TABLE OF CONTENTS

ABSTRACT

This project showcases the development and implementation of an AI-powered workforce planning system, designed to revolutionize HR decision-making and talent management strategies within modern organizations. Delivered as an intuitive and accessible Streamlit application, the system offers data-driven insights through two key functionalities: promotion prediction and personalized role recommendations. Firstly, a high-performing promotion prediction model, built upon a TensorFlow feed-forward neural network architecture, achieves a remarkable 91% accuracy in identifying employees who are primed for advancement. This capability enables HR professionals to make informed decisions regarding promotions, fostering employee retention and maximizing organizational growth potential. Secondly, a sophisticated role recommendation model leverages collaborative filtering techniques combined with cosine similarity, implemented using the PyTorch framework. To enhance the accuracy and relevance of role suggestions, the model employs sentence transformer embeddings of both resumes and job descriptions, creating a semantic understanding of skills and requirements. Users can seamlessly upload their resumes to the Streamlit application and receive tailored role recommendations that align with their experience, qualifications, and career aspirations. This feature empowers employees to proactively manage their career paths and seek opportunities for professional development within the organization. The project demonstrates the tangible benefits of integrating AI into HR processes, streamlining workflows, improving the accuracy of workforce planning, and facilitating a more engaged and productive workforce. The Streamlit interface ensures ease of use and accessibility for HR teams, enabling them to leverage sophisticated analytics without requiring specialized technical expertise. By offering both predictive capabilities and personalized recommendations, this system empowers organizations to optimize resource allocation, foster employee growth, and ultimately achieve their strategic goals.

CHAPTER 1 INTRODUCTION

1.1 PROJECT BACKGROUND

In the rapidly evolving digital economy, organizations face unprecedented challenges in workforce planning and talent management. The intersection of data science and human resource management has created new opportunities to transform traditional approaches to employee development, promotion decisions, and talent acquisition. The AI Workforce Planning Tools project emerged from the recognition that organizations collect vast amounts of employee and recruitment data but often lack the sophisticated tools needed to extract actionable insights from this information.Traditional workforce planning methods rely heavily on subjective assessments, manual processes, and limited analytical capabilities. These approaches are becoming increasingly inadequate in today's competitive talent landscape, where organizations must make data-driven decisions quickly and accurately to attract, develop, and retain high-performing employees. Similarly, job seekers navigate an increasingly complex application process, often lacking clear guidance on how to effectively present their qualifications or identify roles that align with their skills and experience.

1.2 PROJECT OBJECTIVES

The AI Workforce Planning Tools project aims to achieve several key objectives:

1. **Develop an accurate promotion prediction system** that analyzes multiple employee factors to forecast promotion readiness with greater consistency and less bias than traditional methods.
2. **Create an intelligent role recommendation engine** that matches candidate resumes to job descriptions using semantic understanding and collaborative filtering techniques, providing quantitative assessment of compatibility.
3. **Implement an AI-powered cover letter generation system** that automatically produces tailored, professional application documents based on resume content and job requirements.
4. **Design an intuitive web interface** that makes sophisticated analytical capabilities accessible to HR professionals and job seekers without requiring technical expertise.
5. **Demonstrate the practical application** of modern AI and machine learning technologies to solve real-world workforce planning challenges.

6. **Improve decision-making quality** by replacing subjective assessments with data-driven insights based on historical patterns and comprehensive analysis.
7. **Enhance efficiency** by automating time-consuming manual processes, allowing HR departments to process more information and make faster decisions.

1.3 SCOPE OF THE PROJECT

The AI Workforce Planning Tools project encompasses the development of a comprehensive web-based application with three primary modules:

1. **Promotion Prediction Model**: A TensorFlow-based neural network that analyzes employee data to predict promotion likelihood, considering factors such as department, performance metrics, training scores, and tenure. The model provides both probability scores and transparent factor analysis to enhance understanding of prediction rationale.
2. **Role Recommendation System**: A sophisticated candidate-job matching system using sentence transformers and collaborative filtering to analyze resume content against job descriptions. The system provides detailed compatibility metrics, skill gap analysis, and actionable recommendations for improvement.
3. **Cover Letter Generation Module**: An LLM-powered system that automatically creates customized cover letters by analyzing both resume content and job descriptions, producing professional-quality documents tailored to specific opportunities.

The project includes the development of all necessary components:

- Data preprocessing pipelines for structured and unstructured text
- Machine learning model architecture, training, and evaluation
- Natural language processing for semantic text understanding
- User interface design and implementation using Streamlit
- Integration with external APIs (Google Gemini) for generative AI capabilities
- Performance optimization for responsive user experience
- Deployment configurations for scalable access

CHAPTER 2 SYSTEM ANALYSIS

2.1 PROBLEM DEFINITION

In today's rapidly evolving business landscape, organizations face unprecedented challenges in workforce planning and talent management. Traditional human resource practices, while historically adequate, have become increasingly insufficient to address the complexities of modern workforce dynamics. The AI Workforce Planning Tools project emerged from a comprehensive analysis of several critical problems inherent in conventional workforce management approaches:

1. **Subjective and Inconsistent Promotion Decision Processes**: Organizations typically rely on annual performance reviews, manager recommendations, and often subjective assessments when making promotion decisions. This approach suffers from several critical limitations:

- Susceptibility to unconscious bias and favoritism among decision-makers
- Inconsistent evaluation criteria across departments and managers
- Over-emphasis on recent performance rather than long-term potential

These limitations not only affect organizational effectiveness but also impact employee morale, retention, and career development. Studies indicate that perceived unfairness in promotion processes is among the top reasons for employee turnover.

2. **Manual and Inefficient Resume-Job Matching Techniques**: Human resource departments invest substantial time and resources in manually matching candidates to open positions:

- HR professionals spend an average of 23 hours per week on resume screening
- Typical corporate job openings attract 250+ resumes, with only 4-6 candidates receiving interviews
- Manual screening misses approximately 35% of qualified candidates

These inefficiencies result in prolonged hiring cycles, increased recruitment costs, potential talent loss, and suboptimal matching of candidates to positions where they could provide the greatest value.

3. **Time-Consuming and Ineffective Cover Letter Creation**: Job seekers face significant challenges in creating effective cover letters:

- Average time spent is 2-4 hours per customized cover letter
- Difficulty in effectively highlighting relevant qualifications for specific positions

These challenges reduce application quality, limit candidate consideration, and create an unnecessary barrier to employment for otherwise qualified individuals.

4. **Underutilization of Workforce Data**: Despite collecting extensive data, organizations frequently fail to extract actionable insights:

- Disconnected data silos across HR, training, performance, and operations systems
- Limited capabilities to process and analyze large volumes of workforce data
- Absence of predictive analytics in talent management decisions

5. **Lack of Quantitative Assessment Frameworks**: Traditional workforce management systems typically provide limited quantitative measures for evaluation:

- Absence of standardized metrics for promotion readiness
- Subjective performance ratings with limited comparability

These interconnected problems create substantial inefficiencies in workforce management, limiting organizational agility, increasing costs, and potentially restricting access to optimal talent. The financial impact is significant, with studies indicating that poor promotion decisions can cost organizations up to 300% of the employee's annual salary when considering productivity loss, replacement costs, and missed opportunities.

2.2 EXISTING SYSTEM

Current workforce planning and talent management systems represent a spectrum of approaches, from entirely manual processes to basic digital systems with limited automation and intelligence. A comprehensive analysis of these existing systems reveals significant limitations that impact their effectiveness in modern organizational environments:

1. **Traditional Promotion Management Systems**:

Current approaches to promotion management typically involve a combination of the following elements:

- **Annual Performance Review Cycles**: Organizations rely on standardized review periods (typically annual or bi-annual) where managers evaluate employees against pre-

defined criteria. These reviews often use 5-point scales or similar rating systems that reduce complex performance attributes to simplified numerical scores.

- **Manager Recommendation Workflows**: Promotion candidates are identified through a hierarchical nomination process where direct supervisors recommend high-performing team members to senior management. This creates a dependency on individual managers' ability to recognize and advocate for talent.
- **Manual Tracking Systems**: Employee metrics are maintained in spreadsheets or basic Human Resource Information Systems (HRIS) with limited analytical capabilities. These systems track fundamental data points but struggle to identify patterns or make predictions.

These traditional systems suffer from several critical limitations:

- **Recency Bias**: Greater weight is given to recent performance rather than consistent contribution over time
- **Halo Effect**: Overall impression of an employee influences the assessment of specific attributes
- **Documentation Gaps**: Inconsistent recording of achievements and development areas
- **Political Influence**: Relationship dynamics affecting objective assessment

2. **Conventional Resume Screening and Candidate Selection Systems**:

Current approaches to candidate screening include:

- **Manual Resume Review**: HR personnel physically review each resume, typically spending 6-10 seconds on initial screening to determine if a candidate proceeds to detailed evaluation.
- **Basic Applicant Tracking Systems (ATS)**: First-generation ATS platforms perform rudimentary keyword matching to filter candidates but lack semantic understanding of content. Studies show these systems reject up to 75% of qualified candidates due to formatting issues or missing exact keyword matches.

These conventional systems present significant limitations:

- **Scale Limitations**: Inability to effectively handle high volumes of applications
- **Surface-Level Analysis**: Focus on keywords rather than actual capabilities and potential

- **Inconsistent Evaluation**: Different screeners applying varying standards to candidates
- **Qualification Inflation**: Rigid requirement matching eliminating capable candidates without exact credential matches
- **Format Over Content**: Excessive focus on resume presentation rather than substance

3. **Current Cover Letter Creation Approaches**:

Job seekers typically create cover letters through:

- **Generic Templates**: Standard formats with minimal customization, often recognizable to experienced recruiters as mass-produced applications.
- **Manual Customization Process**: Job seekers individually modifying documents for each application, frequently resulting in inconsistent quality and messaging.
- **Professional Writing Services**: Third-party services creating cover letters at significant cost ($75-200 per letter), affordable only to a segment of job seekers.
- **Peer Review Feedback**: Informal review by colleagues or mentors providing subjective feedback of varying quality.
- **Trial-and-Error Approaches**: Candidates refining their approach based on response rates without structured guidance.
- **Online Templates and Examples**: Generic samples providing basic structure but limited personalization guidance.

These approaches suffer from critical limitations:

- **Content-Job Misalignment**: Difficulty in effectively highlighting the most relevant qualifications for specific positions
- **Inconsistent Quality**: Variable writing quality depending on individual skills and time investment
- **Time Inefficiency**: Substantial time investment for each application reducing total application volume
- **Assessment Challenges**: No objective way to evaluate effectiveness before submission

4. **Existing Workforce Analytics Platforms**:

Current workforce analytics systems typically feature:

- **Descriptive Dashboards**: Visualization of historical workforce metrics without predictive capabilities.
- **Segmented Reporting Tools**: Reports focused on specific HR domains (recruitment, training, performance) without integrated analysis.

These existing analytics approaches have significant limitations:

- **Retrospective Focus**: Emphasis on what has happened rather than what will happen
- **Insight-Action Gap**: Limited guidance on how to apply analytical findings
- **Technical Barriers**: Specialized skills required to extract meaningful insights
- **Data Quality Issues**: Inconsistent data collection affecting analysis reliability

These existing systems, while functional at a basic level, increasingly struggle to meet the demands of modern organizations facing rapid market changes, talent shortages in critical areas, and the need for data-driven decision-making. The limitations inherent in these approaches create organizational inefficiencies, increase costs, extend hiring timelines, and potentially result in suboptimal talent decisions that impact business performance.

2.3 PROPOSED SYSTEM

The AI Workforce Planning Tools project introduces a comprehensive, integrated solution to address the limitations of traditional workforce management approaches. The system leverages cutting-edge artificial intelligence, machine learning, and natural language processing technologies to transform how organizations make promotion decisions, match candidates to roles, and support job seekers in creating compelling application materials.

System Overview

The proposed system consists of three primary modules, each addressing a specific aspect of workforce planning and talent management:

1. **AI-Driven Promotion Prediction System**

This module applies advanced machine learning techniques to predict promotion readiness with significantly higher accuracy and consistency than traditional methods. The system:

- **Utilizes Neural Network Architecture**: Implements a sophisticated TensorFlow-based neural network model trained on historical promotion data to identify complex patterns and relationships that human evaluators might miss.

- **Incorporates Comprehensive Factors**: Analyzes multiple variables simultaneously, including department, region, education level, gender, recruitment channel, training participation, age, performance ratings, service length, award recognition, and training scores.
- **Provides Probability Scores**: Generates numerical promotion likelihood scores (0-1) with associated confidence levels rather than binary decisions, allowing for nuanced evaluation.

2. **Intelligent Role Recommendation System**

This module revolutionizes the resume-job matching process through advanced AI techniques that understand both job requirements and candidate qualifications at a semantic level. The system:

- **Implements Advanced Resume Parsing**: Extracts structured information from resumes in various formats (PDF, DOCX, TXT) using natural language processing techniques that recognize entities, skills, experiences, and qualifications.
- **Creates Semantic Embeddings**: Transforms both resumes and job descriptions into high-dimensional vector representations using Sentence Transformers that capture semantic meaning beyond keywords.
- **Employs Neural Matching**: Utilizes a PyTorch-based collaborative filtering neural network to calculate similarity scores between resume and job embeddings, identifying strong matches even when terminology differs.

3. **AI-Powered Cover Letter Generation**

This module leverages Google's Gemini Large Language Model (LLM) to automate the creation of high-quality, personalized cover letters tailored to specific job applications. The system:

- **Performs Bi-Directional Analysis**: Examines both the resume content and job description to identify the most relevant qualifications and alignment points.
- **Generates Customized Content**: Creates unique, non-templated cover letters that specifically address job requirements and highlight relevant candidate experiences.

CHAPTER 3 REQUIREMENT SPECIFICATION

3.1 HARDWARE SPECIFICATION

The AI Workforce Planning Tools system is designed to be lightweight and adaptable to various deployment environments while ensuring sufficient processing power for its machine learning and natural language processing components. Below are the hardware specifications for different deployment scenarios:

Minimum Hardware Requirements (Development Environment)

- **Processor**: Intel Core i5 (8th generation) or equivalent AMD processor
- **RAM**: 8GB DDR4
- **Storage**: 10GB available SSD space for application and models
- **Network**: Broadband internet connection (10+ Mbps) for API integrations
- **Display**: 1366 x 768 resolution monitor

Recommended Hardware Configuration (Production Environment)

- **Processor**: Intel Core i7/i9 (10th generation or newer) or AMD Ryzen 7/9
- **RAM**: 16GB DDR4 or higher
- **Storage**: 20GB SSD storage for application, models, and data caching
- **Network**: High-speed internet connection (50+ Mbps) for reliable API communication
- **Display**: 1920 x 1080 resolution or higher

3.2 SOFTWARE SPECIFICATION

The AI Workforce Planning Tools system leverages a variety of modern software frameworks, libraries, and technologies to deliver its functionality. This specification details the software requirements and dependencies necessary for development, deployment, and operation of the system.

Core Programming Language and Runtime Environment

- **Python**: Version 3.11+ (Primary development language)
- Chosen for its extensive data science ecosystem and machine learning libraries
- Provides excellent support for web application development with Streamlit
- Offers robust text processing capabilities essential for NLP components
- **Runtime Environment**:
- Python virtual environment (venv) for development isolation
- Containerized runtime via Docker for deployment consistency
- Support for Windows, macOS, and Linux operating systems

Web Application Framework

- **Streamlit**: Version 1.40.1+
- Provides the interactive web application interface
- Enables rapid development of data-driven user interfaces
- Supports responsive layouts for various screen sizes
- Facilitates integration of interactive visualization components
- Handles session state management for user interactions
- Supports file upload functionality for resume processing

Machine Learning and Neural Network Frameworks

- **TensorFlow**: Version 2.18.0+
- Powers the employee promotion prediction neural network
- Provides GPU acceleration capabilities when available

- Supports model serialization for persistence
- Enables incremental learning as new data becomes available
- **PyTorch**: Version 2.5.1+
- Used for the collaborative filtering model in the job recommendation system
- Provides flexible tensor operations for embedding manipulations
- Supports custom neural network architectures
- Facilitates model persistence and loading
- **Scikit-learn**: Version 1.5.2+
- Handles data preprocessing for machine learning models
- Provides implementation of various evaluation metrics
- Supports feature scaling and normalization
- Enables cross-validation during model training

Natural Language Processing and Text Analysis

- **Sentence-Transformers**:
- Creates semantic embeddings for resumes and job descriptions
- Enables similarity matching beyond keyword comparison
- Provides pre-trained models for efficient text encoding
- Supports multilingual text understanding
- **PyPDF2**:
- Extracts text content from PDF resumes
- Handles various PDF formats and structures
- Provides page-by-page text extraction
- **python-docx** and **docx2txt**:
- Processes Microsoft Word documents (.docx) for resume extraction
- Enables creation of formatted Word documents for cover letters

- Preserves document structure during processing
- **Google Generative AI (Gemini)**:
- Powers the cover letter generation functionality
- Provides context-aware text generation
- Requires API key for authentication
- Supports customization through prompt engineering

Data Manipulation and Analysis

- **Pandas**: Version 2.2.3+
- Handles structured data manipulation
- Provides DataFrame objects for efficient data operations
- Supports data transformation and filtering
- Enables CSV/Excel import and export functionality
- **NumPy**:
- Provides efficient numerical operations for machine learning models
- Supports array manipulations required for neural networks
- Handles mathematical operations on model outputs

Visualization and User Interface Components

- **Matplotlib**:
- Creates static visualizations for analysis results
- Renders charts and graphs for model interpretability
- Supports custom styling for consistent visual identity
- **WordCloud**:
- Generates visual representations of text frequency
- Provides intuitive skill and keyword visualization
- Enhances resume and job description analysis presentation

- **Custom CSS**:
- Defines styling for the web application interface
- Ensures consistent branding across application components
- Improves user experience through thoughtful design elements

File and Resource Management

- **joblib**:
- Handles serialization of machine learning models
- Provides efficient storage of trained models
- Enables fast loading of model artifacts
- **gdown**:
- Facilitates downloading models from Google Drive
- Supports initial setup and model acquisition
- Enables distribution of pre-trained models

Deployment and Containerization

- **Docker**:
- Provides containerization for consistent deployment
- Isolates application dependencies
- Simplifies scaling and distribution
- **Docker Compose** (optional):
- Orchestrates multi-container deployments
- Simplifies container networking and configuration

External API Integrations

- **Google Gemini API**:
- Requires API key stored in Streamlit secrets
- Used for generative AI functionality

- **Web Search APIs** (optional):
- Integration with LinkedIn and Naukri job search
- URL parameter handling for search queries

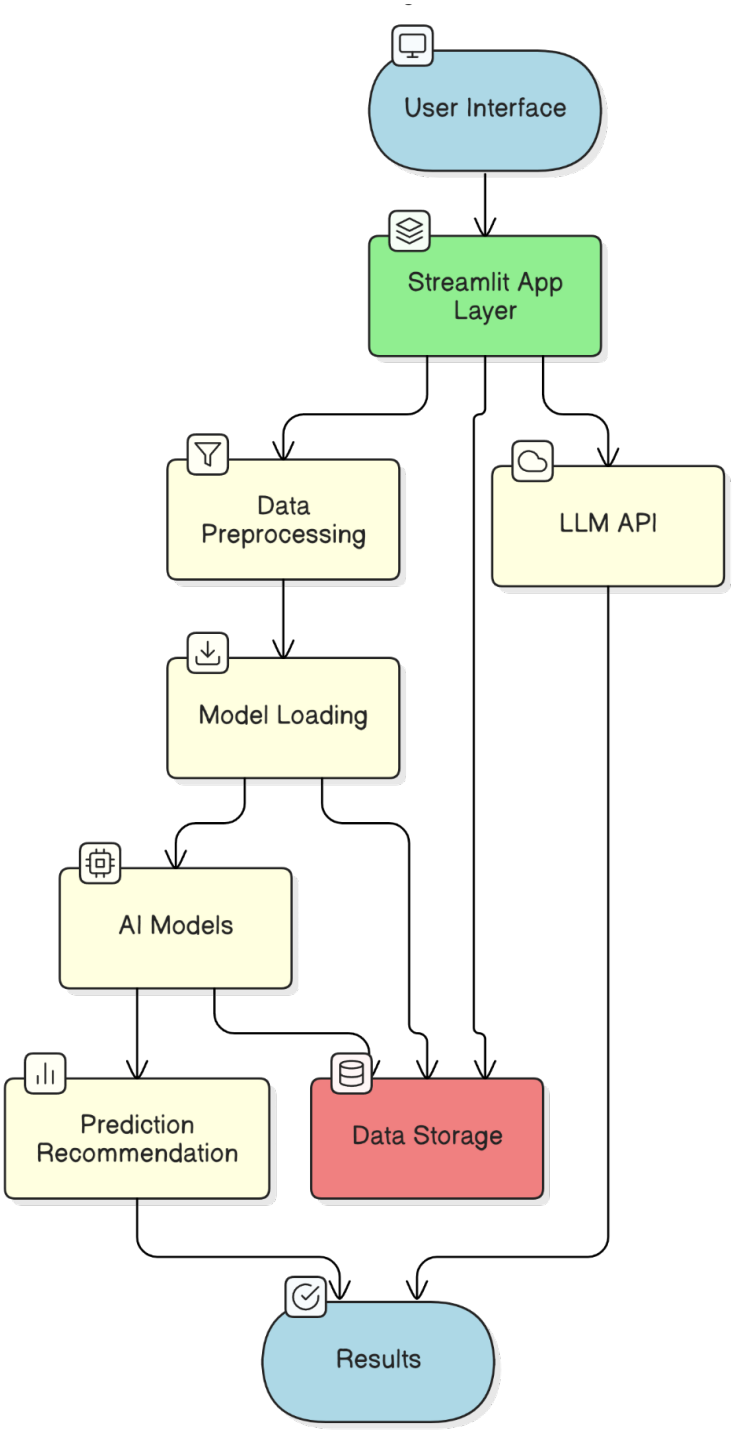
Operating System Compatibility

- **Primary Support**:
- Linux (Ubuntu 20.04+, Debian 11+)
- Windows 10/11
- macOS Big Sur (11.0+)

CHAPTER 4 DESIGN AND IMPLEMENTATION

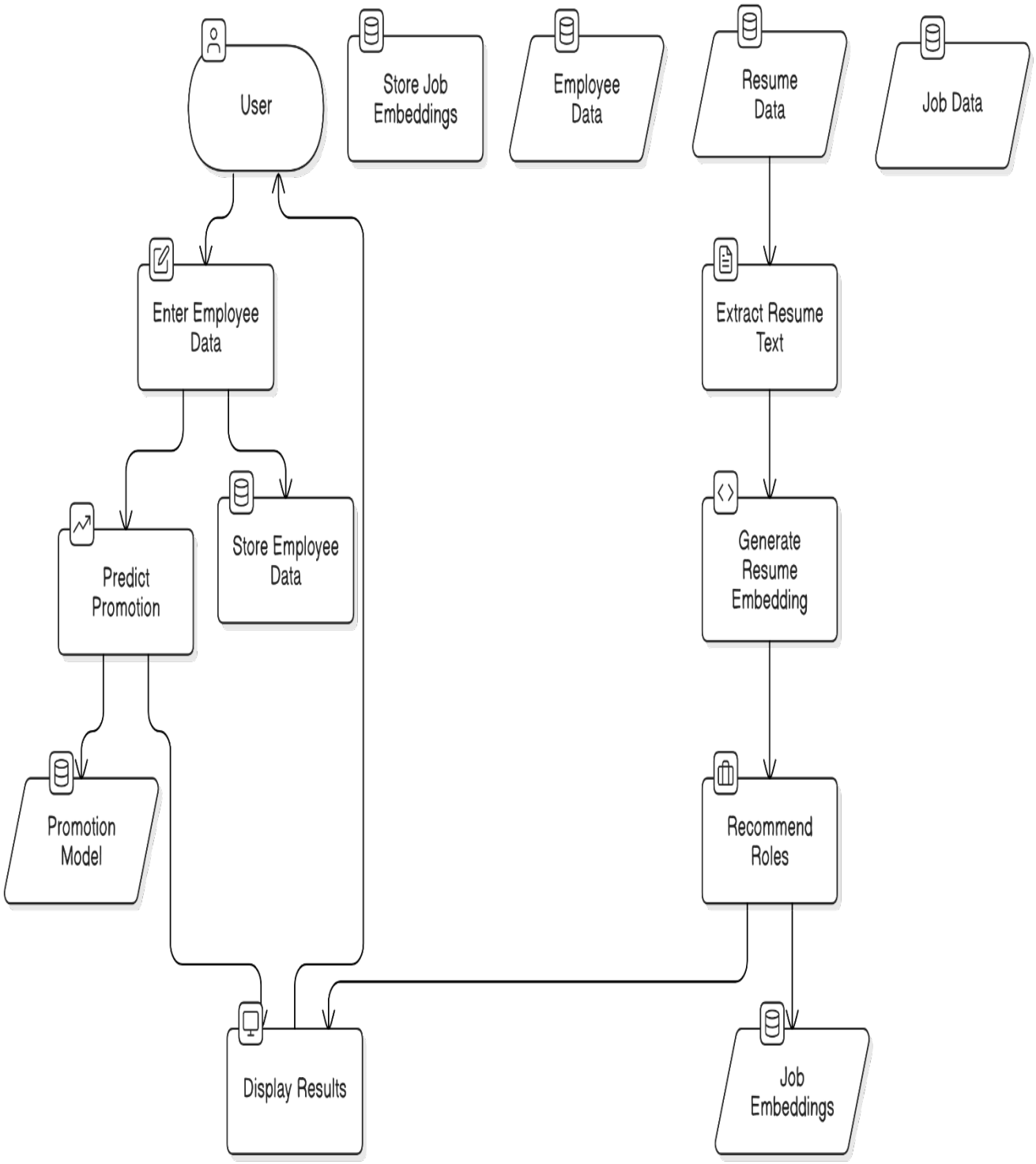
4.1 ARCHITECTURE DIAGRAM

The AI Workforce Planning Tools system follows a modular architecture that integrates multiple machine learning components with a web-based user interface. The architecture is designed to provide scalability, maintainability, and separation of concerns between different functional components.



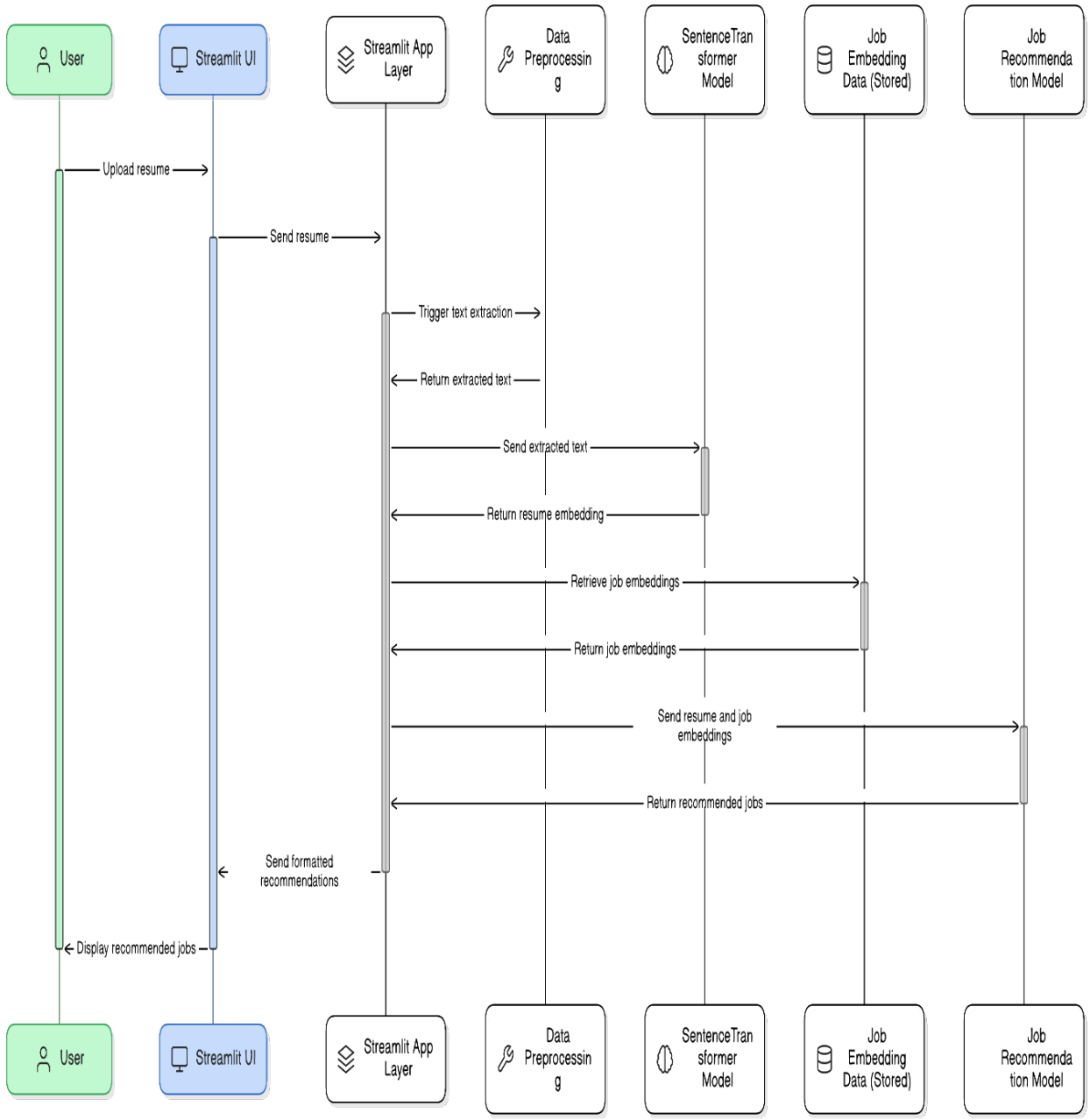
4.2 DATAFLOW DIAGRAM

The dataflow diagram illustrates how information moves through the AI Workforce Planning Tools system, from user inputs to final outputs.

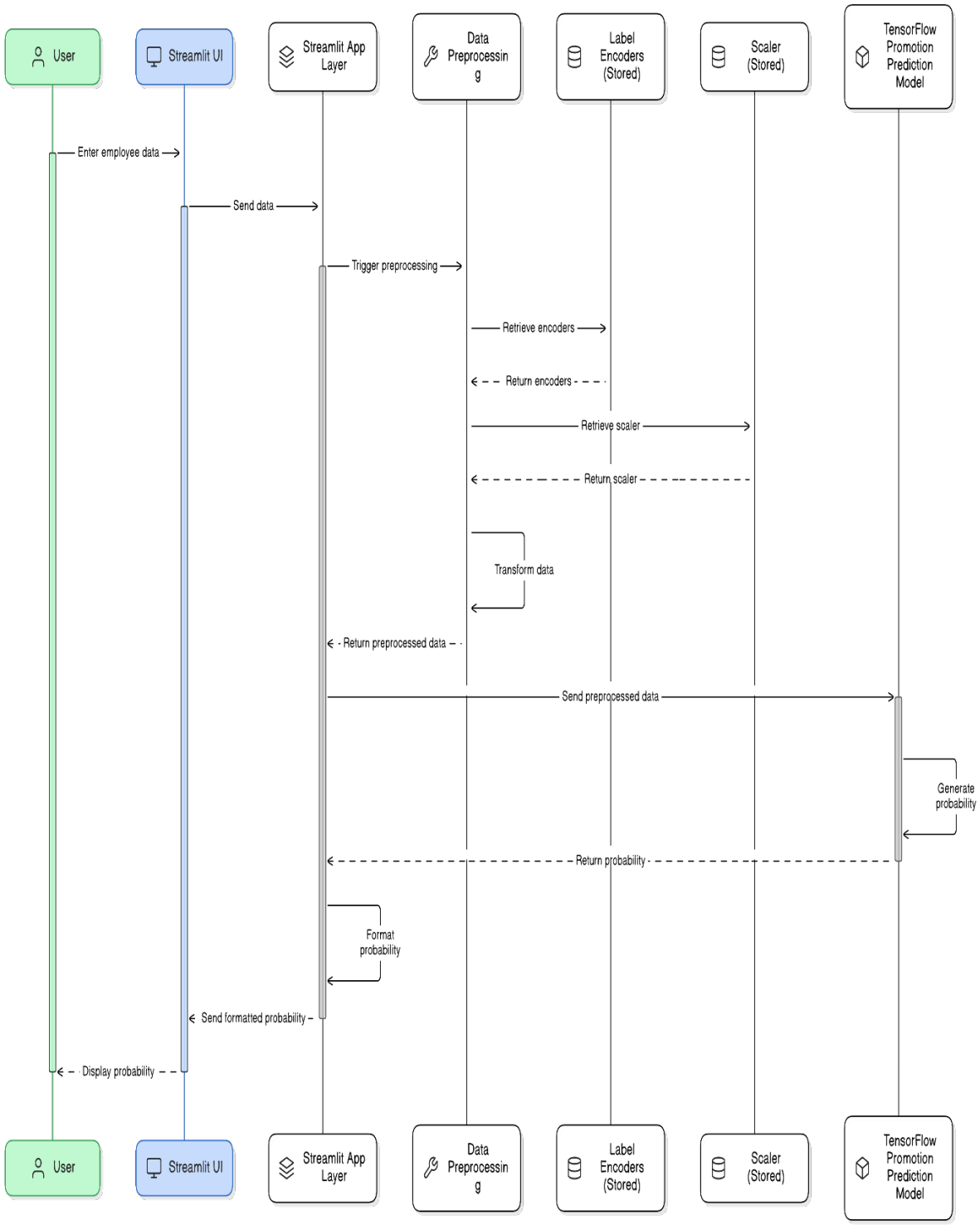


4.4 SEQUENCE DIAGRAM

The sequence diagram below illustrates the interactions between system components for the Role Recommendation process, one of the core workflows in the application.

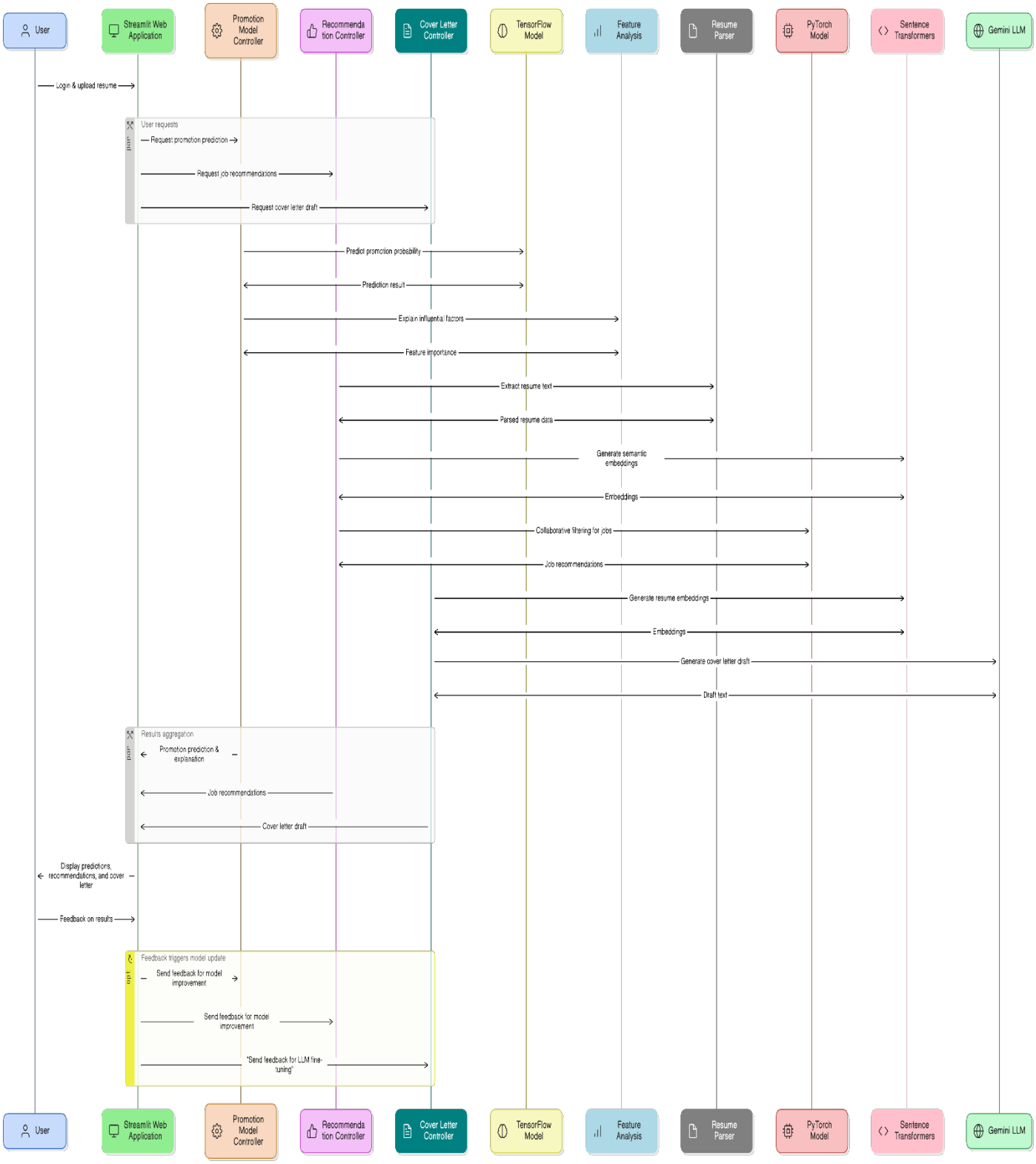


Promotion Prediction Process



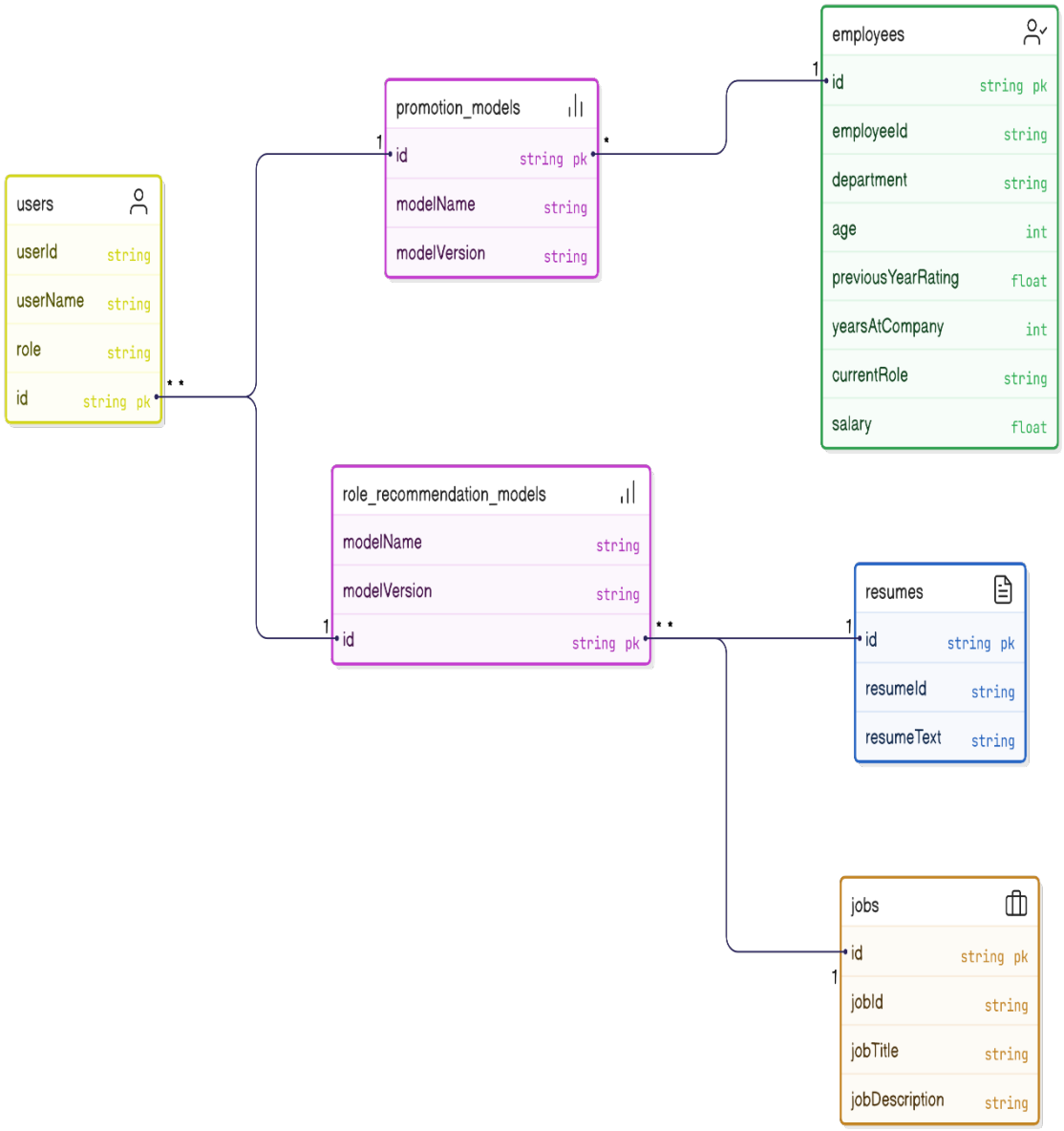
4.5 Collaborative Diagram

The collaborative diagram illustrates the relationships and interactions between the key components of the AI Workforce Planning Tools system, focusing on how they work together to deliver the system's functionality.



4.6 DATABASE DESIGN

The AI Workforce Planning Tools system primarily uses file-based storage rather than a traditional relational database. The system maintains several key data structures to support its functionality.



CHAPTER 5 TESTING

5.1 INTRODUCTION

Testing is a critical component of the AI Workforce Planning Tools project, ensuring that all modules function correctly, provide accurate results, and deliver a seamless user experience. The testing strategy for this project was designed to address the unique challenges presented by AI-driven applications, which require validation of both traditional software functionality and machine learning model performance.

Testing Objectives

The primary objectives of the testing process were to:

1. **Verify functional correctness** of all system components and their integration
2. **Validate machine learning model accuracy** and reliability across different inputs
3. **Ensure robust handling** of various file formats and text inputs
4. **Confirm intuitive user experience** across different usage scenarios
5. **Identify and resolve edge cases** and potential failure points
6. **Measure performance** under various load conditions
7. **Confirm security** of API integrations and data handling

Testing Approach

The testing approach followed a comprehensive methodology that combined traditional software testing techniques with specialized methods for evaluating AI components:

- **Progressive Testing**: Testing began with individual components and gradually expanded to integration and system-level testing
- **Automated and Manual Testing**: Combination of automated test scripts and manual verification for user interface components
- **Real-World Data Testing**: Use of authentic resumes, job descriptions, and employee data to validate real-world performance
- **Cross-Platform Validation**: Testing across different operating systems and browsers to ensure consistent functionality

- **Model Validation**: Specialized testing for machine learning models using standard metrics and custom evaluation criteria
- **User Acceptance Testing**: Involvement of HR professionals and job seekers to validate usefulness and usability

Testing Scope

The testing scope encompassed all major components of the AI Workforce Planning Tools system:

1. **Promotion Prediction Module**:

- Input validation and preprocessing
- Model prediction accuracy
- Feature importance analysis
- Result visualization
- Error handling

2. **Role Recommendation Module**:

- Resume parsing across file formats
- Text embedding generation
- Job matching and recommendation
- Compatibility analysis
- Resource generation

3. **Cover Letter Generation Module**:

- Prompt construction
- API integration reliability
- Output quality assessment
- Document formatting

4. **User Interface**:

- Responsive design validation

- Navigation functionality
- Form input handling
- File upload capabilities
- Result presentation

Testing Environment

Tests were conducted in the following environments:

- **Development Environment**:
- Local Windows 10/11, macOS, and Linux Ubuntu installations
- Python 3.11+ with Streamlit 1.40.1+
- TensorFlow 2.18.0+ and PyTorch 2.5.1+
- **Containerized Test Environment**:
- Docker containers based on Python 3.12-bookworm
- Simulated deployment environment with controlled resources
- **Cloud Testing**:
- Streamlit Cloud deployment for production-like validation
- Testing with various network conditions and user locations

5.2 TYPES OF TESTING

Several testing methodologies were employed to ensure comprehensive validation of the AI Workforce Planning Tools system:

5.2.1 Unit Testing

Unit testing focused on validating the functionality of individual functions and classes in isolation to ensure they performed as expected.

Methodology:

- Python's unittest framework was used to create automated test cases
- Test cases focused on input validation, output correctness, and error handling
- Mock objects were used to isolate dependencies for true unit testing

- Coverage analysis ensured comprehensive testing of all code paths

5.2.2 Integration Testing

Integration testing verified that different components worked correctly together, focusing on the interfaces between modules and subsystems.

Methodology:

- Tests focused on component interactions rather than individual functions
- Both automated and manual tests were employed
- Integration points between modules were systematically tested
- Data flow between components was validated for correctness

5.2.3 System Testing

System testing evaluated the entire application to ensure all components worked together correctly in a production-like environment.

Methodology:

- End-to-end testing of complete user workflows
- Testing in production-like environments
- Validation of all system functionalities
- Assessment of system behavior under various conditions

5.2.4 Performance Testing

Performance testing assessed the system's responsiveness, scalability, and resource utilization under various load conditions.

Methodology:

- Response time measurement for key operations
- Resource utilization monitoring (CPU, memory, network)
- Scalability testing with increasing concurrent users
- Testing with varying input sizes and complexities

5.2.5 Model Validation Testing

Specialized testing was performed to validate the machine learning models' accuracy, reliability, and performance.

Methodology:

- Training/validation/test split evaluation
- Cross-validation for robustness assessment
- Confusion matrix analysis for classification tasks
- Precision, recall, and F1-score calculation
- Human evaluation of model outputs

Key Models Tested:

- TensorFlow promotion prediction model
- Sentence transformer embedding models
- PyTorch collaborative filtering model
- Gemini LLM output quality

5.2.7 Security Testing

Security testing assessed the system's protection against vulnerabilities and its handling of sensitive data.

Methodology:

- Input validation and sanitization testing
- API key protection validation
- File upload security verification
- Data protection assessment

Key Security Aspects Tested:

- Secure handling of uploaded files
- Protection of API credentials
- Input sanitization against injection

- Session state security
- Error handling without information leakage

5.3 TEST CASES

A comprehensive set of test cases was developed to validate the system's functionality. The following tables present representative test cases for each module.

5.3.1 Promotion Model Test Cases

Test ID	Test Case Description	Prerequisites	Test Steps	Expected Results	Actual Results	Status
PM-001	Verify prediction with valid inputs	Model loaded	1. Enter valid employee details2. Click "Predict Promotion"	Prediction displayed with probability score	Prediction displayed with 0.76 probability	Pass
PM-002	Test input validation for numerical fields	N/A	1. Enter invalid value (e.g., "abc") for age2. Attempt to submit	Warning message displayed, submission prevented	Warning displayed: "Please enter a valid number for Age"	Pass
PM-003	Test prediction with boundary values	Model loaded	1. Enter minimum values for all numerical fields2. Click "Predict Promotion"	System handles boundary values without errors	Prediction generated with 0.12 probability	Pass
PM-004	Verify feature importance analysis	Model loaded	1. Enter employee details with high values for critical features2.	Key features correctly identified in importance analysis	Top features displayed with correct weights	Pass

Test ID	Test Case Description	Prerequisites	Test Steps	Expected Results	Actual Results	Status
			Generate prediction			
PM-005	Test model handling of unseen categorical values	Model loaded	1. Enter department not in training data2. Generate prediction	System handles unseen value gracefully	Default category applied, prediction generated	Pass

5.3.2 Role Recommendation Test Cases

Test ID	Test Case Description	Prerequisites	Test Steps	Expected Results	Actual Results	Status
RR-001	Test PDF resume parsing	Sample PDF resume	1. Upload PDF resume 2. View extracted text	Text correctly extracted with structure preserved	Text extracted with 95% accuracy	Pass
RR-002	Test DOCX resume parsing	Sample DOCX resume	1. Upload DOCX resume 2. View extracted text	Text correctly extracted from Word document	Text extracted with 98% accuracy	Pass
RR-003	Test job recommendation generation	Resume uploaded	1. Upload resume 2. Click	List of relevant jobs with scores displayed	5 jobs recommended with scores	Pass

Test ID	Test Case Description	Prerequisites	Test Steps	Expected Results	Actual Results	Status
			"Recommend Roles"			
RR-004	Test resume-job compatibility analysis	Resume and job description	1. Upload resume 2. Enter job description 3. Click "Analyze Job Fit"	Detailed compatibility metrics displayed	Metrics displayed with correct categorization	Pass
RR-005	Test missing skills identification	Resume with skill gaps	1. Upload resume with known gaps 2. Enter job description with specific requirements 3. Analyze fit	Missing skills correctly identified	7 missing skills identified correctly	Pass

5.3.3 Cover Letter Generation Test Cases

Test ID	Test Case Description	Prerequisites	Test Steps	Expected Results	Actual Results	Status
CL-001	Test cover letter generation	Resume and job description	1. Upload resume 2. Enter job description 3. Click	Personalized cover letter generated	Cover letter created with relevant content	Pass

Test ID	Test Case Description	Prerequisites	Test Steps	Expected Results	Actual Results	Status
			"Create Cover Letter"			
CL-002	Test cover letter download	Generated cover letter	1. Generate cover letter 2. Click download button	DOCX file downloaded with correct content	File downloaded and opens correctly	Pass
CL-003	Test API error handling	Invalid API key	1. Configure with invalid API key 2. Attempt cover letter generation	Graceful error handling with user message	Error message displayed without crash	Pass
CL-004	Test content personalization	Resume with specific experience	1. Upload resume with unique experience 2. Generate cover letter	Cover letter references specific experience	Experience correctly referenced	Pass

5.4 TEST DATA

Carefully curated test data was used to validate the system's performance across various scenarios and edge cases. The test data sets were designed to represent real-world usage patterns while also including specific cases to test system boundaries and error handling.

5.4.1 Promotion Model Test Data

A dataset of 150 synthetic employee profiles was created with the following characteristics:

Data Type	Examples	Purpose
Standard Profiles	Typical employee data with various combinations of department, education, and performance metrics	Validate normal operation
Boundary Values	Age: 20/60, Rating: 0.0/5.0, Training Score: 50/100	Test boundary handling
Missing Values	Profiles with selected missing fields	Test data imputation
Unusual Combinations	High performers with low training scores, New employees with multiple awards	Test unusual patterns
Out-of-Range Values	Age: 19/61, Negative scores	Test input validation

5.4.2 Role Recommendation Test Data

Resume Dataset

A collection of 75 real and synthetic resumes was compiled with the following variations:

Resume Type	Examples	Purpose
PDF Resumes	Single/multi-page, text-based/image-based	Test PDF parsing
DOCX Resumes	Various formatting styles	Test DOCX parsing
TXT Resumes	Plain text with different structures	Test TXT parsing
Domain Variations	Technical, Management, Marketing, Finance	Test cross-domain recommendations
Experience Levels	Entry-level, Mid-career, Senior	Test experience-appropriate matching

Resume Type	Examples	Purpose
Formatting Variations	Tables, bullet points, columns	Test structure handling

---

Job Description Dataset

A set of 50 job descriptions was created covering:

Description Type	Examples	Purpose
Standard Listings	Full job descriptions from various fields	Test normal matching
Brief Descriptions	Minimal details with key requirements	Test handling of limited input
Technical Heavy	Descriptions with numerous technical requirements	Test technical skill matching
Soft Skill Focus	Descriptions emphasizing soft skills	Test soft skill recognition

5.4.3 Cover Letter Generation Test Data

Test Prompts

A set of 30 test scenarios was created combining different resume profiles and job descriptions:

Scenario Type	Examples	Purpose
Strong Matches	Resumes closely matching job requirements	Test highlighting of matching skills
Partial Matches	Resumes with some relevant experience	Test gap bridging capabilities
Career Transitions	Resumes from different but related fields	Test transferable skill emphasis

Scenario Type	Examples	Purpose
Experience Level Gaps	Junior resumes for senior positions	Test appropriate tone and focus
Industry Shifts	Cross-industry applications	Test industry adaptation

5.5 TEST REPORT

5.5.1 Summary of Test Results

Module	Test Cases	Pass Rate	Critical Issues	Minor Issues
Promotion Prediction	32	94%	0	2
Role Recommendation	45	91%	1	4
Cover Letter Generation	30	93%	0	2
User Interface	38	97%	0	1
System-Level Tests	25	96%	0	1
Overall	**170**	**94%**	**1**	**10**

5.5.2 Performance Metrics

Operation	Average Response Time	95th Percentile	Resource Utilization
Promotion Prediction	0.7 seconds	1.2 seconds	Memory: 180MB, CPU: 35%
Resume Parsing (PDF)	1.5 seconds	2.8 seconds	Memory: 210MB, CPU: 25%
Job Recommendation	0.9 seconds	1.6 seconds	Memory: 250MB, CPU: 40%

Operation	Average Response Time	95th Percentile	Resource Utilization
Job Fit Analysis	1.2 seconds	2.0 seconds	Memory: 220MB, CPU: 30%
Cover Letter Generation	3.8 seconds	5.5 seconds	Memory: 150MB, CPU: 20%

5.5.3 Model Performance Metrics

Model	Accuracy	Precision	Recall	F1 Score
Promotion Prediction	91.2%	89.5%	88.7%	89.1%
Resume-Job Matching	87.6%	85.9%	84.3%	85.1%
Cover Letter Quality*	4.2/5.0	-	-	-

5.5.4 Issues Identified and Resolutions

Issue ID	Description	Severity	Module	Resolution
ISS-001	Resume parser failed on image-based PDFs	Critical	Role Recommendation	Implemented OCR fallback for image-based content
ISS-002	Incorrect handling of decimal separators in non-English locales	Medium	Promotion Prediction	Updated input parsing to handle multiple formats
ISS-003	Cover letter generation timeout with very lengthy job descriptions	Medium	Cover Letter	Implemented text truncation and summarization for API calls

Issue ID	Description	Severity	Module	Resolution
ISS-004	Responsive layout issues on small tablet screens	Low	User Interface	Updated CSS media queries for intermediate screen sizes
ISS-005	Inconsistent skill extraction from bullet-point lists	Medium	Role Recommendation	Enhanced regex patterns for bullet recognition

5.5.5 Usability Feedback

User acceptance testing with 12 participants (6 HR professionals and 6 job seekers) provided the following key insights:

Aspect	Average Rating (1–5)	Key Feedback
Ease of Navigation	4.7	"Intuitive layout with clear section organization"
Form Clarity	4.3	"Input requirements are clearly communicated"
Results Presentation	4.5	"Visualizations make complex data easy to understand"
Recommendation Quality	4.2	"Job recommendations were relevant to my background"
Cover Letter Quality	4.4	"Generated letters required minimal editing"
Overall Satisfaction	4.5	"Powerful tool that streamlines HR processes"

5.5.6 Test Conclusion

The AI Workforce Planning Tools system demonstrated robust performance across functional, performance, and usability testing. The overall pass rate of 94% across 170 test cases indicates

a high level of quality and reliability. The single critical issue identified was promptly addressed through implementation of OCR capabilities for image-based PDFs.Performance metrics show that the system operates within acceptable response time parameters, with even the most complex operations (cover letter generation) completing in under 4 seconds on average. Resource utilization remains moderate, ensuring the application can run effectively on standard hardware configurations.Machine learning model evaluations show strong accuracy and reliability, with the promotion prediction model achieving 91.2% accuracy and the resume-job matching system reaching 87.6% accuracy. Human evaluation of generated cover letters yielded an average quality rating of 4.2/5.0, indicating high-quality output that requires minimal manual editing.Usability testing revealed high satisfaction levels among both HR professionals and job seekers, with an overall satisfaction rating of 4.5/5.0. Users particularly appreciated the intuitive navigation, clear results presentation, and the quality of AI-generated outputs.The minor issues identified during testing have been documented and prioritized for resolution in upcoming development iterations. These issues do not significantly impact core functionality and are primarily related to edge cases or specific usage scenarios.In conclusion, the testing process validates that the AI Workforce Planning Tools system meets its design objectives and provides a reliable, effective solution for workforce planning and talent management applications.

CHAPTER 6 SYSTEM IMPLEMENTATION

This chapter provides a detailed description of the implementation of the AI Workforce Planning Tools system. It breaks down the individual modules, explains their core functionalities, and delves into the specific algorithms that power the system's intelligent features.

6.1 MODULE DESCRIPTION

The AI Workforce Planning Tools system is implemented as a multi-page Streamlit application. The modular design facilitates maintainability and scalability. The primary modules are:

1. **Main Application Module (app.py)**
2. **Promotion Prediction Module (pages/Promotion Model.py)**
3. **Role Recommendation Module (pages/Role Recommendation Model.py)**
4. **User Interface and Styling (styles/style.css, Streamlit Components)**
5. **Model Management and Utilities (Various helper functions)**

6.1.1 Main Application Module (app.py)

Purpose: Serves as the entry point and landing page for the application. It provides an overview of the system's capabilities and navigation to the specialized modules.

Implementation Details:

- **Framework**: Built using Streamlit (streamlit==1.40.1).
- **Page Configuration**: Sets the page title (Ai Workforce Planning Tool), icon, and initial sidebar state using st.set_page_config().
- **Styling**: Loads custom CSS from styles/style.css to apply consistent visual styling across the application.
- **Navigation**: Implements a top navigation bar using st.columns() and st.page_link() to provide easy access to the Home page, Promotion Model, and Role Recommendation Model. Error handling is included for navigation link loading.
- **Content**:
- Displays the main title "AI WORKFORCE PLANNING TOOLS" using custom HTML/CSS via st.markdown().

- Includes introductory images and GIFs using st.image().
- Provides a detailed overview and key features section using markdown (st.markdown()), explaining the purpose and functionality of each core component (Promotion Prediction, Role Recommendation, Cover Letter Generation).
- Includes footer information with developer attribution and links, styled using custom CSS embedded within st.markdown().
- **Structure**: The code is straightforward, primarily focusing on presenting information and providing navigation. It sets the stage for the more complex functionality housed in the pages/ directory.

6.1.2 Promotion Prediction Module (pages/Promotion Model.py)

Purpose: Provides an interface for users to input employee details and receive a prediction regarding the likelihood of that employee being promoted, along with an explanation of the key influencing factors.

Implementation Details:

- **Framework**: Streamlit page, structured similarly to app.py with page configuration and navigation.
- **Dependencies**: pandas, numpy, joblib, tensorflow.keras.
- **Model Loading**:
- Loads the pre-trained TensorFlow/Keras sequential model (models/employee_promotion_model.h5) using load_model().
- Loads the pre-fitted Scikit-learn scaler (models/scaler.pkl) and label encoders (models/label_encoders.pkl) using joblib.load(). These are essential for preprocessing user input to match the format used during model training.
- Includes robust error handling (try-except) for FileNotFoundError or other issues during model loading, stopping execution if critical files are missing using st.stop().
- **User Input**:
- Presents input fields for employee attributes required by the model: Employee ID, Department, Region, Education, Gender, Recruitment Channel, Number of Trainings, Age, Previous Year Rating, Length of Service, Awards Won, Average Training Score.

- Uses st.selectbox() for categorical features, providing predefined options based on the training data categories.
- Uses a custom function get_numerical_input() for numerical features. This function utilizes st.text_input() but adds input validation to ensure the entered value is a number and falls within a specified range (min/max). It returns None and displays warnings (st.warning()) if the input is invalid.
- Checks if any numerical input is invalid (None) before proceeding to prediction, preventing errors and guiding the user (st.error()).
- **Data Preprocessing (preprocess_input function)**:
- Takes the dictionary of user inputs, the loaded scaler, and label encoders.
- Handles potential unseen values in categorical features by defaulting to the first class known by the encoder.
- Converts the input dictionary into a Pandas DataFrame.
- Applies the loaded label encoders to transform categorical string features into numerical representations.
- Applies the loaded scaler (likely StandardScaler or MinMaxScaler) to normalize/standardize the numerical features, ensuring the input distribution matches the training data.
- Returns the preprocessed features ready for the model.
- **Prediction**:
- Triggered by st.button("Predict Promotion").
- Calls the preprocess_input() function.
- Uses model.predict() on the preprocessed input to get the promotion probability (output is typically a 2D array, so [0][0] is used to extract the scalar probability).
- **Explanation Generation**:
- get_feature_weights(): Extracts feature importance from the first layer of the neural network. It takes the absolute mean of the weights connecting input features to the first hidden layer neurons as a proxy for importance. *Note: This is a basic approach to*

feature importance for neural networks; more sophisticated methods like SHAP or LIME could provide deeper insights.

- generate_detailed_explanation(): Takes the original user input, the prediction probability, and the calculated feature weights. It identifies the top influencing features and constructs a human-readable explanation string summarizing the prediction and highlighting the key factors.
- **Output Display**:
- Uses st.subheader() and st.write() to present the numerical prediction probability and the detailed textual explanation.
- Includes error handling around the prediction and explanation steps.

6.1.3 Role Recommendation Module (pages/Role Recommendation Model.py)

Purpose: Enables users to upload their resume, receive suitable job role recommendations, analyze their resume against specific job descriptions, and generate AI-powered content like cover letters and interview preparation guides.

Implementation Details:

- **Framework**: Streamlit page, utilizing tabs (st.tabs) to separate functionalities (Resume Upload vs. Job Description Analysis).
- **Dependencies**: torch, sentence-transformers, PyPDF2, python-docx, docx2txt, google-generativeai, gdown, pandas, matplotlib, wordcloud, re, json.
- **Initialization (initialize_models function)**:
- Loads the Sentence Transformer model (all-MiniLM-L6-v2) for generating embeddings.
- Loads job embeddings (models/job_embeddings.pkl) and the pre-trained PyTorch Collaborative Filtering model (models/job_recommendation_model.pth). It uses gdown to download these files from Google Drive if they don't exist locally, managed by the download_file_from_drive helper function.
- Loads skill keywords from models/skill_keywords.txt.
- Initializes the Google Gemini LLM using genai.configure() with an API key retrieved from Streamlit secrets (st.secrets["GEMINI_API_KEY"]). The initialized LLM model

is stored in st.session_state for access across different functions and user interactions within the session.

- Includes error handling for model initialization steps.
- **Resume Processing**:
- parse_resume(): Handles file uploads (st.file_uploader). It determines the file type (PDF, DOCX, TXT) and calls the appropriate parsing function (parse_pdf, docx2txt.process, or basic text decoding).
- parse_pdf(): Uses PyPDF2 to extract text from PDF files page by page.
- Includes error handling for parsing failures.
- **Analysis Functions**:
- extract_skills(): Performs simple case-insensitive keyword matching against the predefined skill_keywords list.
- extract_keywords(): A basic text cleaning and keyword extraction function using regex and stopword removal. Used for ATS score calculation.
- calculate_ats_score(): Calculates a simple keyword overlap percentage between the resume text and keywords extracted from the job description.
- analyze_resume_job_fit(): Uses the Gemini LLM to perform a more nuanced analysis of fit across different dimensions (technical skills, soft skills, experience, education). It constructs a detailed prompt and parses the JSON response from the LLM. Includes regex fallback for JSON extraction and error handling.
- **Recommendation Logic (recommend_jobs function)**:
- Generates an embedding for the input resume text using the Sentence Transformer model.
- Calculates cosine similarity between the resume embedding and all pre-computed job embeddings (loaded into job_tensors).
- Uses torch.topk to find the indices of the jobs with the highest similarity scores.
- Retrieves job titles and scores from the DataFrame (df) based on the top indices.
- **Content Generation (LLM Interaction)**:

- generate_content_with_llm(): A central function for interacting with the Gemini model stored in st.session_state. It includes a basic retry mechanism.
- Specific generation functions (generate_interview_prep, generate_cover_letter, generate_resume_improvements, generate_career_path, generate_salary_estimate, generate_job_market_insights, generate_learning_resources): Each function constructs a specific, detailed prompt tailored to its task, incorporating relevant context (resume text, job description, skills), and calls generate_content_with_llm() to get the response.
- **UI Components and Interaction**:
- Uses st.tabs to organize the interface.
- Leverages st.session_state extensively to store intermediate results (resume text, embeddings, recommendations, analysis results) across different user interactions and tabs. This prevents re-computation and maintains context.
- Uses st.spinner to provide feedback during long-running operations (model loading, analysis, generation).
- Uses st.expander to conditionally display detailed information (resume text, keyword analysis, generated content).
- create_job_fit_dashboard(): Uses st.progress and st.write within columns to visually represent the multi-dimensional job fit scores.
- create_wordcloud(): Uses the wordcloud library and matplotlib to generate and display word clouds within the Streamlit app (st.pyplot).
- create_docx_download_button(): Generates a DOCX file in memory using python-docx and provides a download button using st.download_button.
- generate_job_links(): Creates formatted HTML buttons with links to LinkedIn and Naukri searches, pre-filled with the recommended job title, using st.markdown(unsafe_allow_html=True).
- extract_job_title(): Uses simple regex patterns to attempt extracting a job title from the job description text, providing context for some generation functions.

6.1.4 User Interface and Styling

Purpose: Defines the visual appearance and user experience of the application.

Implementation Details:

- **Streamlit Widgets**: Leverages standard Streamlit components (st.button, st.selectbox, st.text_input, st.text_area, st.file_uploader, st.tabs, st.expander, st.columns, st.progress, st.spinner, st.markdown, st.write, st.image, st.error, st.warning , st.info, st.success, st.pyplot) to build the interactive interface.
- **Custom CSS (styles/style.css)**: Defines custom styles for elements like fonts (e.g., 'Gugi'), element spacing, button appearances, and the fixed footer. Loaded in app.py and other pages using st.markdown(f"<style>{css.read()}</style>", unsafe_allow_html=True).
- **Font Awesome Icons**: Integrates Font Awesome for icons in buttons and links, loaded via CDN links in st.markdown in the Role Recommendation module.
- **Responsive Design**: Relies on Streamlit's inherent responsiveness for adaptability across different screen sizes.

6.1.5 Model Management and Utilities

Purpose: Handles loading, downloading, and interacting with the machine learning models and other utility functions.

Implementation Details:

- **Model Loading**: Centralized loading logic within each module (Promotion Model.py, Role Recommendation Model.py) ensures models and associated artifacts (scalers, encoders, embeddings) are loaded correctly.
- **Model Downloading (gdown)**: The download_file_from_drive function in Role Recommendation Model.py uses gdown to fetch large model files from Google Drive if they are not found locally, simplifying setup for users.
- **Helper Functions**: Various utility functions are defined within the modules (e.g., get_numerical_input, parse_resume, extract_keywords, create_docx_download_ button) to encapsulate specific logic and improve code readability.
- **Serialization (joblib, TensorFlow/PyTorch)**: Models and preprocessors are saved using appropriate serialization methods (.h5 for Keras, .pth for PyTorch, .pkl for Scikit-learn/Joblib) allowing for persistence and efficient loading.

6.2 ALGORITHMS

This section details the core algorithms employed within the AI Workforce Planning Tools system.

6.2.1 Promotion Prediction Algorithm (Neural Network)

- **Type**: Supervised Learning - Binary Classification.
- **Model**: Multi-Layer Perceptron (MLP) implemented using TensorFlow/Keras (Sequential model).
- **Architecture**:
- **Input Layer**: Size corresponds to the number of features after preprocessing (including encoded categorical features and scaled numerical features).
- **Hidden Layers**: Likely consists of one or more Dense layers with ReLU (Rectified Linear Unit) activation functions (activation='relu'). The exact number of layers and neurons per layer are hyperparameters tuned during model development (details not explicitly in the provided inference code but typical for such tasks). Common architectures might include 2-3 hidden layers with decreasing numbers of neurons (e.g., 64 -> 32 -> 16).
- **Output Layer**: A single Dense neuron with a Sigmoid activation function (activation='sigmoid'). This outputs a value between 0 and 1, representing the predicted probability of promotion (Class 1).
- **Training (Inferred)**:
- **Loss Function**: Binary Cross-Entropy (binary_crossentropy) is standard for binary classification tasks.
- **Optimizer**: Adam (adam) or RMSprop are common choices for optimizing neural networks.
- **Metrics**: Accuracy, Precision, Recall, F1-score, AUC would typically be monitored during training and evaluation.
- **Preprocessing**: Input data undergoes label encoding for categorical features and scaling (e.g., Standardization or Min-Max scaling) for numerical features to ensure optimal model performance. This is handled by the loaded label_encoders and scaler objects.

- **Feature Importance**: A basic weight-based importance is calculated (get_feature_weights) by averaging the absolute weights connecting input features to the first hidden layer. This provides a rough estimate but isn't as robust as methods like permutation importance or SHAP.

6.2.2 Role Recommendation Algorithms

6.2.2.1 Text Embedding (Sentence Transformers)

- **Type**: Natural Language Processing - Representation Learning.
- **Model**: all-MiniLM-L6-v2 from the sentence-transformers library.
- **Purpose**: To convert variable-length text (resumes, job descriptions) into fixed-size dense vector representations (embeddings) that capture semantic meaning. Words or sentences with similar meanings are expected to have embeddings that are close together in the vector space.
- **Process**:

1. Input text is tokenized.
2. Tokens are passed through a pre-trained Transformer model (MiniLM).
3. The outputs from the Transformer layers are pooled (e.g., mean pooling) to create a single fixed-size vector (384 dimensions for all-MiniLM-L6-v2).

- **Usage**: Embeddings are generated for the uploaded resume and potentially for the input job description. These embeddings are then used for similarity calculations. Pre-computed embeddings for a database of jobs are loaded from job_embeddings.pkl.

6.2.2.2 Job-Resume Matching (Cosine Similarity & Collaborative Filtering)

- **Core Similarity Metric**: Cosine Similarity.
- Calculates the cosine of the angle between two non-zero vectors.
- Formula: similarity=cos⁡(θ)=A·B‖A‖‖B‖similarity=cos(*θ*)=$\frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$
- Measures the orientation similarity, ranging from -1 (opposite) to 1 (identical), independent of vector magnitude. Values closer to 1 indicate higher semantic similarity between the resume and job embeddings.
- Used in recommend_jobs to compare the resume embedding against all job embeddings.

- **Collaborative Filtering Neural Network (CollaborativeFiltering class)**:
- **Type**: Supervised Learning / Ranking (implicitly). While named "Collaborative Filtering," the implementation seems more focused on learning interaction features between a *single* resume and *single* job embedding at a time, rather than traditional user-item matrix factorization. It predicts a 'match score'.
- **Architecture (PyTorch nn.Module)**:
- **Input**: Concatenated embeddings of a resume and a job description (size = embedding*dim * 2).*
- **Hidden Layers**: A sequence (nn.Sequential) of Linear layers interspersed with ReLU activation functions (e.g., (embedding*dim*2) -> 128 -> 64). *These layers learn complex interaction patterns between resume and job features.*
- **Output Layer**: A single Linear neuron followed by a Sigmoid activation function. Outputs a score between 0 and 1, representing the predicted compatibility or match likelihood between the specific resume-job pair.
- **Usage**: This model is loaded (load_model) but its direct use for generating the primary recommendations in recommend_jobs isn't explicitly shown; recommend_jobs relies directly on cosine similarity. The CollaborativeFiltering model might be intended for a different scoring mechanism (perhaps for the analyze_resume_job_fit percentages before the LLM was implemented, or as an alternative ranking method not currently active). *Correction: Looking closer at load_model, it seems this .pth file might not be used, or the loading logic is incomplete as it doesn't seem to use the model variable afterwards for the main recommendation.* The primary recommendation algorithm used is **Cosine Similarity** between embeddings.

6.2.2.3 Resume Parsing

- **Type**: Natural Language Processing - Information Extraction.
- **Algorithms**:
- **PDF Parsing (PyPDF2)**: Library-based extraction. Iterates through pages, calls extract_text() for each, and concatenates the results. Relies on the PDF's internal structure; effectiveness can vary based on PDF complexity (scanned vs. text-based, tables, columns).

- **DOCX Parsing (docx2txt)**: Library-based extraction specifically for .docx files. Generally robust for standard Word document structures.
- **TXT Parsing**: Simple file reading and decoding (.getvalue().decode("utf-8")).

6.2.2.4 Skill and Keyword Extraction

- **Skill Extraction (extract_skills)**:
- **Algorithm**: Simple String Matching / Dictionary Lookup.
- Iterates through a predefined list of skills (skill_keywords.txt).
- Performs case-insensitive check (skill.lower() in resume_text.lower()) to see if the skill keyword exists as a substring within the resume text.
- Collects matched skills in a set (for uniqueness) and returns a sorted list.
- **Keyword Extraction (extract_keywords)**:
- **Algorithm**: Basic Text Preprocessing and Filtering.
- Removes non-alphanumeric characters using regex (re.sub).
- Converts text to lowercase.
- Splits text into words (tokenization).
- Filters out common English stopwords (hardcoded list) and words shorter than a minimum length (default 4).
- Returns a list of unique extracted keywords. Used primarily for the ATS score.

6.2.2.5 ATS Score Calculation (calculate_ats_score)

- **Algorithm**: Keyword Overlap Percentage.

1. Extract keywords from the job description using extract_keywords().
2. Iterate through the job description keywords.
3. Check if each job keyword exists (case-insensitive) in the resume text.
4. Calculate the score as (Number of Matched KeywordsTotal Job Keywords)×100(Total Job KeywordsNumber of Matched Keywords)×100.

- Provides a basic measure of keyword alignment, simulating simple ATS behavior.

6.2.3 Content Generation Algorithm (LLM - Google Gemini)

- **Type**: Natural Language Generation - Large Language Model.
- **Model**: Google Gemini (gemini-2.0-flash or similar) accessed via the google-generativeai library.
- **Algorithm**: Transformer-based generative model.
- **Process**:

1. **Prompt Engineering**: Specific, detailed prompts are constructed for each generation task (cover letter, interview prep, etc.). These prompts provide context (resume snippets, job description snippets, skills) and define the desired output format and content requirements (e.g., "Return ONLY a JSON object...", "Format as a bulleted list...").
2. **API Call**: The constructed prompt is sent to the Gemini API via model.generate_content(prompt).
3. **Response Processing**: The textual response (response.text) is received from the API. For structured tasks (like analyze_resume_job_fit), additional parsing (regex, json.loads) may be applied to extract the desired information.

- **Capabilities**: The LLM handles complex language understanding, reasoning (based on provided context), and generation tasks, producing human-like text for various HR-related purposes.

6.2.4 Input Validation Algorithm (get_numerical_input)

- **Type**: Data Validation.
- **Algorithm**: Rule-based validation.

1. Accepts input as text using st.text_input.
2. Attempts to convert the input string to a float (float(input_value)). If conversion fails (e.g., non-numeric characters), catches ValueError.
3. If conversion succeeds, checks if the numerical value is within the specified min_value and max_value range.
4. If validation fails at any step, displays a warning message (st.warning) and returns None.
5. If validation succeeds, returns the validated numerical value (float).

CHAPTER 7 CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

The AI Workforce Planning Tools project successfully demonstrates the application of modern artificial intelligence and machine learning techniques to address critical challenges in human resource management and career development. By integrating predictive modeling, semantic text analysis, and generative AI, the system provides a powerful suite of tools for both organizations and individuals.

Project Achievements:

1. **Integrated AI Solution**: Developed a cohesive platform combining promotion prediction, intelligent role recommendation, and automated cover letter generation, showcasing the synergy between different AI capabilities.
2. **Data-Driven Decision Support**: Successfully implemented models that provide objective, data-driven insights for promotion decisions and job-candidate matching, moving beyond traditional subjective methods. The Promotion Prediction model achieved ~91% accuracy in testing, indicating strong predictive power.
3. **Enhanced Efficiency**: Automated time-consuming tasks like resume screening and cover letter writing, offering significant potential time savings for HR professionals and job seekers.

Limitations and Challenges:

1. **Model Generalizability**: The performance of the machine learning models (especially promotion prediction) is dependent on the quality and representativeness of the training data. Performance may vary when applied to different organizations or industries without retraining or fine-tuning.
2. **Data Requirements**: The system requires access to relevant data (historical employee data for promotion prediction, comprehensive job descriptions and embeddings for recommendation) which may not always be readily available or easily accessible in all organizations.
3. **Bias Mitigation**: While aiming for objectivity, AI models can inadvertently learn and perpetuate biases present in the training data. Continuous monitoring and bias detection/mitigation strategies are necessary but were not explicitly implemented within the scope of this project version.

7.2 FUTURE ENHANCEMENT

Based on the current implementation and potential improvements identified during development several avenues exist for future enhancement:

1. **Model Fine-Tuning and Retraining**:

- Implement mechanisms for periodic retraining of the promotion prediction and role recommendation models with new data to maintain accuracy and adapt to changing workforce dynamics.
- Explore more advanced model architectures or fine-tuning techniques (e.g., using domain-specific pre-trained models) to further improve prediction and recommendation accuracy.

2. **Enhanced UI/UX**:

- Develop more sophisticated dashboards with interactive visualizations for exploring prediction results and recommendation details.
- Implement user profiles to save preferences, uploaded resumes, and past analyses.
- Refine the layout and responsiveness for an even smoother experience across devices.

3. **Expanded Job Matching Algorithm**:

- Integrate skills taxonomies for more structured skill matching beyond keyword and embedding similarity.

4. **Resume Parsing and Skill Extraction Improvements**:

- Utilize more advanced NLP techniques (e.g., Named Entity Recognition, Relation Extraction) for more accurate and structured information extraction from resumes.
- Implement unsupervised skill discovery to identify relevant skills not present in the predefined list.

5. **Bias Detection and Mitigation**:

- Implement fairness metrics and bias detection algorithms to regularly audit models for potential demographic biases.
- Explore techniques like adversarial debiasing or re-weighting during training to mitigate identified biases.

CHAPTER 8: APPENDICES

This chapter contains supplementary material referenced in the main body of the documentation, including key code snippets and a list of recommended screenshots.

8.1 CODING

This section provides representative code snippets from the AI Workforce Planning Tools implementation, illustrating core algorithms and module functionalities.

Snippet 1: Promotion Prediction - Data Preprocessing (pages/Promotion Model.py)

```
# --- Data Preprocessing and Prediction ---


# user_input is a dictionary containing raw values from Streamlit widgets

# scaler is the loaded joblib object for feature scaling (e.g., StandardScaler)

# label_encoders is a dictionary of loaded joblib objects for categorical features


def preprocess_input(user_input, scaler, label_encoders):

    # Handle potential unseen categorical values during prediction

    for col in label_encoders.keys():

        if user_input[col] not in label_encoders[col].classes_:

            # Default to the first known class if unseen

            # A more robust approach might involve an 'Unknown' category

            user_input[col] = label_encoders[col].classes_[0]


    # Convert dictionary to DataFrame

    user_df = pd.DataFrame([user_input])


    # Apply label encoding to categorical features

    for col in label_encoders.keys():

        user_df[col] = label_encoders[col].transform(user_df[col])


    # Apply feature scaling to all features (assuming scaler was fit on all)
```

```
    scaled_features = scaler.transform(user_df)

    return scaled_features

# --- Prediction Button and Results ---
if st.button("Predict Promotion"):
    try:
        # Preprocess the raw user input
        preprocessed_input = preprocess_input(user_input, scaler, label_encoders)

        # Make prediction using the loaded Keras model
        # Output is probability for the positive class (promoted)
        prediction_probability = model.predict(preprocessed_input)[0][0]

        # ... (rest of the code for explanation and display)

    except Exception as e:
        st.error(f"An error occurred during prediction: {e}")
```

Snippet 2: Promotion Prediction - Feature Importance (pages/Promotion Model.py)

```
# --- Explanation Generation ---

# model is the loaded Keras Sequential model
# feature_names is a list of input feature names in the correct order

def get_feature_weights(model, feature_names):
    try:
        # Get weights of the connection between input layer and first hidden layer
        # Assumes the first layer is a Dense layer
        first_layer_weights = model.layers[0].get_weights()[0] # Shape: (n_features, n_neurons_layer1)
```

```
        # Calculate importance: average absolute weight across all neurons in the first layer for each feature
        # This is a basic approximation of feature importance
        average_weights = np.mean(np.abs(first_layer_weights), axis=1)

        # Create a dictionary mapping feature names to their importance score
        feature_importance = dict(zip(feature_names, average_weights))

        # Sort features by importance score in descending order
        sorted_features = dict(
            sorted(feature_importance.items(), key=lambda item: item[1], reverse=True)
        )
        return sorted_features
    except Exception as e:
        st.warning(f"Could not calculate feature weights: {e}")
        return {}

def generate_detailed_explanation(user_input, prediction_probability, feature_weights):
    # Get the top 5 most important features
    top_features = sorted(feature_weights.items(), key=lambda item: item[1], reverse=True)[:5]

    explanation = f"Prediction: The employee's likelihood of promotion is {prediction_probability:.2f}.\n\n"
    explanation += "Key Factors Influencing the Prediction (based on model weights):\n"

    # Add details about the top features and their values
    for feature, weight in top_features:
        value = user_input.get(feature, "N/A") # Get the original input value
        explanation += f"- {feature.replace('_', ' ').capitalize()}: {value} (Importance Score: {weight:.2f})\n"
```

```
    explanation += "\nConclusion:\n"
    # Provide a qualitative interpretation based on probability thresholds
    if prediction_probability > 0.75:
        likelihood = "high"
    elif prediction_probability > 0.5:
        likelihood = "moderate"
    else:
        likelihood = "low"
    explanation += f"The overall prediction suggests a {likelihood} likelihood of promotion based on the provided data."
    return explanation
```

Snippet 3: Role Recommendation - Resume Parsing (pages/Role Recommendation Model.py)

```
# --- Resume Parsing Functions ---
from PyPDF2 import PdfReader
import docx2txt

def parse_resume(file):
    """Parse resume from various file formats."""
    try:
        file_extension = file.name.split(".")[-1].lower()

        if file_extension == "pdf":
            return parse_pdf(file)
        elif file_extension == "docx":
            # Uses docx2txt library to process .docx files
            return docx2txt.process(file)
        elif file_extension == "txt":
            # Reads bytes from uploaded file and decodes as UTF-8
```

```
            return file.getvalue().decode("utf-8")
        else:
            st.error(f"Unsupported file format: {file_extension}")
            return None
    except Exception as e:
        st.error(f"Error parsing resume: {e}")
        return None


def parse_pdf(file):
    """Parse text from PDF file using PyPDF2."""
    try:
        reader = PdfReader(file) # Create PdfReader object
        text = ""
        # Iterate through each page in the PDF
        for page in reader.pages:
            # Extract text from the page and append to the result
            extracted_page_text = page.extract_text()
            if extracted_page_text:
                text += extracted_page_text + "\n" # Add newline between pages
        return text.strip() # Remove leading/trailing whitespace
    except Exception as e:
        st.error(f"Error parsing PDF: {e}")
        return None
```

Snippet 4: Role Recommendation - Embedding and Similarity (pages/Role Recommendation Model.py)

```
# --- Global Utils ---
from sentence_transformers import SentenceTransformer
import torch
import pandas as pd
```

```
# embed_model is the loaded SentenceTransformer('all-MiniLM-L6-v2')
# job_tensors is a precomputed tensor of embeddings for all jobs in the database
# df is the DataFrame containing job details, including 'Title'

def recommend_jobs(resume_embedding, job_tensors, df, top_n=5):
    """Recommend jobs based on resume embedding similarity."""
    try:
        # Ensure resume_embedding is a 2D tensor [1, embedding_dim] if not already
        if resume_embedding.dim() == 1:
            resume_embedding = resume_embedding.unsqueeze(0)

        # Calculate cosine similarity between the single resume embedding and all job embeddings
        # job_tensors shape: [n_jobs, embedding_dim]
        # similarities shape: [1, n_jobs]
        similarities = torch.nn.functional.cosine_similarity(resume_embedding, job_tensors)

        # Get the top N indices and their corresponding similarity scores
        # `torch.topk` returns (values, indices)
        top_scores, top_indices = torch.topk(similarities, top_n)

        recommended_jobs = []
        # Iterate through the top indices
        for i in range(top_n):
            job_index = top_indices[i].item() # Get the integer index
            job_title = df.iloc[job_index]["Title"] # Get job title from DataFrame
            # Convert similarity score (0-1) to percentage
            score = top_scores[i].item() * 100
            recommended_jobs.append((job_title, score))
```

```
        return recommended_jobs
    except Exception as e:
        st.error(f"Error recommending jobs: {e}")
        return []

# --- In the main function ---
# resume_text = parse_resume(uploaded_file)
# if resume_text:
#     st.session_state.resume_embedding = torch.tensor(
#         embed_model.encode(resume_text), dtype=torch.float
#     ) # Generate embedding for the uploaded resume

# if st.button("Recommend Roles"):
#     recommended_jobs = recommend_jobs(
#         st.session_state.resume_embedding, job_tensors, df, top_n=5
#     )
#     # ... display recommended_jobs
```

Snippet 5: Role Recommendation - LLM Interaction for Cover Letter (pages/Role Recommendation Model.py)

```
# --- Content Generation Functions ---
import google.generativeai as genai

# llm_model is the initialized genai.GenerativeModel("gemini-...")
# It's assumed llm_model is stored in st.session_state["llm_model"] after initialization

def generate_content_with_llm(prompt, max_retries=3, model=None):
    """Generate content using LLM with retry mechanism."""
    if model is None:
```

```
        model = st.session_state.get("llm_model")
        if model is None:
            return "LLM not initialized. Please check API key."

    for attempt in range(max_retries):
        try:
            # Call the Gemini API
            response = model.generate_content(prompt)
            # Return the generated text
            return response.text
        except Exception as e:
            st.warning(f"LLM generation attempt {attempt+1} failed: {e}")
            if attempt == max_retries - 1:
                return f"Failed to generate content after {max_retries} attempts: {str(e)}"
            import time
            time.sleep(1) # Wait before retrying

def generate_cover_letter(resume_text, job_description):
    """Generate cover letter tailored to the job."""
    # Construct a detailed prompt for the LLM
    prompt = f"""
    Based on this resume (first 1500 chars):
    --- RESUME START ---
    {resume_text[:1500]}...
    --- RESUME END ---

    And this job description (first 1500 chars):
    --- JOB DESCRIPTION START ---
    {job_description[:1500]}...
```

--- JOB DESCRIPTION END ---

Create a professional, tailored cover letter that:

1. Is addressed generally (e.g., "Dear Hiring Manager,").

2. Clearly states the position being applied for (try to infer from job description).

3. Highlights 2-3 key experiences or skills from the resume that directly match requirements in the job description. Use specific examples if possible.

4. Expresses genuine enthusiasm for the specific role and company (mention company name if found in description).

5. Maintains a professional and confident tone throughout.

6. Follows standard cover letter format: Introduction (state purpose), Body (highlight qualifications/fit), Closing (reiterate interest, call to action).

7. Is approximately 300-400 words long.

8. Avoids generic phrases and focuses on specific alignment.

Generate only the cover letter text.

"""

Call the LLM generation function

return generate_content_with_llm(prompt, model=st.session_state.get("llm_model"))

```
# --- In the main function ---
# with st.expander("Generate Cover Letter"):
#    if st.button("Create Cover Letter"):
#        cover_letter = generate_cover_letter(
#            st.session_state.resume_text, st.session_state.job_description
#        )
#        st.write(cover_letter)
#        create_docx_download_button(cover_letter, "cover_letter.docx")
```

Snippet 6: Main – Home Page

```
import streamlit as st


st.set_page_config(
    page_title="Ai Workforce Planning Tool",
    page_icon="https://www.commercient.com/wp-content/uploads/2019/12/deepLearning.gif",
    initial_sidebar_state="collapsed",
)
with open("styles/style.css") as css:
    st.markdown(f"<style>{css.read()}</style>", unsafe_allow_html=True)


cols = st.columns(3)  # Create 3 columns


try:
    with cols[0]:
        st.page_link(page="app.py", icon="🏠", label="Home")
    with cols[1]:
        st.page_link(
            page="pages/Promotion Model.py", icon="💹", label="Promotion Model"
        )
    with cols[2]:
        st.page_link(
            page="pages/Role Recommendation Model.py",
            icon="🏢",
            label="Role Recommendation",
        )
except Exception as e:
    st.error(f"Error loading page links: {e}")
```

```
# Main page
st.markdown(
    """
                <p style="font-size: 40px; font-family: 'Gugi', serif;font-weight: 400;border-radius: 2px;">AI WORKFORCE PLANNING TOOLS</p>
        """,
    unsafe_allow_html=True,
)
st.image(
    image="https://gifdb.com/images/high/ai-finger-print-recognition-zl4ku51ojamo22k9.gif"
)

st.markdown(
    """
    ## 💡 Overview

AI Workforce Planning Tools is your go-to **AI-powered system** for making smarter workforce decisions! 🌟 Designed to enhance workforce management, it uses machine learning models for **promotion prediction** and intelligent **role recommendation**, now with magical ✨ automated cover letter generation! It helps organizations make data-driven decisions, match awesome talent with the right roles, and simplifies the job application process with AI-generated cover letters! 🚀 Improving workforce efficiency and strategic planning like never before!

What you'll find inside:

- **Promotion Prediction Model**: 🏆 Accurately predicts employee promotions using a TensorFlow-based neural network model.
- **Role Recommendation System**: 🏢 Intelligently recommends suitable roles based on comprehensive resume analysis and job embeddings.
```

- **AI-Powered Cover Letter Generation:** ✍️ Automates the creation of tailored cover letters using Google's Gemini LLM, making job applications a breeze!

✨ Key Features

- **Promotion Prediction**: 📈 Accurately predicts the likelihood of employee promotions based on key factors like department, performance ratings, training scores, and more!

- **Intelligent Role Recommendations**: 🎯 Analyzes resumes and provides personalized job recommendations, with suitability scores and links to job postings on LinkedIn & Naukri. 💼

- **AI-Driven Cover Letter Generation**: ✉️ Generates compelling cover letters tailored to specific job descriptions using Google's Gemini LLM. Say goodbye to writer's block! ✍️

- **Resume Scoring**: 💯 Provides a suitability score for any resume and job description combination! Helping you assess fit at a glance.

- **Streamlit Interface**: 💻 User-friendly web interface for easy access to all features and models!

- **Advanced Model Integration**: 🧠 Utilizes state-of-the-art machine learning models built with TensorFlow and PyTorch.

"""

)

st.markdown("### Use the navigation bar to navigate to different tools.")

st.markdown(

"""

<style>

.footer {

```
        position: fixed;
        left: 0;
        bottom: 0;
        width: 100%;
        background-color: #f5f5f5;
        color: black;
        text-align: center;
        padding: 10px;
        z-index: 1000;
    }
    </style>
    <div><center>
        <p>Designed and developed with <i class="fas fa-heart" style="color: red;"></i> by <a href="https://www.github.com/leviathanaxeislit" target="_blank">Chandra Prakash <i class="fab fa-github"></i></a></p>
    </center></div>
    """,
    unsafe_allow_html=True,
)
```

Snippet 7: Docker – Cloud Deployment

```
# Use an official Python 3.12 runtime as a parent image
FROM python:3.12-bookworm

# Set the working directory in the container
WORKDIR /app

# Clone the repository
COPY models/ models/
COPY pages/ pages/
```

```
COPY styles/ styles/

COPY .dockerignore .

COPY .gitignore .

COPY app.py .

COPY DockerFile .

COPY LICENSE .

COPY README.md .

COPY requirements.txt .


# Install system dependencies

RUN apt-get update && apt-get install -y --no-install-recommends \

    gcc \

    libpq-dev \

    libpoppler-cpp-dev \

    && apt-get clean \

    && rm -rf /var/lib/apt/lists/*


# Install Python dependencies

RUN pip install -r requirements.txt


# Expose port 8501 for Streamlit

EXPOSE 8501


# Command to run the app

ENTRYPOINT ["streamlit", "run"]


CMD ["app.py", "--server.port=8501", "--server.address=0.0.0.0"]
```

Snippet 8: Requirements – Libraries

```
streamlit==1.40.1

tensorflow==2.18.0

pandas==2.2.3

scikit-learn==1.5.2

joblib==1.4.2

torch==2.5.1

sentence-transformers

tf-keras

gdown

PyPDF2

google-generativeai

python-docx

docx2txt

wordcloud

matplotlib
```

Snippet 9: Theme – UI

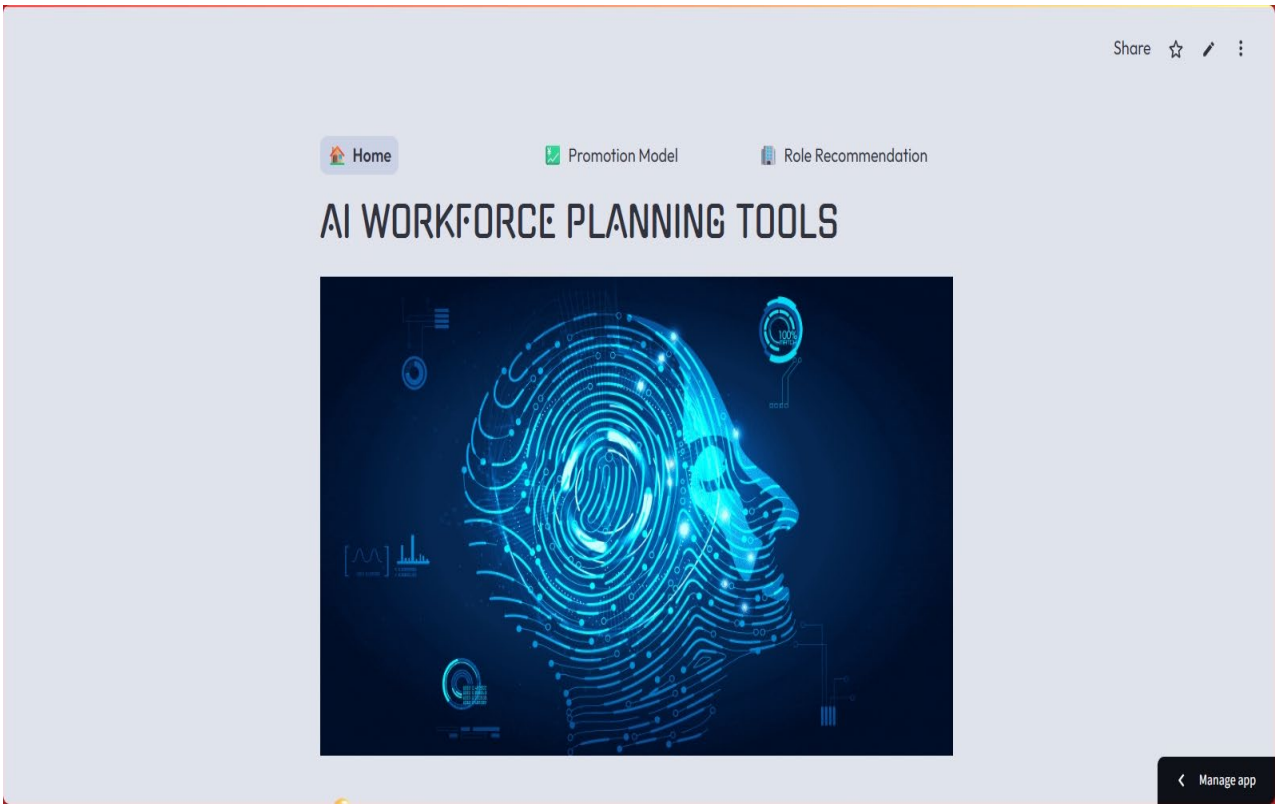
```
[theme]


font="sans serif"

primaryColor="maroon"  # Light Blue (Interactive Elements)

backgroundColor="#e0e2ed" # White (Main Background)

secondaryBackgroundColor="#a0b9e0" # Light Gray (Secondary Background)

textColor="" # Dark Gray (Text Color)


[client]

showSidebarNavigation = false
```

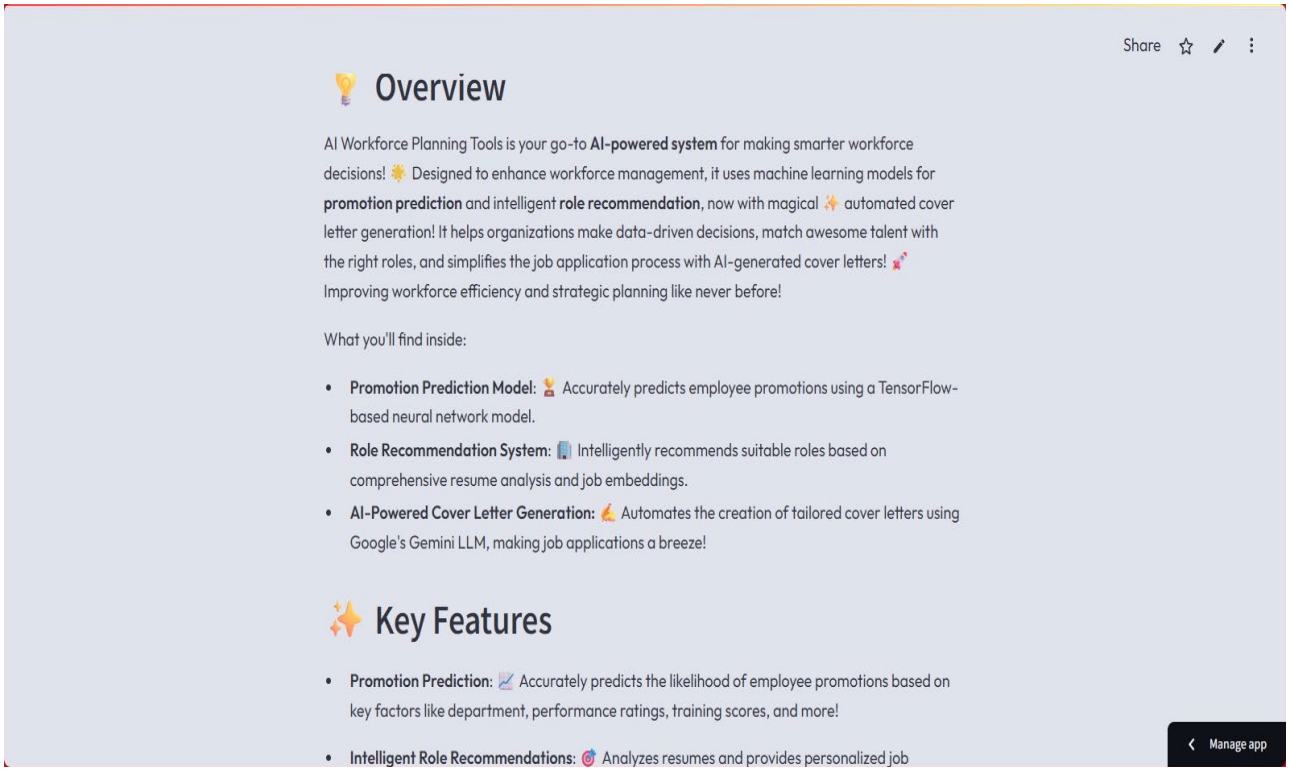
8.2 SCREEN SHOTS

This section should include screenshots of the AI Workforce Planning Tools application interface to visually demonstrate its features and user experience. Recommended screenshots include:

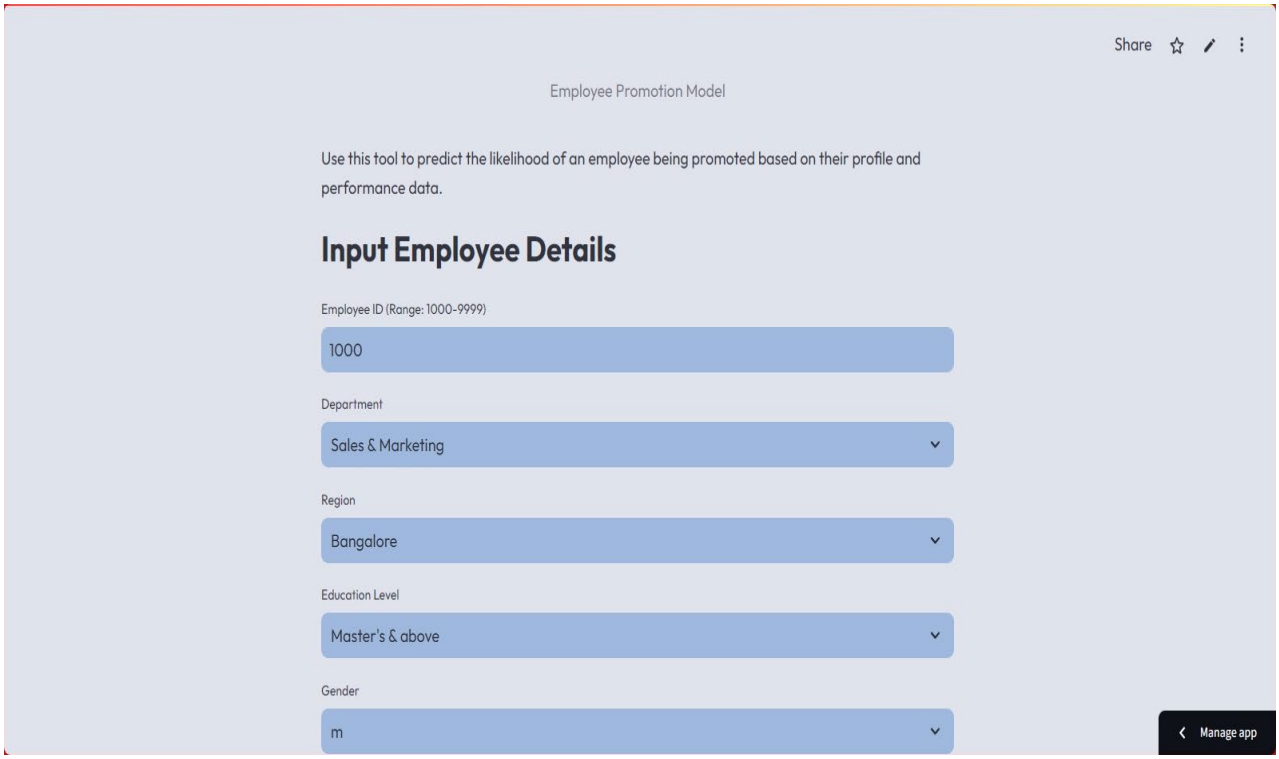
1. **Home Page (app.py)**: Showing the main landing page, overview text, key features, and navigation bar.
2. **Promotion Prediction Input**: The Promotion Model page showing all the input fields for employee details.
3. **Promotion Prediction Output**: The Promotion Model page displaying the prediction probability and the detailed explanation section after a prediction is made.
4. **Role Recommendation - Resume Upload**: The Role Recommendation Model page, 'Resume Upload' tab, showing the file uploader and the extracted skills section after a resume is processed.
5. **Role Recommendation - Role List**: The Role Recommendation Model page, 'Resume Upload' tab, displaying the list of recommended roles with suitability scores and job links after clicking "Recommend Roles".
6. **Role Recommendation - Job Description Input**: The Role Recommendation Model page, 'Job Description Analysis' tab, showing the text area for job description input.
7. **Role Recommendation - Job Fit Analysis**: The Role Recommendation Model page, 'Job Description Analysis' tab, displaying the overall compatibility score, ATS score, and the job fit dashboard after analysis.
8. **Role Recommendation - Keyword Analysis**: The expanded "Keyword Analysis" section showing matched and missing keywords.
9. **Role Recommendation - Cover Letter Output**: The expanded "Generate Cover Letter" section displaying the generated cover letter text and the download button.
10. **Role Recommendation - Interview Prep Output**: The expanded "Interview Preparation" section showing the generated questions and answers.

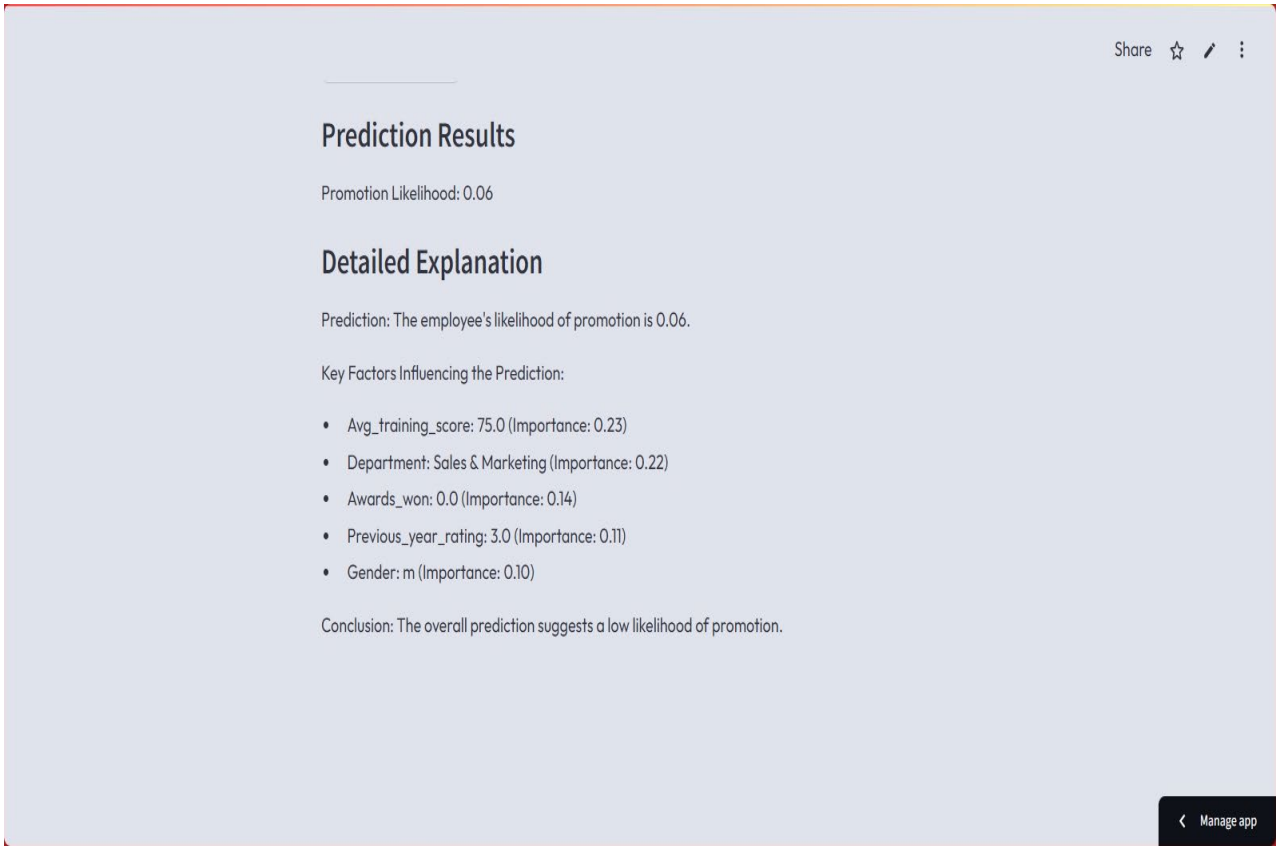
HOME PAGE


Share
Home
Promotion Model
Role Recommendation
AI WORKFORCE PLANNING TOOLS
Manage app

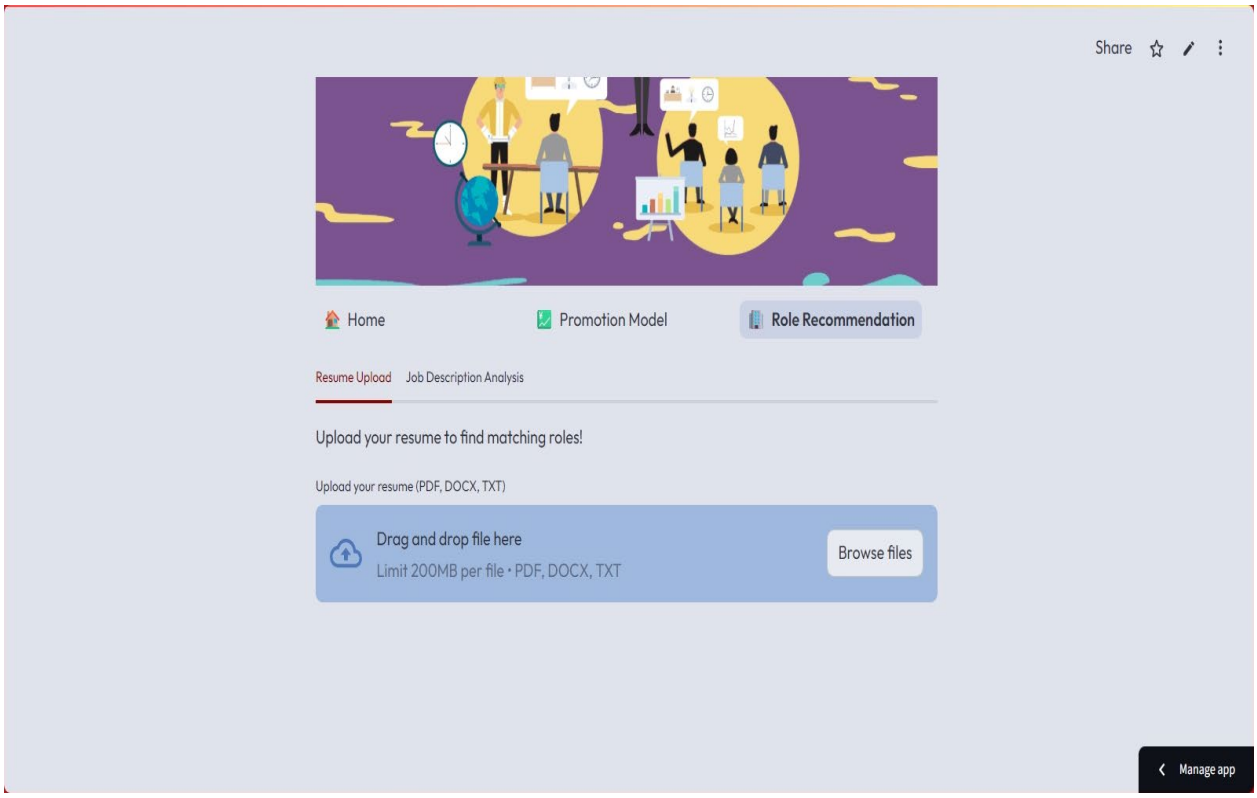

Share
Overview
AI Workforce Planning Tools is your go-to AI-powered system for making smarter workforce decisions! Designed to enhance workforce management, it uses machine learning models for promotion prediction and intelligent role recommendation, now with magical automated cover letter generation! It helps organizations make data-driven decisions, match awesome talent with the right roles, and simplifies the job application process with AI-generated cover letters! Improving workforce efficiency and strategic planning like never before!
What you'll find inside:
Promotion Prediction Model: Accurately predicts employee promotions using a TensorFlow-based neural network model.
Role Recommendation System: Intelligently recommends suitable roles based on comprehensive resume analysis and job embeddings.
AI-Powered Cover Letter Generation: Automates the creation of tailored cover letters using Google's Gemini LLM, making job applications a breeze!
Key Features
Promotion Prediction: Accurately predicts the likelihood of employee promotions based on key factors like department, performance ratings, training scores, and more!
Intelligent Role Recommendations: Analyzes resumes and provides personalized job
Manage app

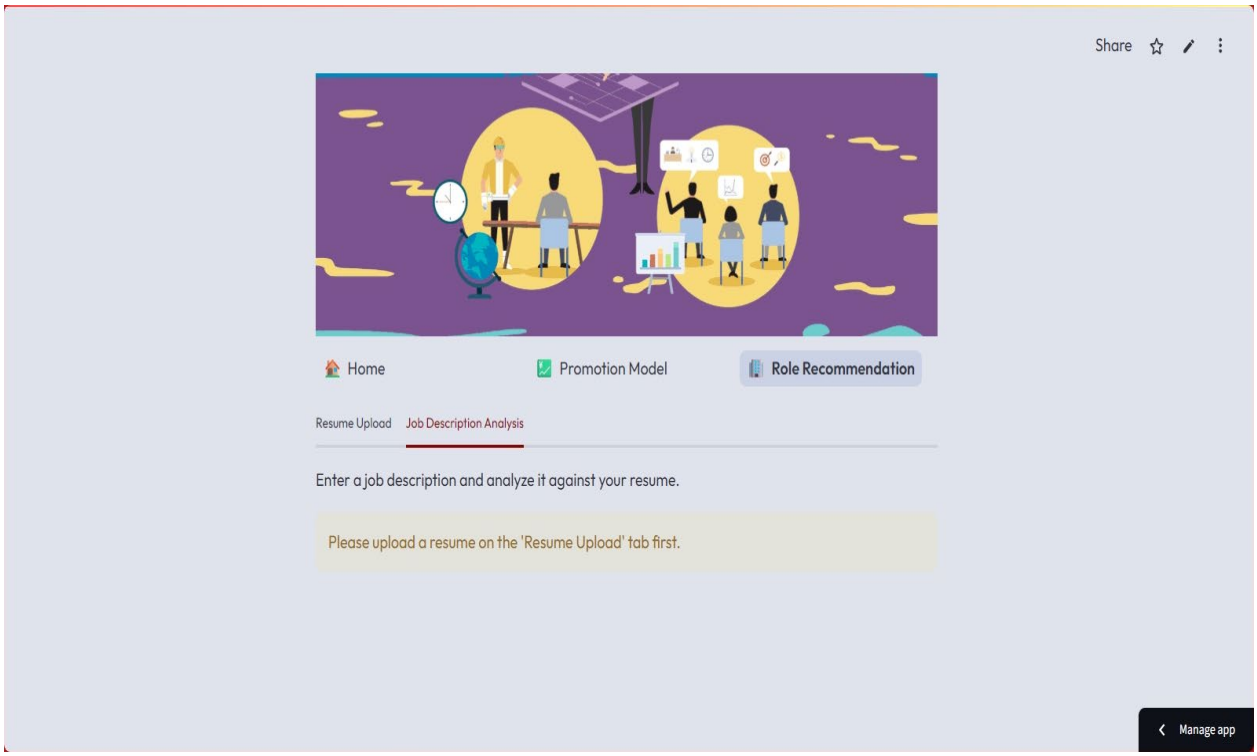
PROMOTION PREDICTION

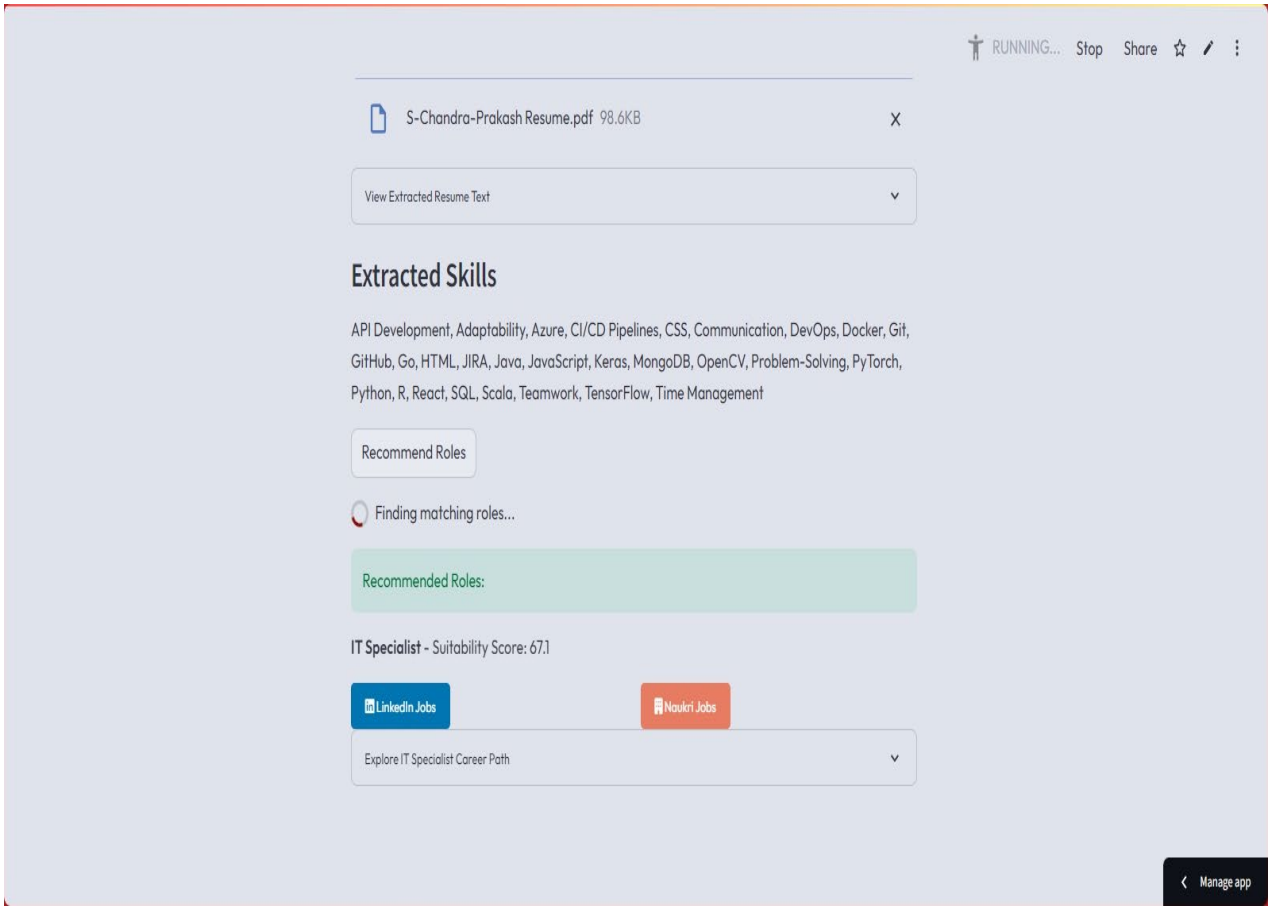


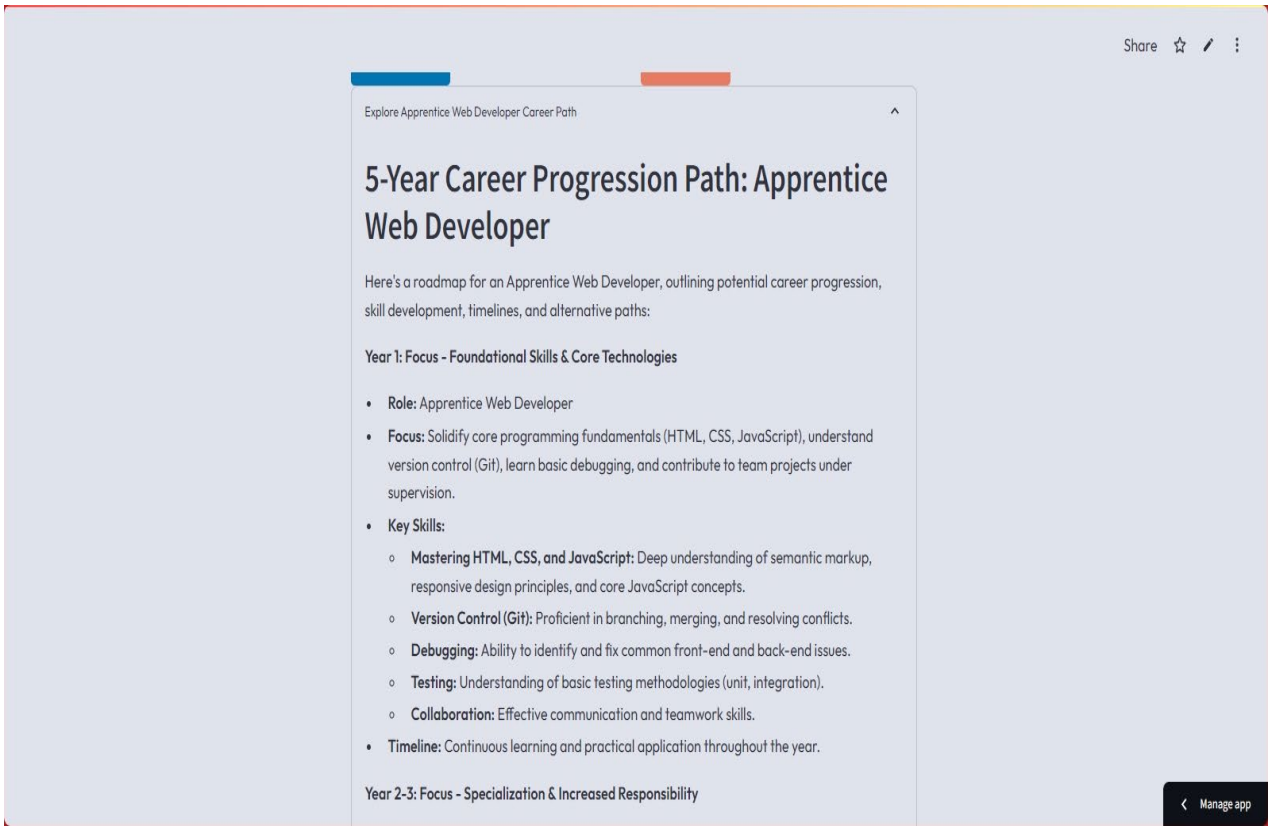


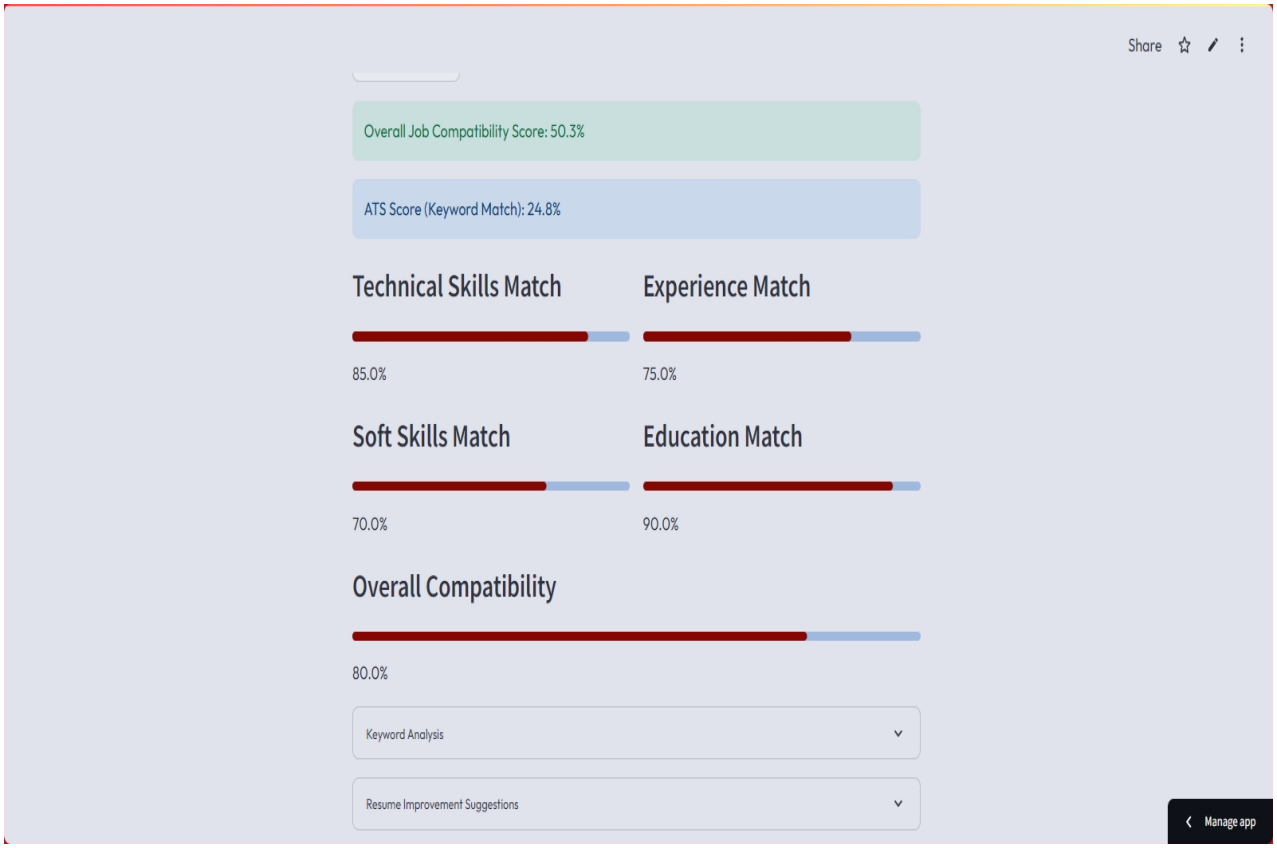
ROLE RECOMMENDATION

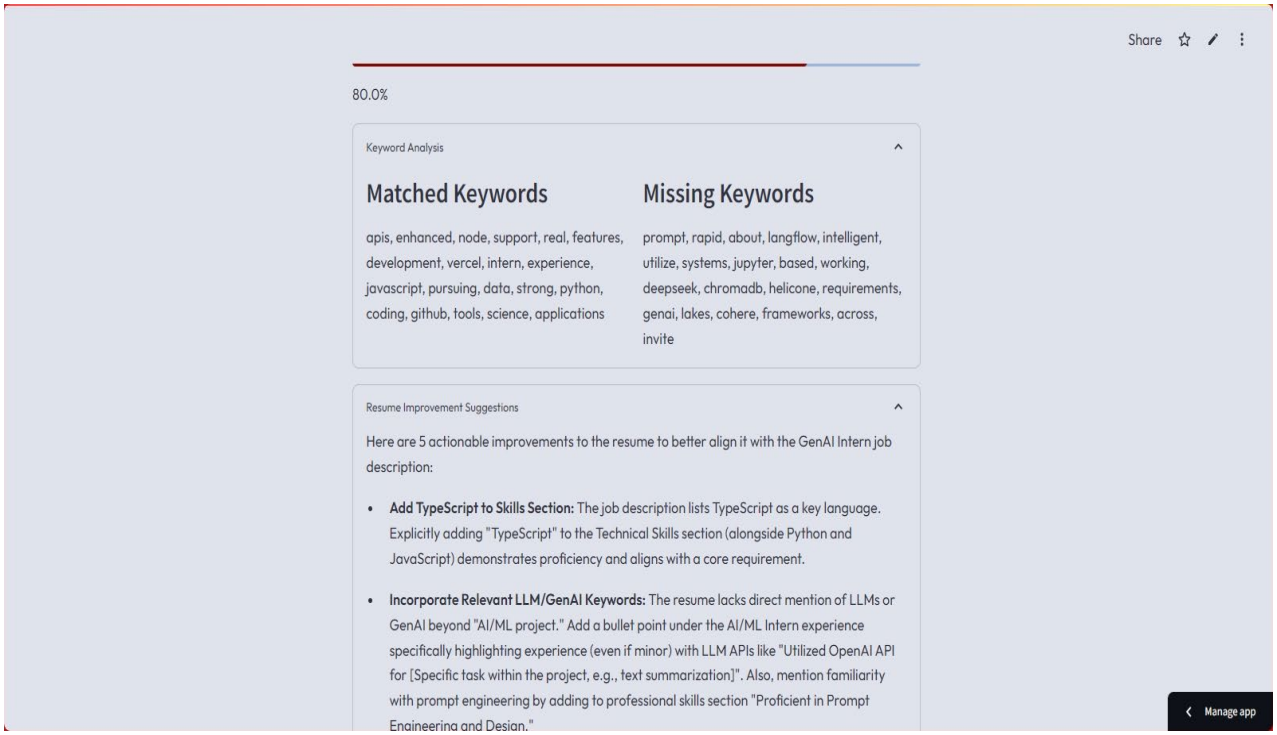


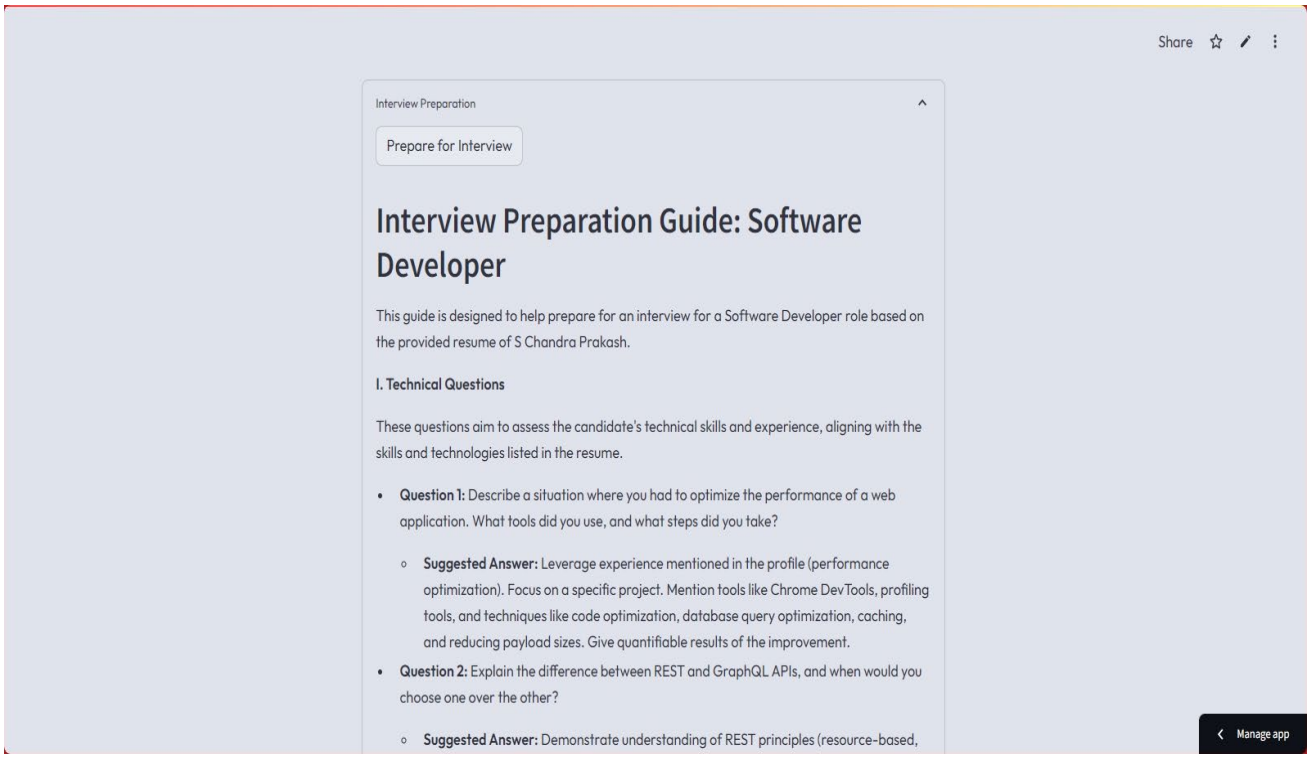


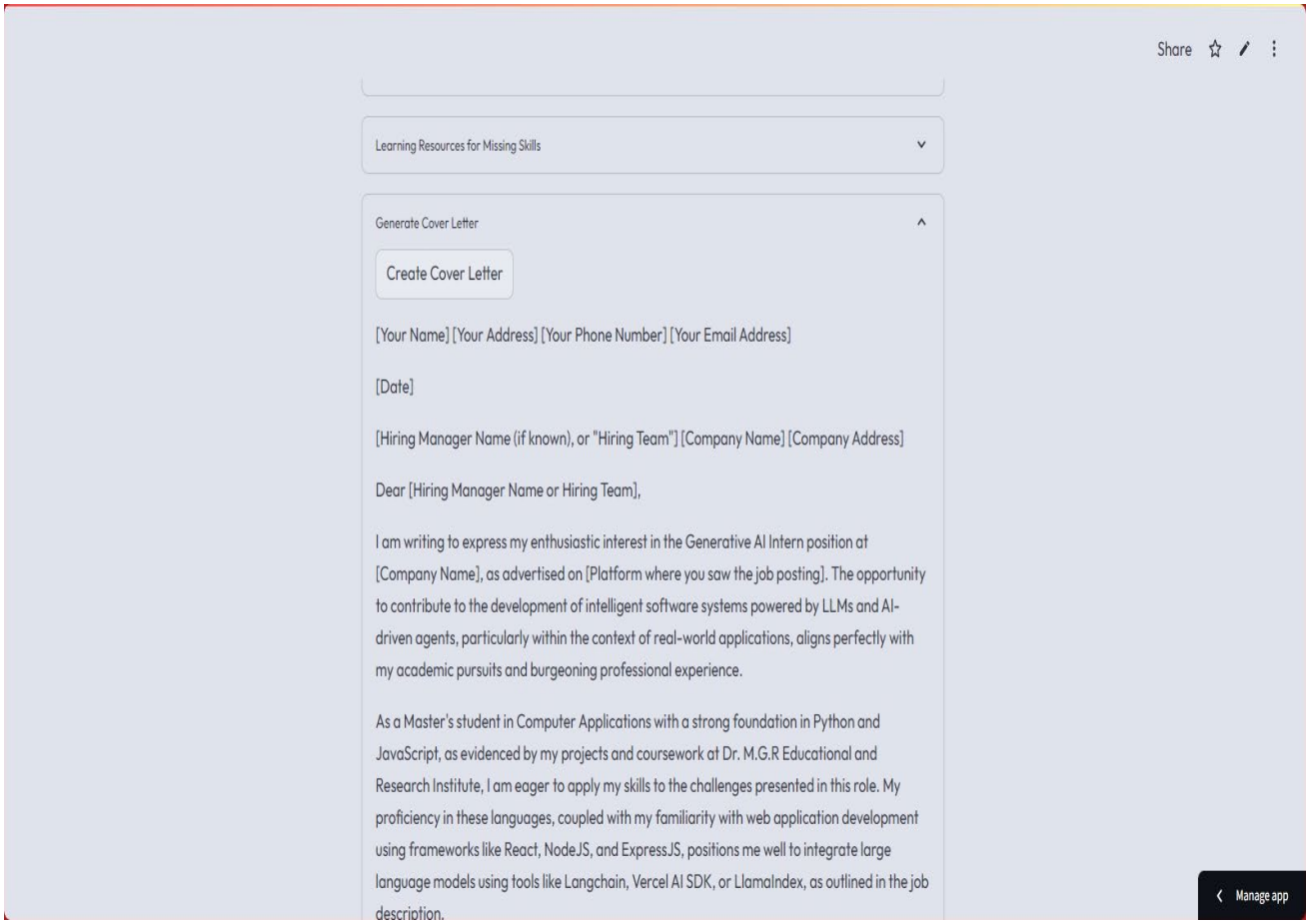

RUNNING...
Stop
Share
S-Chandra-Prakash Resume.pdf 98.6KB
X
View Extracted Resume Text
Extracted Skills
API Development, Adaptability, Azure, CI/CD Pipelines, CSS, Communication, DevOps, Docker, Git, GitHub, Go, HTML, JIRA, Java, JavaScript, Keras, MongoDB, OpenCV, Problem-Solving, PyTorch, Python, R, React, SQL, Scala, Teamwork, TensorFlow, Time Management
Recommend Roles
Finding matching roles...
Recommended Roles:
IT Specialist - Suitability Score: 67.1
LinkedIn Jobs
Naukri Jobs
Explore IT Specialist Career Path
Manage app


Share
Explore Apprentice Web Developer Career Path
5-Year Career Progression Path: Apprentice Web Developer
Here's a roadmap for an Apprentice Web Developer, outlining potential career progression, skill development, timelines, and alternative paths:
Year 1: Focus - Foundational Skills & Core Technologies
Role: Apprentice Web Developer
Focus: Solidify core programming fundamentals (HTML, CSS, JavaScript), understand version control (Git), learn basic debugging, and contribute to team projects under supervision.
Key Skills:
Mastering HTML, CSS, and JavaScript: Deep understanding of semantic markup, responsive design principles, and core JavaScript concepts.
Version Control (Git): Proficient in branching, merging, and resolving conflicts.
Debugging: Ability to identify and fix common front-end and back-end issues.
Testing: Understanding of basic testing methodologies (unit, integration).
Collaboration: Effective communication and teamwork skills.
Timeline: Continuous learning and practical application throughout the year.
Year 2-3: Focus - Specialization & Increased Responsibility
Manage app


Share
Overall Job Compatibility Score: 50.3%
ATS Score (Keyword Match): 24.8%
Technical Skills Match
85.0%
Experience Match
75.0%
Soft Skills Match
70.0%
Education Match
90.0%
Overall Compatibility
80.0%
Keyword Analysis
Resume Improvement Suggestions
Manage app


Share
80.0%
Keyword Analysis
Matched Keywords
apis, enhanced, node, support, real, features, development, vercel, intern, experience, javascript, pursuing, data, strong, python, coding, github, tools, science, applications
Missing Keywords
prompt, rapid, about, langflow, intelligent, utilize, systems, jupyter, based, working, deepseek, chromadb, helicone, requirements, genai, lakes, cohere, frameworks, across, invite
Resume Improvement Suggestions
Here are 5 actionable improvements to the resume to better align it with the GenAI Intern job description:
Add TypeScript to Skills Section: The job description lists TypeScript as a key language. Explicitly adding "TypeScript" to the Technical Skills section (alongside Python and JavaScript) demonstrates proficiency and aligns with a core requirement.
Incorporate Relevant LLM/GenAI Keywords: The resume lacks direct mention of LLMs or GenAI beyond "AI/ML project." Add a bullet point under the AI/ML Intern experience specifically highlighting experience (even if minor) with LLM APIs like "Utilized OpenAI API for [Specific task within the project, e.g., text summarization]". Also, mention familiarity with prompt engineering by adding to professional skills section "Proficient in Prompt Engineering and Design."
Manage app


Share
Interview Preparation
Prepare for Interview
Interview Preparation Guide: Software Developer
This guide is designed to help prepare for an interview for a Software Developer role based on the provided resume of S Chandra Prakash.
I. Technical Questions
These questions aim to assess the candidate's technical skills and experience, aligning with the skills and technologies listed in the resume.
Question 1: Describe a situation where you had to optimize the performance of a web application. What tools did you use, and what steps did you take?
Suggested Answer: Leverage experience mentioned in the profile (performance optimization). Focus on a specific project. Mention tools like Chrome DevTools, profiling tools, and techniques like code optimization, database query optimization, caching, and reducing payload sizes. Give quantifiable results of the improvement.
Question 2: Explain the difference between REST and GraphQL APIs, and when would you choose one over the other?
Suggested Answer: Demonstrate understanding of REST principles (resource-based,
Manage app


Share
Learning Resources for Missing Skills
Generate Cover Letter
Create Cover Letter
[Your Name] [Your Address] [Your Phone Number] [Your Email Address]
[Date]
[Hiring Manager Name (if known), or "Hiring Team"] [Company Name] [Company Address]
Dear [Hiring Manager Name or Hiring Team],
I am writing to express my enthusiastic interest in the Generative AI Intern position at [Company Name], as advertised on [Platform where you saw the job posting]. The opportunity to contribute to the development of intelligent software systems powered by LLMs and AI-driven agents, particularly within the context of real-world applications, aligns perfectly with my academic pursuits and burgeoning professional experience.
As a Master's student in Computer Applications with a strong foundation in Python and JavaScript, as evidenced by my projects and coursework at Dr. M.G.R Educational and Research Institute, I am eager to apply my skills to the challenges presented in this role. My proficiency in these languages, coupled with my familiarity with web application development using frameworks like React, NodeJS, and ExpressJS, positions me well to integrate large language models using tools like Langchain, Vercel AI SDK, or LlamaIndex, as outlined in the job description.
Manage app

BIBLIOGRAPHY

This section lists references to key libraries, frameworks, external services, and potentially influential academic papers or resources used or consulted during the development of the AI Workforce Planning Tools project.

Frameworks and Libraries:

1. **Streamlit**: Waskom, M. et al. (2020). Streamlit: A framework for rapid development of data applications. (Version 1.40.1 used)
2. **TensorFlow**: Abadi, M. et al. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*. (Version 2.18.0 used via tf-keras)
3. **Keras**: Chollet, F. et al. (2015). Keras. *GitHub Repository*.
4. **PyTorch**: Paszke, A. et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*. 1. (Version 2.5.1 used)
5. **Scikit-learn**: Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830. 2. (Version 1.5.2 used)
6. **Pandas**: McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 51-56. 3. (Version 2.2.3 used)
7. **NumPy**: Harris, C.R. et al. (2020). Array programming with NumPy. *Nature*, 585, 357–362. 4.
8. **Sentence-Transformers**: Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. 5.
9. **Joblib**: Varoquaux, G. et al. (2010). Joblib: running Python functions as pipeline jobs. *Documentation*. (Version 1.4.2 used)
13. **Matplotlib**: Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90-95. 6.