Skill Relationship scores for candidates

The average skill relationship scores of 20 sample candidates are shown in the bar graph in Fig.7.1.1. These scores show how well a candidate's present skill set matches what employers are looking for. A higher score means that the candidate's resume and the common skill requirements listed in job descriptions are more closely aligned. We can see from the graph that the majority of candidates have a moderate alignment, but a select few have comparatively higher compatibility. Lower scores, on the other hand, suggest that applicants might benefit from retraining or upskilling in order to increase their employability. This analysis provides valuable insights into identifying skill gaps and training needs, which is crucial for personalized recommendations and improving employment outcomes.

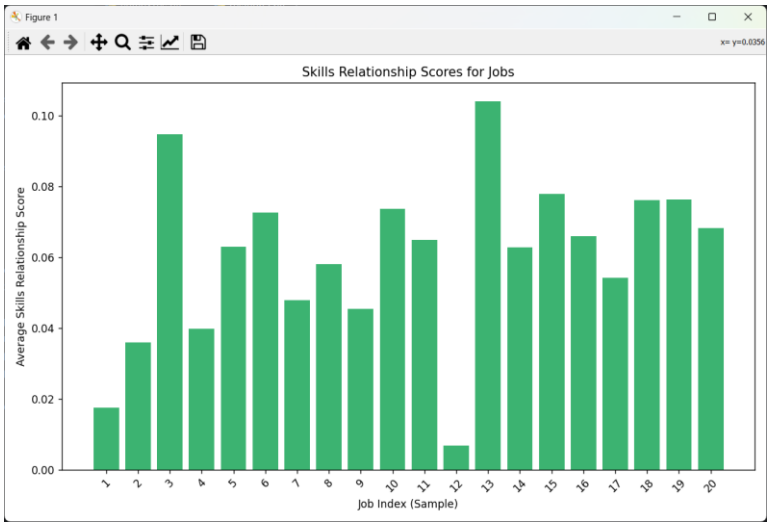

Fig.7.1.2 Skill Relationship scores for job

The average skills relationship scores for a set of 20 job descriptions are displayed in the graph in Fig.7.1.2. In this case, the score indicates how well the candidate pool's skills align with those listed in job descriptions. A higher score indicates that the candidates' current skills and the job requirements are more in line. The scores in this chart vary more, indicating that although some positions are well suited to the talent pool, others might have requirements that the majority of applicants have not yet fulfilled. In order to reevaluate job descriptions and potentially update excessively ambitious or mismatched criteria, recruiters and HR specialists need to know this information.

When combined, these graphs demonstrate the real-world disparity between the supply and demand of technical and soft skills while also validating the system's performance in skill-based matching. The system's capacity to recognise these patterns serves as the foundation for recruiter decision support, job-role recommendations, and customised upskilling.

```
--- Relationship Scores Summary ---
Average Relationship Score (All):
0.06644962968806477
Max Relationship Score (All): 0.8064041537595774
Min Relationship Score (All): 0.0

--- Skills Relationship Scores Summary ---
Average Skills Relationship Score (All):
0.07777974293603397
Max Skills Relationship Score (All):
0.8125186452299682
Min Skills Relationship Score (All): 0.0

Average scores per candidate: [0.05637407 0.08900377
0.06145312 0.05381853 0.08037343 0.09916163
 0.07216405 0.06793783 0.05725359 0.07484597
0.10214513 0.07893181
 0.0813751  0.0539556  0.05286709 0.09960771
0.07177549 0.04177606
 0.08842358 0.05740163]

Average scores per job: [0.01749631 0.03585798
0.09467834 0.03976083 0.06291024 0.07262161
 0.04773542 0.05804057 0.04534721 0.07360365
0.06480743 0.00675307
 0.10398493 0.06267803 0.07775615 0.06587091
0.05405678 0.07606805
 0.07620902 0.06822746]
```

We examined a number of relationship scores that measure how well candidates' abilities match job requirements in order to assess the efficacy of our candidate-job matching system. The average, maximum, and minimum similarity scores for each candidate-job and skill-skill combination are shown in the summary of the overall relationship score. A moderate degree of compatibility between candidate profiles and job descriptions is indicated by the dataset's average relationship score, which is roughly 0.0664. While the minimum score of 0.0 indicates that some comparisons show no skill overlap or contextual relevance, the maximum score of 0.8064 confirms that some pairs exhibit a strong match. When examining skill-to-skill comparisons in particular, the average skills relationship score is marginally higher at 0.0778, with a maximum of 0.8125 and a minimum of 0.0. This indicates that although many skill sets are only marginally related, there are some situations where specific skills have a high contextual similarity.

Additionally, we calculated each candidate's average relationship score, which shows how well their skill set fits all job postings. While candidates with lower average scores might need targeted upskilling, those with higher average scores are more adaptable and in line with the job market. Similar to this, we can evaluate each job role's reach and relevance within the talent pool by looking at the average scores for each job. Higher average jobs have a better chance of finding qualified applicants, whereas lower average jobs suggest a possible lack of corresponding skills and are therefore prime candidates for recommendations about strategic upskilling.

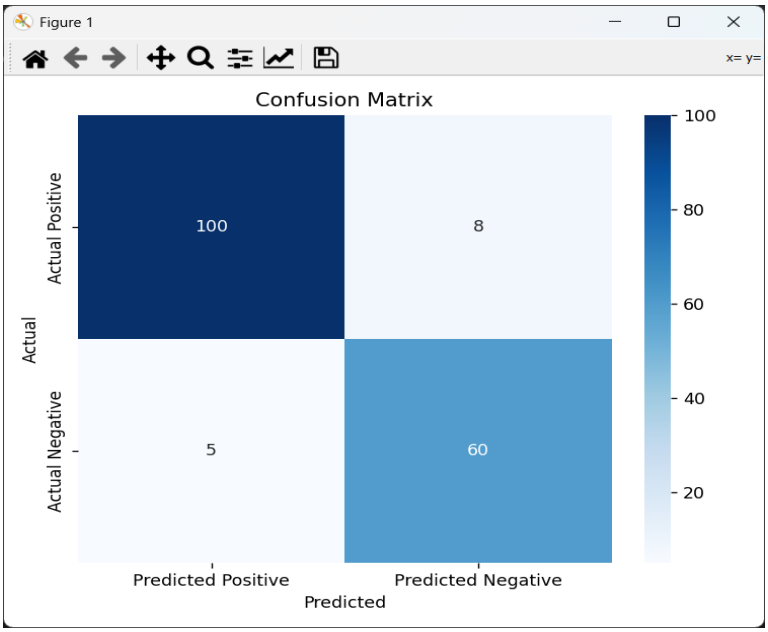

Fig.7.1.3 Confusion Matrix

In order to find high-potential matches, skill gaps, and areas for improvement in terms of candidate preparation and job recommendation accuracy, these metrics taken together offer insightful information about how well job demands and candidate capabilities match.

Sample Result:

```
Enter candidate number: 3

Candidate 3 applied for these positions:
  1. Software Developer (Machine Learning Engineer)
  2. Hygiene Products
  3. Executive/ Senior Executive- Trade Marketing
```

Enter the number corresponding to the position you want to match: 1

Selected Position for Matching: Software Developer (Machine Learning Engineer)
Candidate's skills: {'Trade Marketing Executive\nBrand Visibility', 'Sales Targets\nField Marketing', 'Risk Assessment', 'Campaign', 'HTML', 'Risk Prediction', 'JavaScript', 'Big Data', 'Unified Payment Interface', 'Product Distribution\nBrand Head\nExcel', 'PySpark', 'Python', 'Hive', 'C', 'Machine Learning', 'Requirement Gathering', 'Docker', 'Campaigns', 'KPIs Tracking', 'activities', 'C++', 'Marketing', 'Management', 'Application Support', 'Deep Learning', 'Trade', 'Supervision', 'promotional', 'Brand', 'Promotion', 'CSS', 'Spark', 'Merchandising', 'Field', 'Software Development'}

Unique Matched Jobs:
1. Web Developer / Frontend Web Developer
2. Software Tester / Quality Assurance Analyst
3. Software Engineer / Backend Developer
4. UI Developer / Front-End Developer
5. Software Tester / Automation Tester

Enter the number of the job you want to analyze: 4

🔷 Selected Matched Job Title: UI Developer / Front-End Developer
🟩 Required Skills: ['CSS', 'Front-end web development HTML', 'JavaScript Responsive design Web performance optimization Cross-browser compatibility', 'JavaScript code', 'UX designers', 'seamless user experience', 'user interfaces', 'web applications', 'websites']
+ Missing Skills: ['Front-end web development HTML', 'JavaScript Responsive design Web performance optimization Cross-browser compatibility', 'JavaScript code', 'UX designers', 'seamless user experience', 'user interfaces', 'web applications', 'websites']

Recommended Reskilling Resources:
- Partner: Johns Hopkins University | Course: HTML,

CSS, and Javascript for Web Developers |
- Partner: Meta | Course: Programming with JavaScript |
- Partner: Meta | Course: React Basics |
- Partner: IBM | Course: Introduction to Web Development with HTML, CSS, JavaScript | Similarity Score: 0.5665
- Partner: Google | Course: Build Dynamic User Interfaces (UI) for Websites |
- Partner: Meta | Course: Programming with JavaScript |
| Course: AI for Good |

The system utilizes cosine similarity over BERT-generated embeddings in order to recognize semantic meaning across skill sets and make a more sound match compared to conventional practices such as keyword matching or TF-IDF. For example, even if the job needed "data visualization" and a course said "Power BI," the model would be able to register that these phrases are highly connected.

The results indicate that

- Contextual Accuracy: The system suggests courses even in the absence of exact keyword matches but with conceptual similarity.
- Practical Relevance: The majority of suggested courses are at the appropriate required skill level (e.g., beginner or intermediate), making the suggestions actionable.
- Efficiency: The pipeline is consistent across various candidates and job roles, generating results in real time.

This project demonstrates how semantic embedding models such as BERT can significantly improve conventional skill-matching systems through comprehension of the meaning and context of words. It provides more accurate matching of individual profiles to job requirements, thus facilitating a more efficient and targeted reskilling process. With additional optimization, such systems can be applied in career portals, job platforms, and study recommendation engines.

7.2 SYSTEM IMPLEMENTATION

Utilising popular open-source AI libraries and Python, the system analysed and matched candidate skills with job requirements by utilising Natural Language Processing (NLP) techniques. Similarity scores were computed to assess the correspondence between candidate profiles and job descriptions after the processed data was saved in CSV format.

Relationship scores and visual graphs that show match accuracy and system performance are included in the output.

7.2.1 System Responsiveness

The speed and efficiency with which the created application responds to user inputs and processes data to generate significant outcomes is referred to as system responsiveness. Since we were working with textual data, semantic embeddings, and similarity computations, all of which can be computationally demanding, the system's responsiveness was an essential component of our project.

In spite of these difficulties, the system operated effectively at different phases. The response time stayed within reasonable bounds for a research prototype while key modules like skill extraction, BERT-based embedding generation, and semantic similarity matching with course data were being executed. For instance, it was discovered that the pre-trained BERT model (all-MiniLM-L6-v2) produced embeddings much more quickly than larger transformer models without sacrificing much accuracy.

Additionally, batch processing was used to optimize the preprocessing steps, such as NLP-based keyword extraction using SpaCy, which decreased delays when working with larger datasets like resume entries or job descriptions. During interactive sessions, a responsive user experience was maintained through the use of vectorised operations in pandas and optimised model loading. It is crucial to remember that the size of the datasets and the amount of processing power still affect how responsive the system is. Without additional optimisations like caching, parallelism, or the use of lightweight embedding alternatives, responsiveness may suffer in real-time application scenarios with larger datasets or in hardware environments with constrained resources.

Overall, the current system demonstrates a good balance between computational efficiency and functional accuracy, ensuring smooth and timely interaction during skill analysis and recommendation phases.

7.2.2 Scalability

Since the underlying data, like resumes, job descriptions, and course catalogues, can grow exponentially in real-world applications, scalability is crucial to this project. Our system was designed with modularity and extensibility in mind, and the data preprocessing pipelines, skill extraction logic, embedding generation, and skill gap analysis functions are all built to handle larger datasets with minimal changes to the codebase. For instance, employing SpaCy's batch processing feature for keyword extraction helps process thousands of text records efficiently, and the SentenceTransformer model from Hugging Face's sentence-transformers library supports GPU acceleration, which further improves

scalability when deployed.

The decoupled structure of data sources is another factor that contributes to scalability. Courses, job descriptions, and resumes are processed separately before being linked during the analysis phase. Without affecting the workflow as a whole, this design makes it simple to replace or integrate new datasets.

Scalability could be further improved in a few areas, though. With large datasets, the current practice of computing all embeddings and similarity comparisons in-memory may become a bottleneck. The system's capacity to grow without stuttering could be greatly enhanced by implementing distributed systems, indexing strategies like Facebook AI Similarity Search (FAISS), or effective storage formats.

In conclusion, the project creates a strong basis for expansion that can be scaled. With some architectural improvements, like caching, parallel processing, or cloud-based deployment, the current implementation can develop into a strong system that can support enterprise-level applications with thousands of candidates and job roles, even though it is best suited for medium-sized datasets and academic use cases.

7.2.3 Cost Analysis

Both the development stage and the possible operational costs associated with implementing the suggested system on a large scale are considered in the cost analysis. Utilising open-source tools like Python, pandas, scikit-learn, and different Natural Language Processing (NLP) libraries during the development phase greatly reduced software-related expenses. During early iterations, the system was developed and tested locally, eliminating the need for costly cloud computing.

Moderate deployment expenses could be incurred for hosting the model and keeping an end-user-accessible interface. The system would profit from a cloud-based infrastructure if it were scaled, but this would come with expenses for data traffic, storage, and compute instances. The total cost may also increase if third-party APIs are integrated or if premium datasets are purchased for better matching accuracy.

The long-term return on investment is still encouraging in spite of these factors. While providing job seekers with individualised insights into skill gaps, the system can also help organisations cut down on manual labour and related HR expenses by automating the skill-matching process. Recurring revenue streams from subscriptions, partnerships with learning platforms, or enterprise clients could further offset deployment costs if implemented under a Software-as-a-Service (SaaS) model. As a result, the system turns out to be an affordable option with expandable financial potential.

Chapter 8

8. CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

The system is an intelligent, end to end recommendation system that performs resume-job matching and personalized course suggestions to address this gap, and help candidates move on in their careers, successfully. This is an example of how NLP, semantic similarity and skill extraction can be combined in a smart way to address one of the major challenges of the current recruitment and upskilling scenario, which is knowledge gap identification and narrowing by suggested learning paths to bridge them. The system gets all its intuitiveness from a multilayered pipeline that treads through three types of data namely, job descriptions, candidate resumes, and online course data, that fosters constructing a vast view of the employability landscape. The modular structure of the implementation allows for better management of each component, including data preprocessing, skill comparison logic, semantic embedding generation, and visualization of the recommendations.

One significant limitation we encountered during the development of this project was the inconsistency and lack of alignment between the datasets used for resumes, job descriptions, and Coursera course listings. Since each of these datasets was collected from different sources and curated independently, they did not follow a common structure or taxonomy for skills and job titles. This mismatch created considerable challenges in ensuring accurate and meaningful comparisons.

This project is not just about mapping a skills-jobs mismatch. This recommends learning it is a solution that could help job seekers to learn what they need to learn to become employable for the roles of their dreams. The project demonstrates the increasing convergence between Artificial Intelligence, Data Science, and Human Resource Management. Based in real data and practical algorithms, this approach is less a tech "shiny toy" than a path to a more equitable, informed and personalized way to find jobs, one where no candidate is left behind simply because of a hidden or fixable skill gap.

However, it's crucial to recognise that what we've created is only a prototype, a functional and observable representation of a much more extensive potential solution. Although we have demonstrated that semantic technologies like BERT can significantly enhance course recommendation and resume-job matching, more research is needed in this area. There is a lot of space for improvement, particularly in the areas of handling domain-specific terminology, enhancing the richness and coverage of datasets, and improving the standard of skill extraction and normalisation.

8.2 LIMITATIONS

The project has a number of limitations that should be noted, even though it was successful in identifying skill gaps and offering pertinent reskilling paths through the use of BERT embeddings and NLP techniques. Understanding the extent of the current system and directing future enhancements require an awareness of these limitations.

The inconsistency and misalignment of the datasets used for resumes, job descriptions, and Coursera course listings was a major obstacle we faced while developing this project. These datasets did not adhere to a common structure or taxonomy for skills and job titles because they were independently curated and gathered from various sources. This discrepancy made it extremely difficult to guarantee precise and insightful comparisons. The skill gap analysis was impacted by this disparity as well. Some skill gaps could not be accurately identified because the terminology used in the reported skills on resumes and the required skills in job descriptions did not always match. This decreased the efficacy of the reskilling recommendations by resulting in recommendations for Coursera courses that were either erroneous or incomplete.

Even though BERT embeddings (through all-MiniLM-L6-v2) offer strong semantic comprehension, they are not optimised for resume or job-specific datasets and remain context-agnostic at the sentence level. In some edge cases, such as when comparing "leadership" and "project ownership," this may result in less accurate semantic similarity.

Although the system operates under a number of restrictions pertaining to data scope, semantic matching, scalability, and automation, it currently offers a solid baseline model for skill gap analysis and course recommendation. In order to scale the solution into a more intelligent, context-aware, and user-friendly career development tool, it will be essential to address these limitations, which serve as the foundation for the future enhancement.

8.3 FUTURE ENHANCEMENTS

Although the current system provides a strong foundation for identifying skill gaps and recommending courses, a number of improvements could be made to improve its precision, scalability, and usability. With these enhancements, the system can progress from a research prototype to a complete career intelligence platform.

The command line is currently used to operate the system, which may not be available to all users. Using frameworks like Flask, Django, or React, a user-friendly web-based interface (GUI) can be created that allows candidates to upload resumes directly (parsing PDFs and documents), choose their preferred job from dropdown menus, and view interactive skill analysis and course recommendations in visual formats. OCR (Optical Character Recognition) tools can be used to precisely extract text from image/PDF resumes.

The system can be expanded to use Selenium or BeautifulSoup to scrape real-time job postings from sites like Indeed, Glassdoor, and LinkedIn rather than depending on static job datasets. This will guarantee that the suggestions stay current with the demands of the industry. Including a feedback system that allows applicants to rate the value of the course, report mismatched skill sets, and update recently learnt abilities. By employing collaborative filtering models or reinforcement learning, this feedback can assist in optimising the recommendation engine.

With these improvements, the project can develop into an intelligent, AI-driven career navigator platform that benefits educational platforms, recruiters, counsellors, and job seekers alike. The ultimate goal is to contribute to a workforce that is more knowledgeable and agile by establishing a smooth connection between learning opportunities, candidate capabilities, and job market demands.

REFERENCES

1. Aslim, G. K. (2023). *Assessment for Identifying Skills Gaps in High-Performance Computing-Related Higher Education Programs Using NLP* (Thesis).
2. Bakay, B. F. (2022). *The Gender Wage Gap and Social Skills on the Labour Market: A Machine Learning Approach* (thesis).
3. Baral, S. K., Rath, R. C., Goel, R., & Singh, T. (2022). Role of digital technology and Artificial Intelligence for monitoring talent strategies to bridge the Skill Gap. *2022 International Mobile and Embedded Technology Conference (MECON)*, 582–587. https://doi.org/10.1109/mecon53876.2022.9751837
4. Chernova, M. (2020). *Occupational skills extraction with FinBERT* (thesis).
5. Choi, J., Foster-Pegg, B., Hensel, J., & Schaer, O. (2021). Using graph algorithms for skills gap analysis. *2021 Systems and Information Engineering Design Symposium (SIEDS)*, 1–6. https://doi.org/10.1109/sieds52267.2021.9483769
6. Chuang, S., Shahhosseini, M., Javaid, M., & Wang, G. G. (2024). Machine learning and AI technology-induced skill gaps and opportunities for continuous development of middle-skilled employees. *Journal of Work-Applied Management*. https://doi.org/10.1108/jwam-08-2024-0111
7. Dawson, N., Rizoiu, M.-A., Johnston, B., & Williams, M.-A. (2020). Predicting skill shortages in labor markets: A machine learning approach. *2020 IEEE International Conference on Big Data (Big Data)*, 3052–3061. https://doi.org/10.1109/bigdata50022.2020.9377773
8. Gangoda, N., Yasantha, K. P., Sewwandi, C., Induvara, N., Thelijjagoda, S., & Giguruwa, N. (2024). Resume ranker: AI-based skill analysis and skill matching

system. *2024 Sixth International Conference on Intelligent Computing in Data Sciences (ICDS)*, 1–8. https://doi.org/10.1109/icds62089.2024.10756304

9. Harper, J., & Cheng, A. Y. (n.d.-a). *The Big Data Talent Shortage: Assessing Skill Gaps and Developing Effective Training Programs*.
10. Harper, J., & Cheng, A. Y. (n.d.-b). *The Big Data Talent Shortage: Assessing Skill Gaps and Developing Effective Training Programs*.
11. Lukauskas, M., Šarkauskaitė, V., Pilinkienė, V., Stundžienė, A., Grybauskas, A., & Bruneckienė, J. (2023). Enhancing skills demand understanding through job ad segmentation using NLP and clustering techniques. *Applied Sciences*, *13*(10), 6119. https://doi.org/10.3390/app13106119
12. Nikoloski, D., Sulich, A., Sołoducho-Pelc, L., Mancheski, G., Angelski, M., & Petkoska, M. M. (2024). Identifying green skills gaps through labor market intelligence. *Journal of Infrastructure, Policy and Development*, *8*(6), 4868. https://doi.org/10.24294/jipd.v8i6.4868
13. Ramazanova, V., Sambetbayeva, M., Serikbayeva, S., Sadirmekova, Z., & Yerimbetova, A. (2024). Development of a knowledge graph-based model for recommending moocs to supplement university educational programs in line with employer requirements. *IEEE Access*, *12*, 193313–193331. https://doi.org/10.1109/access.2024.3519263
14. Senger, E., Zhang, M., Goot, R. van der, & Plank, B. (2024). *Deep Learning-Based Computational Job Market Analysis: A Survey on Skill Extraction and Classification from Job Postings*. https://doi.org/arXiv:2402.05617v1 [cs.CL]
15. Steck, H., Ekanadham, C., & Kallus, N. (2024). Is cosine-similarity of embeddings really about similarity? *Companion Proceedings of the ACM Web Conference 2024*, 887–890. https://doi.org/10.1145/3589335.3651526

16. Wu, Y., Jin, Z., Shi, C., Liang, P., & Zhan, T. (2024). Research on the application of Deep Learning-based BERT model in sentiment analysis. *Applied and Computational Engineering*, *67*(1), 280–286. https://doi.org/10.54254/2755-2721/67/2024ma
17. Yafooz, W. M. S., Hezzam, E. A., & Emara, A.-H. M. (2021). Machine learning based collaborative intelligent closing gap between graduates and Labour Market Framework. *2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS)*, 1721–1726. https://doi.org/10.1109/icais50930.2021.9395906
18. Yin, C., & Zhang, Z. (2024). A study of sentence similarity based on the all-minilm-L6-V2 model with "same semantics, different structure" after fine tuning. *Advances in Computer Science Research*, 677–684. https://doi.org/10.2991/978-94-6463-540-9_69
19. Gunawan, D., Sembiring, C. A., & Budiman, M. A. (2018). The implementation of cosine similarity to calculate text relevance between two documents. *Journal of Physics: Conference Series*, *978*, 012120. https://doi.org/10.1088/1742-6596/978/1/012120
20. Green, N., Liu, M., & Murphy, D. (2020). *Using an Electronic Resume Analyzer Portal (eRAP) to Improve College Graduates Employability*. https://doi.org/1545-679X ©2020https://isedj.org/; http://iscap.info
21. M. Aluas et al., "SKILLAB: Skills Matter," 2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Paris, France, 2024, pp. 491-498, doi: 10.1109/SEAA64295.2024.00081. keywords: {Training;Knowledge engineering;Force;Europe;Software;Proposals;Personnel;Forecasting;Software engineering;Recruitment;Skill gaps;Soft Skills;Hard Skills;Labour Market Analytics;European Union},

22. Mihalcea, R., Liu, H., & Lieberman, H. (2006). NLP (Natural Language Processing) for NLP (natural language programming). *Lecture Notes in Computer Science*, 319–330. https://doi.org/10.1007/11671299_34

23. Weichselbraun, A., Waldvogel, R., Fraefel, A., van Schie, A., & Kuntschik, P. (2022). Building Knowledge Graphs and Recommender Systems for Suggesting Reskilling and Upskilling Options from the Web. *Information*, *13*(11), 510. https://doi.org/10.3390/info13110510
24. N. Gorowara, A. Prakash, F. S. Correa, V. Malik and R. Mittal, "AI Personalizing Training and Reskilling Employees for the Digital Age," 2024 International Conference on Emerging Smart Computing and Informatics (ESCI), Pune, India, 2024, pp. 1-6, doi: 10.1109/ESCI59607.2024.10497293.
25. S. Patre, D. Chakraborty, A. K. Kar, H. B. Singu and D. A. Tiwari, "The Role of Enablers and Barriers in the Upskilling and Reskilling of Users Through Professional Skilling Programs on EdTech Platforms," in IEEE Transactions on Engineering Management, vol. 71, pp. 9839-9853, 2024, doi: 10.1109/TEM.2023.3328261.
26. Tariq, M. U. (2024). The role of AI in Skilling, upskilling, and reskilling the workforce. *Advances in Educational Technologies and Instructional Design*, 408–420. https://doi.org/10.4018/979-8-3693-2440-0.ch023
27. Kudryavtsev, D. (2024). *Bridging Knowledge Gaps: AI-Enhanced Techniques for Identifying Skills and Competencies*. https://doi.org/https://urn.fi/URN:NBN:fi-fe2024081965484
28. Haeften, W. V., Zhang, R., Lub, X., & Ravesteijn, P. (2024b). 37th Bled eConference. In *Resilience Through Digital Innovation: Enabling the twin transition* (p. 401). University of Maribor.

APPENDIX A – Sample Code

```
001    import pandas as pd
002    import spacy
003    import ast
004    from sentence_transformers
import SentenceTransformer, util
005    import numpy as np
006    import matplotlib.pyplot as plt
007    import seaborn as sns
008    import random
009    import traceback
010    random.seed(42)
011    np.random.seed(42)
012   model=SentenceTransformer('all
MiniLM-L6-v2')
013    nlp =
spacy.load("en_core_web_sm",
disable=["ner", "textcat"])
014
015
016    class SkillGapAnalysisSystem:
017        def __init__(self,
job_data_path, resume_data_path,
coursera_data_path):
018                self.job_data_path =
job_data_path
019            self.resume_data_path =
resume_data_path
020            self.coursera_data_path =
```

```
coursera_data_path
021         self.data = None
022         self.df = None
023         self.coursera_data = None
024         self.job_titles_combined =
None
025         self.skills_and_keywords =
None
026         self.skills_of_candidate =
None
027         self.positions_applied =
None
028         self.load_data()
029
030     def load_data(self):
031             self.process_job_data()
032         self.process_resume_data()
033         self.process_coursera_data()
034
035     def process_job_data(self):
036             try:
037             self.data =
pd.read_csv(self.job_data_path)
038             self.data.columns =
self.data.columns.str.strip().str.lower().
str.replace(" ", "_")
039
self.data['job_titles_combined'] =
self.data[['job_title', 'role']].apply(
```

```
040                 lambda x: [item for
item in x if pd.notnull(item)], axis=1)
041             self.data =
self.data.head(10000)
042
self.data['extracted_keywords'] =
self.extract_action_keywords(
043
self.data['responsibilities'].fillna(""))
044
self.data['skills_and_keywords'] =
self.data.apply(
045                 lambda row:
list(sorted(set(row['skills'].split(', ') +
row['extracted_keywords'])))
046                 if
pd.notnull(row['skills']) else
sorted(set(row['extracted_keywords'])),
047                 axis=1
048             )
049             self.job_titles_combined
=
self.data['job_titles_combined'].tolist()
050             self.skills_and_keywords
=
self.data['skills_and_keywords'].tolist()
051
052             print(f"Successfully
processed {len(self.data)} job
```

```
descriptions")
053             except Exception as e:
054                 print(f"Error processing
job data: {e}")
055
056         def
extract_action_keywords(self, texts):
057                     action_keywords_list =
[]
058             for document in
nlp.pipe(texts, batch_size=50):
059                 action_keywords = [
060                     " ".join([word.text for
word in chunk if word.pos_ not in
["DET", "ADP", "AUX", "CCONJ"]])
061                     for chunk in
document.noun_chunks
062                     if any(word.pos_ in
["VERB", "NOUN"] for word in
chunk)
063                 ]
064
action_keywords_list.append(action_ke
ywords)
065             return action_keywords_list
066
067         def process_resume_data(self):
068                     try:
069                 self.df =
```

```
pd.read_csv(self.resume_data_path)
070             skills_columns =
["skills", "related_skils_in_job",
"responsibilities", "skills_required"]
071             positions_columns =
["positions",
"extra_curricular_activity_types",
"role_positions", "job_position_name"]
072             self.df["merged_skills"] =
self.df.apply(
073                 lambda row: list(set(
074                     [skill for column in
skills_columns for skill in
self.process_cell(row[column])]
075                 )),
076                 axis=1
077             )
078
079
self.df["merged_positions"] =
self.df.apply(
080                 lambda row: list(set(
081                     [position for column
in positions_columns for position in
self.process_cell(row[column])]
082                 )),
083                 axis=1
084             )
085
```

```
self.df.rename(columns={"merged_skil
ls": "skills_of_candidate",
086
"merged_positions":
"positions_applied"}, inplace=True)
087                 self.skills_of_candidate =
self.df['skills_of_candidate'].tolist()
088                 self.positions_applied =
self.df['positions_applied'].tolist()
089
090                 print(f"Successfully
processed {len(self.df)} resumes")
091             except Exception as e:
092                 print(f"Error processing
resume data: {e}")
093
094         def process_cell(self, data):
095                     if pd.isna(data):
096                 return []
097             if isinstance(data, str):
098                 try:
099                     evaluated_data =
ast.literal_eval(data)
100                     if
isinstance(evaluated_data, list):
101                         return [item for
sublist in evaluated_data for item in
102                                 (sublist if
isinstance(sublist, list) else [sublist])]
```

```
103             except (ValueError,
SyntaxError):
104                 return data.split(", ") if
", " in data else data.split()
105         return []
106
107     def
process_coursera_data(self):
108                 try:
109             self.coursera_data =
pd.read_csv(self.coursera_data_path)
110             self.coursera_data =
self.coursera_data.dropna(subset=['skill
s'])
111
self.coursera_data['embedding'] =
self.coursera_data['skills'].apply(
112                 lambda x:
model.encode(str(x),
convert_to_tensor=True))
113
114             print(f"Successfully
processed {len(self.coursera_data)}
Coursera courses")
115         except Exception as e:
116             print(f"Error processing
Coursera data: {e}")
117
118     def analyze_skill_gap(self,
```

```
candidate_number, interactive=True):
119                 try:
120             candidate_index =
candidate_number - 1
121             applied_positions =
self.positions_applied[candidate_index]
122             candidate_skills =
set(self.skills_of_candidate[candidate_i
ndex])
123
124             if interactive:
125                 print(f"\nCandidate
{candidate_number} applied for these
positions:")
126                 for idx, pos in
enumerate(applied_positions):
127                     print(f"  {idx + 1}.
{pos}")
128
129                 selected_index =
int(input("\nEnter the number
corresponding to the position you want
to match: ")) - 1
130                 if selected_index < 0 or
selected_index >=
len(applied_positions):
131                     print("Invalid
selection. Exiting.")
132                     return None
```

```
133             else:
134                 selected_index = 0
135
136             applied_position =
applied_positions[selected_index]
137
138             if interactive:
139                 print(f"\nSelected
Position for Matching:
{applied_position}")
140                 print(f"Candidate's
skills: {candidate_skills}\n")
141             matched_jobs = {}
142
143             for idx, job_title in
enumerate(self.job_titles_combined):
144                 job_words = set("
".join(job_title).lower().split()) if
isinstance(job_title, list) else set(
145
job_title.lower().split())
146                 applied_words =
set(applied_position.lower().split())
147                 if job_words &
applied_words:
148                     required_skills =
set(self.skills_and_keywords[idx])
149                     missing_skills =
required_skills - candidate_skills
```

```
150
151                     job_title_str = " /
".join(job_title) if isinstance(job_title,
list) else job_title
152
153                     if job_title_str in
matched_jobs:
154
matched_jobs[job_title_str]["required_
skills"].update(required_skills)
155
matched_jobs[job_title_str]["missing_s
kills"].update(missing_skills)
156                     else:
157
matched_jobs[job_title_str] = {
158
"required_skills": required_skills,
159
"missing_skills": missing_skills
160                         }
161
162             if not matched_jobs:
163                 if interactive:
164                     print("\nNo
matching jobs found.")
165                 return None
166             unique_jobs_list =
list(matched_jobs.keys())
```

```
167
168             if interactive:
169                 print("\nUnique
Matched Jobs:")
170                 for i, job_title in
enumerate(unique_jobs_list):
171                     print(f"  {i + 1}.
{job_title}")
172
173                 job_selection =
int(input("\nEnter the number of the
job you want to analyze: ")) - 1
174                 if job_selection < 0 or
job_selection >= len(unique_jobs_list):
175                     print("Invalid
selection. Exiting.")
176                     return None
177             else:
178                 job_selection = 0
179
180             selected_job_title =
unique_jobs_list[job_selection]
181             selected_job =
matched_jobs[selected_job_title]
182
183             if interactive:
184                 print(f"\n 🔹 Selected
Matched Job Title:
{selected_job_title}")
```

```
185                 print(f"  ✅ Required
Skills:
{sorted(selected_job['required_skills'])
}")
186                 print(f"  ❌ Missing
Skills:
{sorted(selected_job['missing_skills'])}
\n")
187             recommended_courses =
[]
188
189             for skill in
selected_job['missing_skills']:
190                 skill_embedding =
model.encode(skill,
convert_to_tensor=True)
191
self.coursera_data['similarity'] =
self.coursera_data['embedding'].apply(
192                     lambda x:
util.cos_sim(skill_embedding,
x).item())
193
194                 best_matches =
self.coursera_data.nlargest(3,
'similarity')
195                 for _, course in
best_matches.iterrows():
196                     if interactive:
```

```
197                     print(
198                         f"  - Partner:
{course['partner']} | Course:
{course['course']} | Similarity Score:
{course['similarity']:.4f}")
199
200
recommended_courses.append({
201                     'partner':
course['partner'],
202                     'course':
course['course'],
203                     'similarity':
course['similarity']
204                 })
205             results = {
206                 'candidate_number':
candidate_number,
207                 'applied_position':
applied_position,
208                 'candidate_skills':
list(candidate_skills),
209                 'selected_job_title':
selected_job_title,
210                 'required_skills':
list(selected_job['required_skills']),
211                 'missing_skills':
list(selected_job['missing_skills']),
212
```

```
'recommended_courses':
recommended_courses
213             }
214
215             return results
216
217         except Exception as e:
218             print(f"Error in skill gap
analysis: {e}")
219             return None
220
221     def get_missing_skills(self,
candidate_number, position_index=0):
222                 if hasattr(self,
'test_data') and not
self.test_data.empty:
223             if candidate_number - 1
>= len(self.test_data):
224                 return []
225
226             applied_position =
self.test_data.iloc[candidate_number -
1]['position_applied']
227             candidate_skills = set()
228         else:
229             candidate_index =
candidate_number - 1
230             if candidate_index < 0 or
candidate_index >=
```

```
len(self.positions_applied):
231                 return []
232
233             applied_positions =
self.positions_applied[candidate_index]
234             if position_index < 0 or
position_index >=
len(applied_positions):
235                 return []
236
237             applied_position =
applied_positions[position_index]
238             candidate_skills =
set(self.skills_of_candidate[candidate_i
ndex])
239         if pd.isna(applied_position):
240             return []
241
242         applied_words =
set(applied_position.lower().split())
243         all_required_skills = set()
244         for idx, job_title in
enumerate(self.job_titles_combined):
245             job_words = set("
".join(job_title).lower().split()) if
isinstance(job_title, list) else set(
246
job_title.lower().split())
247             if job_words &
```

```
applied_words:
248
all_required_skills.update(self.skills_an
d_keywords[idx])
249
250             missing_skills =
all_required_skills - candidate_skills
251             return list(missing_skills)
252
253         def
get_recommended_courses(self,
missing_skills):
254                     recommended_courses
= []
255
256             for skill in missing_skills:
257                 skill_embedding =
model.encode(skill,
convert_to_tensor=True)
258
self.coursera_data['similarity'] =
self.coursera_data['embedding'].apply(
259                     lambda x:
util.cos_sim(skill_embedding,
x).item())
260
261                 best_matches =
self.coursera_data.nlargest(1,
'similarity')
```

```
262                 for _, course in
best_matches.iterrows():
263
recommended_courses.append(f"{cour
se['partner']}: {course['course']}")
264
265             return
recommended_courses
266
267
268     class SkillGapSystemEvaluator:
269         def __init__(self, system,
test_data=None, ground_truth=None):
270                     self.system = system
271             self.test_data = test_data
272             self.ground_truth =
ground_truth
273
274         def
create_evaluation_dataset(self,
num_candidates=50):
275                     all_skills = [
276                 'python', 'java', 'c++',
'javascript', 'html', 'css', 'sql', 'nosql',
277                 'react', 'angular', 'node.js',
'django', 'flask', 'docker', 'kubernetes',
278                 'aws', 'gcp', 'azure', 'data
analysis', 'machine learning', 'deep
learning',
```

```
279             'natural language
processing', 'computer vision',
'statistics', 'calculus',
280             'linear algebra', 'database
design', 'data modeling', 'data
visualization',
281             'tableau', 'power bi',
'excel', 'project management', 'agile',
'scrum'
282         ]
283
284         all_courses = [
285             'Python Programming',
'Java Development', 'C++
Fundamentals',
286             'JavaScript for Web
Development', 'HTML and CSS', 'SQL
Database Management',
287             'NoSQL Database
Systems', 'React.js', 'Angular
Framework', 'Node.js Development',
288             'Django Web
Framework', 'Flask API Development',
'Docker Containers',
289             'Kubernetes
Orchestration', 'AWS Cloud Services',
'Google Cloud Platform',
290             'Microsoft Azure', 'Data
Analysis with Python', 'Machine
```

```
Learning Fundamentals',
291             'Deep Learning with
TensorFlow', 'NLP with Python',
'Computer Vision Applications',
292             'Statistics for Data
Science', 'Calculus for Machine
Learning',
293             'Linear Algebra
Essentials', 'Database Design
Principles', 'Data Modeling
Techniques',
294             'Data Visualization Best
Practices', 'Tableau Dashboarding',
'Power BI Analytics',
295             'Advanced Excel', 'Project
Management', 'Agile Development',
'Scrum Master Certification'
296         ]
297
298         all_positions = [
299             'Software Engineer', 'Data
Scientist', 'Web Developer', 'Frontend
Developer',
300             'Backend Developer',
'Full Stack Developer', 'DevOps
Engineer', 'Cloud Engineer',
301             'Machine Learning
Engineer', 'AI Researcher', 'Database
Administrator',
```

```
302             'Data Analyst', 'Business
Intelligence Analyst', 'Project Manager'
303         ]
304         candidate_ids = list(range(1,
num_candidates + 1))
305         test_data = {
306             'candidate_id':
candidate_ids,
307             'position_applied':
[random.choice(all_positions) for _ in
range(num_candidates)]
308         }
309         ground_truth = {
310             'candidate_id': [],
311             'position_applied': [],
312             'required_skills': [],
313             'missing_skills': [],
314             'recommended_courses':
[]
315         }
316
317         for cid, position in
zip(test_data['candidate_id'],
test_data['position_applied']):
318             if 'Software' in position or
'Developer' in position:
319                 req_skills =
random.sample(all_skills[:15],
random.randint(5, 10))
```

```
320             elif 'Data' in position or
'Machine' in position or 'AI' in position:
321                 req_skills =
random.sample(all_skills[15:],
random.randint(5, 10))
322             else:
323                 req_skills =
random.sample(all_skills,
random.randint(5, 10))
324             missing_skills =
random.sample(req_skills,
random.randint(2, min(5,
len(req_skills))))
325             rec_courses = []
326             for skill in missing_skills:
327                 matching_courses =
[course for course in all_courses if
skill.lower() in course.lower()]
328                 if matching_courses:
329
rec_courses.append(random.choice(mat
ching_courses))
330             if not rec_courses or
len(rec_courses) < len(missing_skills):
331                 additional_courses =
random.sample(all_courses, max(1,
len(missing_skills) - len(rec_courses)))
332
rec_courses.extend(additional_courses)
```

```
                rec_courses =
list(set(rec_courses))[:len(missing_skill
s)]

ground_truth['candidate_id'].append(ci
d)

ground_truth['position_applied'].appen
d(position)

ground_truth['required_skills'].append(
req_skills)

ground_truth['missing_skills'].append(
missing_skills)

ground_truth['recommended_courses'].
append(rec_courses)
        test_df =
pd.DataFrame(test_data)
        ground_truth_df =
pd.DataFrame(ground_truth)

        self.test_data = test_df
        self.ground_truth =
ground_truth_df

        return test_df,
ground_truth_df
```

```
346
347     def evaluate(self):
348             if self.test_data is None
or self.ground_truth is None:
349             print("Test data or ground
truth not available. Creating synthetic
data...")
350
self.create_evaluation_dataset()
351         results = {
352             'candidate_id': [],
353             'position_applied': [],
354             'predicted_skills_needed':
[],
355             'actual_skills_needed': [],
356             'predicted_courses': [],
357             'actual_courses': [],
358             'skill_precision': [],
359             'skill_recall': [],
360             'skill_f1': [],
361             'course_relevance_score':
[],
362             'satisfaction_score': []
363         }
364         for _, candidate in
self.test_data.iterrows():
365             candidate_id =
candidate['candidate_id']
366             position =
```

```
candidate['position_applied']
367             gt =
self.ground_truth[(self.ground_truth['ca
ndidate_id'] == candidate_id) &
368
(self.ground_truth['position_applied']
== position)]
369
370             if gt.empty:
371                 continue
372
373             gt = gt.iloc[0]
374             predicted_skills =
self.system.get_missing_skills(candidat
e_id)
375             actual_skills =
gt['missing_skills']
376             precision, recall, f1 =
self.calculate_skill_metrics(predicted_s
kills, actual_skills)
377             predicted_courses =
self.system.get_recommended_courses(
predicted_skills)
378             actual_courses =
gt['recommended_courses']
379             course_relevance =
self.calculate_course_relevance(predict
ed_courses, actual_courses)
380             satisfaction =
```

```
self.calculate_satisfaction_score(precisi
on, recall, course_relevance)
381
results['candidate_id'].append(candidat
e_id)
382
results['position_applied'].append(posit
ion)
383
results['predicted_skills_needed'].appen
d(predicted_skills)
384
results['actual_skills_needed'].append(a
ctual_skills)
385
results['predicted_courses'].append(pre
dicted_courses)
386
results['actual_courses'].append(actual_
courses)
387
results['skill_precision'].append(precisi
on)
388
results['skill_recall'].append(recall)
389
results['skill_f1'].append(f1)
390
results['course_relevance_score'].appen
```

```
d(course_relevance)
391
results['satisfaction_score'].append(sati
sfaction)
392             results_df =
pd.DataFrame(results)
393             overall_metrics = {
394                 'avg_skill_precision':
results_df['skill_precision'].mean(),
395                 'avg_skill_recall':
results_df['skill_recall'].mean(),
396                 'avg_skill_f1':
results_df['skill_f1'].mean(),
397                 'avg_course_relevance':
results_df['course_relevance_score'].m
ean(),
398                 'avg_satisfaction':
results_df['satisfaction_score'].mean()
399             }
400             cm =
self.generate_confusion_matrix(results
_df)
401
402             return results_df,
overall_metrics, cm
403
404         def
calculate_skill_metrics(self,
predicted_skills, actual_skills):
```

```
405                 pred_set =
set(predicted_skills)
406             actual_set =
set(actual_skills)
407             tp =
len(pred_set.intersection(actual_set))
408             fp = len(pred_set -
actual_set)
409             fn = len(actual_set -
pred_set)
410             precision = tp / (tp + fp) if
(tp + fp) > 0 else 0
411             recall = tp / (tp + fn) if (tp +
fn) > 0 else 0
412             f1 = 2 * (precision * recall)
/ (precision + recall) if (precision +
recall) > 0 else 0
413
414             return precision, recall, f1
415
416         def
calculate_course_relevance(self,
predicted_courses, actual_courses):
417                 if not
predicted_courses or not
actual_courses:
418                     return 0
419             similarities = []
420             for pred_course in
```

```
predicted_courses:
421             course_sims = []
422             pred_embedding =
model.encode(pred_course,
convert_to_tensor=True)
423
424             for actual_course in
actual_courses:
425                 actual_embedding =
model.encode(actual_course,
convert_to_tensor=True)
426                 sim =
util.cos_sim(pred_embedding,
actual_embedding).item()
427
course_sims.append(sim)
428
429
similarities.append(max(course_sims)
if course_sims else 0)
430         return sum(similarities) /
len(similarities) if similarities else 0
431
432     def
calculate_satisfaction_score(self,
precision, recall, course_relevance,
weights=(0.3, 0.3, 0.4)):
433                 return weights[0] *
precision + weights[1] * recall +
```

```
weights[2] * course_relevance
434
435       def
generate_confusion_matrix(self,
results_df):
436                   all_predicted = set()
437           all_actual = set()
438           true_pos = 0
439           false_pos = 0
440           false_neg = 0
441           true_neg = 0
442           for pred, actual in
zip(results_df['predicted_skills_needed'
], results_df['actual_skills_needed']):
443
all_predicted.update(pred)
444               all_actual.update(actual)
445
446           all_skills =
all_predicted.union(all_actual)
447           for pred, actual in
zip(results_df['predicted_skills_needed'
], results_df['actual_skills_needed']):
448               pred_set = set(pred)
449               actual_set = set(actual)
450
451               for skill in all_skills:
452                   if skill in pred_set and
skill in actual_set:
```

```
453                     true_pos += 1
454                 elif skill in pred_set
and skill not in actual_set:
455                     false_pos += 1
456                 elif skill not in
pred_set and skill in actual_set:
457                     false_neg += 1
458                 else:
459                     true_neg += 1
460         cm = np.array([
461             [true_pos, false_neg],
462             [false_pos, true_neg]
463         ])
464
465         return cm
466
467     def visualize_evaluation(self,
results_df, overall_metrics, cm):
468                 plt.figure(figsize=(10,
8))
469         sns.heatmap(cm,
annot=True, fmt='d', cmap='Blues',
470
xticklabels=['Predicted Needed',
'Predicted Not Needed'],
471
yticklabels=['Actually Needed',
'Actually Not Needed'])
472         plt.title('Confusion Matrix
```

```
for Skills Prediction')
473         plt.tight_layout()
474
plt.savefig('confusion_matrix.png')
475         print("Saved confusion
matrix visualization to
'confusion_matrix.png'")
476         plt.figure(figsize=(12, 6))
477         x = range(len(results_df))
478         plt.plot(x,
results_df['skill_precision'], 'o-',
label='Precision')
479         plt.plot(x,
results_df['skill_recall'], 's-',
label='Recall')
480         plt.plot(x,
results_df['course_relevance_score'],
'p-', label='Course Relevance')
481         plt.plot(x,
results_df['satisfaction_score'], 'D-',
label='Satisfaction')
482         plt.xlabel('Candidate Index')
483         plt.ylabel('Score')
484         plt.title('Evaluation Metrics
by Candidate')
485         plt.legend()
486         plt.grid(True, alpha=0.3)
487         plt.tight_layout()
488
```

```
plt.savefig('metrics_by_candidate.png')
489         print("Saved metrics
visualization to
'metrics_by_candidate.png'")
490         plt.figure(figsize=(10, 6))
491         metrics =
list(overall_metrics.keys())
492         values =
list(overall_metrics.values())
493
494         plt.bar(metrics, values,
color='skyblue')
495         plt.xticks(rotation=45,
ha='right')
496         plt.ylabel('Score')
497         plt.title('Overall System
Performance Metrics')
498         plt.ylim(0, 1)
499
500         for i, v in
enumerate(values):
501             plt.text(i, v + 0.02,
f'{v:.3f}', ha='center')
502
503         plt.tight_layout()
504
plt.savefig('overall_metrics.png')
505         print("Saved overall metrics
visualization to 'overall_metrics.png'")
```

```
506
507
508    def main():
509        job_data_path =
r"D:\Research\dataset\job_descriptions.
csv"
510        resume_data_path =
r"D:\Research\dataset\resume_data.csv
"
511        coursera_data_path =
r"D:\Coursera.csv"
512
513        print("\n===== Skill Gap
Analysis System Evaluation =====\n")
514        print("Choose an option:")
515        print("1. Run the system
interactively for a single candidate")
516        print("2. Evaluate system
performance with synthetic data")
517
518        choice = input("\nEnter your
choice (1/2): ")
519
520        try:
521            print("\nInitializing Skill
Gap Analysis System...")
522            system =
SkillGapAnalysisSystem(job_data_pat
h, resume_data_path,
```

```
coursera_data_path)
523
524         if choice == '1':
525             candidate_number =
int(input("\nEnter candidate number:
"))
526
system.analyze_skill_gap(candidate_nu
mber, interactive=True)
527
528         elif choice == '2':
529             print("\nEvaluating
system performance with synthetic
data...")
530             evaluator =
SkillGapSystemEvaluator(system)
531             num_candidates =
int(input("\nEnter number of synthetic
candidates to generate (recommended
10-50): "))
532             test_data, ground_truth =
evaluator.create_evaluation_dataset(nu
m_candidates=num_candidates)
533
test_data.to_csv('synthetic_test_data.cs
v', index=False)
534
ground_truth.to_csv('synthetic_ground
_truth.csv', index=False)
```

```
535             print("\nGenerated
synthetic data and saved to CSV files.")
536             try:
537                 print("\nRunning
evaluation...")
538                 results_df,
overall_metrics, cm =
evaluator.evaluate()
539             except Exception as e:
540                 print("Detailed error
info:")
541                 traceback.print_exc()
542
results_df.to_csv('evaluation_results.cs
v', index=False)
543             print("\n===== Overall
System Performance =====")
544             for metric, value in
overall_metrics.items():
545                 print(f"{metric}:
{value:.4f}")
546
547             print("\n=====
Confusion Matrix =====")
548             print(cm)
549             print("\nCreating
visualizations...")
550
evaluator.visualize_evaluation(results_
```

```
df, overall_metrics, cm)
551
552
553             else:
554                 print("Invalid choice.
Please run the script again and choose 1
or 2.")
555
556         except Exception as e:
557             print(f"An error occurred:
{e}")
558
559         print("\nEvaluation complete!
Results saved to CSV and
visualizations saved as PNG files.")
560
561     if __name__ == "_main_":
562         main()
```