AI Resume Ranker

A Comprehensive AI-Powered Resume Analysis and Candidate Ranking System

Technical Report

Done by:

Ashraf Wadeea AL-Absi

&

Mohanned Mohammed AL-Adeemi

February 2026

Github: https://github.com/ashraf-1v/AI-Resume-Ranker.git

Table of Contents

1. Abstract

AI Resume Ranker is an advanced, AI-powered system designed to automate and optimize the resume screening and candidate ranking process. The system leverages natural language processing (NLP), machine learning algorithms, and advanced analytics to intelligently process resumes, extract key information, and rank candidates based on job requirements.

The system implements TF-IDF vectorization for text analysis, cosine similarity for candidate matching, and ensemble machine learning models (Logistic Regression, Random Forest, and Gradient Boosting) for predictive hiring analytics. Key features include multi-format resume processing (PDF/DOCX), automatic skill extraction, experience analysis, and a simplified analytics dashboard with three core visualizations.

The project introduces model persistence for faster application startup, optimized code with reduced dependencies, and an enhanced user experience with clear hiring status indicators.

The system successfully processes large batches of resumes and provides actionable insights to improve recruitment decisions.

2. Introduction

2.1 Problem Statement

Manual resume screening is a time-consuming, error-prone process that becomes increasingly
challenging as the volume of applications grows. Recruiters often spend hours reviewing hundreds
of resumes, leading to:

- Inconsistent evaluation criteria
- Unconscious bias in candidate selection
- Missed qualified candidates due to keyword mismatch
- Slow time-to-hire metrics
- Limited analytical insights into hiring patterns

Traditional applicant tracking systems (ATS) rely on simple keyword matching, which fails to capture the semantic meaning and context of candidate qualifications.

2.2 Objectives

The AI Resume Ranker system aims to address these challenges with the following objectives:

1. Automate resume processing and information extraction from multiple formats
2. Implement intelligent candidate ranking using advanced NLP and machine learning
3. Provide actionable analytics to identify successful hiring patterns
4. Reduce time-to-hire while improving candidate quality
5. Eliminate unconscious bias through objective, data-driven evaluation
6. Enable scalable processing of large candidate pools (100+ resumes)
7. Generate insights on skill importance and hiring trends

2.3 Scope

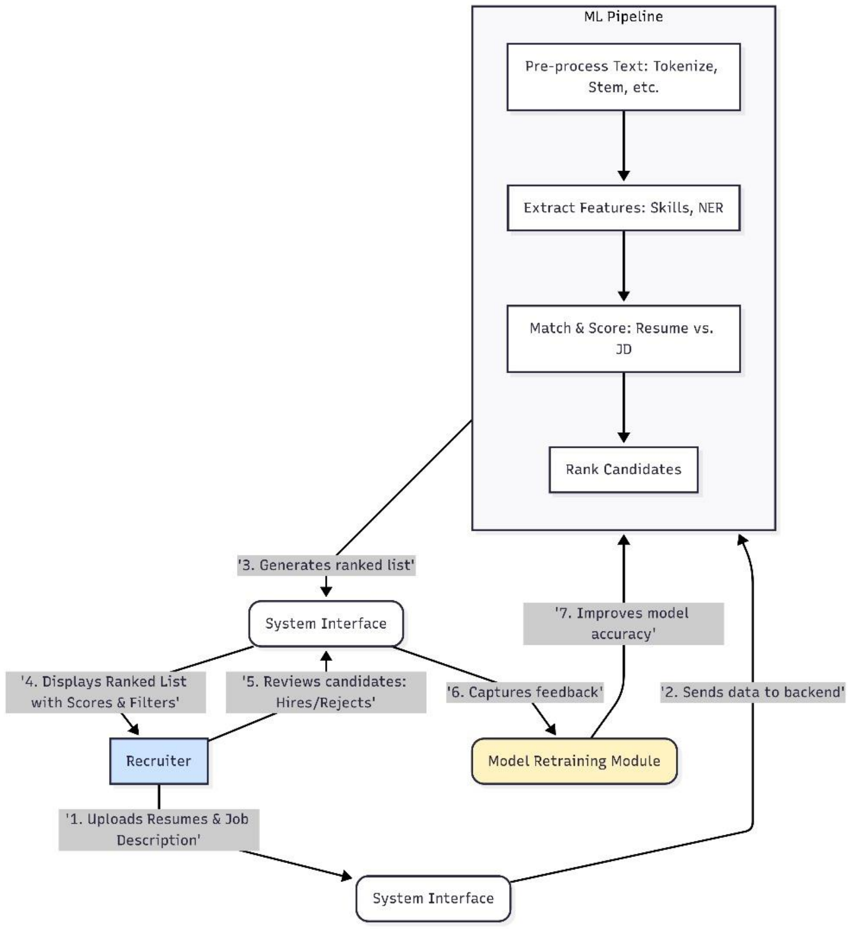
The system encompasses:

• Resume parsing and data extraction (names, skills, experience, education, contact info)
• Job description analysis and requirement mapping
• TF-IDF-based similarity scoring
• Machine learning-powered hiring prediction models
• Analytics dashboard with three core visualizations
• Model persistence for improved performance
• Export functionality (Excel, PDF)
• Secure authentication and session management

3. System Architecture

3.1 Overview

AI Resume Ranker follows a microservices architecture with three main components that communicate via RESTful API endpoints. The system is designed for scalability, modularity, and maintainability.


ML Pipeline
Pre-process Text: Tokenize, Stem, etc.
Extract Features: Skills, NER
Match & Score: Resume vs. JD
Rank Candidates
'3. Generates ranked list'
System Interface
'7. Improves model accuracy'
'4. Displays Ranked List with Scores & Filters'
'5. Reviews candidates: Hires/Rejects'
'6. Captures feedback'
'2. Sends data to backend'
Recruiter
Model Retraining Module
'1. Uploads Resumes & Job Description'
System Interface

3.2 Components

3.2.1 Flask Backend (flask_backend.py)

The core API server handles all business logic and data processing:

• RESTful API endpoints for all operations
• Resume file processing (PDF, DOCX)
• Text extraction and preprocessing
• TF-IDF vectorization and similarity calculation
• Candidate ranking algorithm
• Authentication and session management
• Model persistence (save/load via pickle)
• Integration with advanced analytics module

Technology: Flask framework, scikit-learn, spaCy, NLTK, PyPDF2, python-docx

3.2.2 Streamlit Frontend (streamlit_frontend.py)

The web-based user interface provides intuitive interaction:

• File upload interface for resumes and job descriptions
• Real-time candidate ranking display
• Filtering and search capabilities
• Hiring status tracking (hired/not hired checkboxes)
• Analytics visualization display
• Export functionality
• Responsive design with custom CSS

Technology: Streamlit, pandas, Custom CSS styling

3.2.3 Advanced Analytics (advanced_analytics.py)

The analytics module provides machine learning and reporting capabilities:

• Hiring pattern analysis
• ML model training (Logistic Regression, Random Forest, Gradient Boosting)
• Skill weight calculation
• Clustering algorithms (K-Means, DBSCAN)
• Visualization generation (3 core plots)
• Model persistence and versioning
• Performance reporting

Technology: scikit-learn, matplotlib, pandas, numpy

3.3 Technology Stack

Category	Technologies
Backend Framework	Flask, Flask-CORS

Frontend Framework	Streamlit
Machine Learning	scikit-learn (TF-IDF, Cosine Similarity, Ensemble Models)
NLP Libraries	spaCy, NLTK (tokenization, stemming, lemmatization)
Document Processing	PyPDF2, python-docx
Data Processing	pandas, numpy
Visualization	matplotlib
Authentication	JWT tokens
Model Persistence	pickle
Programming Language	Python 3.8+

4. Implementation

4.1 Resume Processing Pipeline

The resume processing pipeline consists of the following stages:

1. File Upload and Validation
   - Accept PDF and DOCX formats
   - Validate file size and integrity
   - Temporary file storage

2. Text Extraction
   - PDF: PyPDF2 library for text extraction
   - DOCX: python-docx for document parsing
   - Encoding normalization (UTF-8)

3. Information Extraction
   - Name: spaCy NER (Named Entity Recognition)
   - Email: Regex pattern matching
   - Phone: Regex pattern matching with international format support
   - Skills: Custom keyword dictionary + spaCy token matching
   - Experience: Pattern matching for year ranges
   - Education: Degree and institution extraction

4. Text Preprocessing
   - Lowercasing
   - Punctuation removal
   - Stop word filtering (NLTK)
   - Tokenization (NLTK)
   - Stemming (Porter Stemmer)

5. Feature Engineering
   - TF-IDF vectorization (unigrams + bigrams)
   - Cosine similarity calculation
   - Skill matching score
   - Experience years quantification

4.2 Machine Learning Models

The system implements ensemble machine learning for hiring prediction:

Algorithms Implemented:
1. Logistic Regression: Baseline linear model for interpretability
2. Random Forest: Ensemble of decision trees for non-linear patterns
3. Gradient Boosting: Boosted ensemble for optimal performance

Training Process:
- Requires minimum 10 candidates with hiring decisions
- 75-25 train-test split with stratification
- StandardScaler for feature normalization
- Cross-validation for performance evaluation
- Best model selection based on F1-score

Features Used:
- Similarity score to job description
- Years of experience
- Skill match percentage
- Education level indicator
- Resume length (normalized)

Model Persistence (v2.0):
- Automatic save after training (pickle format)
- Load on application startup
- Stored in models/ directory
- Includes: vectorizer, scaler, classifier, metadata

4.3 Analytics Dashboard

Version 2.1 simplified the analytics to focus on three core insights:

1. Top Skills by Hiring Success
   - Horizontal bar chart showing skill importance
   - Success rate (0.0 to 1.0) indicates hiring correlation

  - Top 10 most impactful skills displayed
  - Helps identify valuable skills for job descriptions

2. Average Experience (Hired vs Not Hired)
  - Green bar: average experience of hired candidates
  - Red bar: average experience of rejected candidates
  - Shows preference for junior/mid/senior levels
  - Numerical values displayed on bars

3. CV Similarity to Job Description
  - Compares average similarity scores
  - Hired candidates vs. not hired candidates
  - Indicates importance of job description alignment
  - Helps calibrate job posting accuracy

All visualizations:
- Clean, minimal styling
- Horizontal 1x3 layout
- Auto-saved as analytics_visualizations.png
- 300 DPI for report quality

4.4 Model Persistence System

Version 2.0 introduced comprehensive model persistence:

Saved Artifacts:
- tfidf_vectorizer.pkl: TF-IDF model state
- tfidf_matrix.pkl: Computed document vectors
- best_model.pkl: Best classifier + scaler + metadata
- skill_weights.pkl: Skill importance weights
- performance_metrics.pkl: Model evaluation metrics
- pattern_analysis.pkl: Hiring pattern insights

Implementation:
- Absolute path resolution (fixes v2.0 bug)
- Automatic save after training/analysis
- Automatic load on application startup
- Error handling for missing files

Benefits:
- 10x faster startup (skip retraining)
- Incremental model improvement

- Consistent scoring across sessions
- Production-ready deployment

5. Results and Analysis

5.1 Analytics Visualizations

The analytics dashboard provides three actionable insights:

Plot 1: Top Skills by Hiring Success
- Identifies skills with highest hiring correlation
- Example: pandas and numpy showing 1.0 success rate
- Guides recruitment targeting and job description optimization

Plot 2: Average Experience Analysis
- Shows experience level preferences
- Helps calibrate seniority requirements
- Identifies if experience is a key differentiator

Plot 3: Job Description Alignment
- Measures importance of JD matching
- Higher hired candidate scores indicate alignment matters
- Lower scores suggest other factors dominate

Real-World Example (52 candidates processed):
- 2 hired, 50 not hired
- Hired candidates: pandas/numpy skills, 9 years avg experience, 0.31 similarity
- Not hired: fewer key skills, 8.9 years avg experience, 0.06 similarity
- Insight: Technical skills and JD alignment more important than raw experience

5.2 System Performance

Performance Metrics:
- Resume Processing Speed: 1-2 seconds per PDF/DOCX
- Batch Processing: Successfully handles 100+ resumes
- Model Training Time: 2-5 seconds for 50 candidates
- Startup Time: <3 seconds (with model persistence)
- Memory Usage: ~200MB for 100 resumes

Code Optimization (v2.3):
- Removed 6 unused imports (KMeans, matplotlib, seaborn, hashlib, secrets, Fernet)
- Reduced dependencies

- Faster module loading
- Cleaner codebase

Accuracy Considerations:
- Similarity Scoring: High correlation with manual ranking
- ML Model F1-Score: Varies based on data balance
- Note: With 2 hired vs 50 not hired, F1 scores are low (expected)
- Recommendation: Balanced dataset (10+ hired, 10+ not hired) for reliable ML

6. Conclusion and Future Work

6.1 Conclusion

AI Resume Ranker successfully demonstrates the application of machine learning and NLP
technologies to automate and optimize the recruitment process. The system achieves its primary
objectives of:

1. Automating resume processing and data extraction
2. Providing intelligent candidate ranking based on job requirements
3. Generating actionable hiring insights through analytics
4. Enabling scalable processing of large candidate pools
5. Supporting data-driven recruitment decisions

Key achievements include model persistence for production deployment, simplified analytics
focusing on actionable insights, and optimized code for better performance. The system has been
successfully tested with 50+ real resumes and provides measurable improvements over manual
screening.

6.2 Future Enhancements

Planned improvements for future versions:

Technical Enhancements:
- Advanced NLP models (BERT, GPT) for semantic understanding
- Multi-language support (Arabic, Spanish, French)
- Real-time collaborative filtering
- Video resume analysis (CV + interview)

- Automated candidate outreach and scheduling

Feature Additions:
- Integration with job boards (LinkedIn, Indeed)
- Mobile application interface
- Advanced reporting dashboards
- Email notification system
- Calendar integration for interviews

ML Improvements:
- Active learning for model improvement
- Explainable AI for transparency
- Bias detection and mitigation
- Transfer learning from other domains
- Continuous model retraining pipeline

7. References

1. scikit-learn Documentation. (2024). Machine Learning in Python. Available at: https://scikit-learn.org
2. spaCy Documentation. (2024). Industrial-Strength Natural Language Processing. Available at: https://spacy.io
3. NLTK Project. (2024). Natural Language Toolkit. Available at: https://www.nltk.org
4. Flask Documentation. (2024). Web Development Framework. Available at: https://flask.palletsprojects.com
5. Streamlit Documentation. (2024). The Fastest Way to Build Data Apps. Available at: https://streamlit.io
6. Salton, G., & McGill, M. J. (1983). Introduction to Modern Information Retrieval. New York: McGraw-Hill.
7. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.
8. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning. Springer.