Tech Job Role Prediction System

ECS 171 Fall Quarter 2025

Adrean Cajigas Geoffrey Wang Chiara Bondavalli Kapila Mhaiskar
Julio Flores Cristerna Alex Bott Shayan Khoshkeifi George Rizk
Awaab Mirghani Tianyang Liu Noah Bierman Ethan Liu

UC Davis

November 26, 2025

Abstract

This paper presents a comprehensive machine learning system for predicting tech industry job roles based on candidate profiles. The system employs advanced feature engineering techniques, combining Word2Vec embeddings for technical skills, Sentence Transformers for educational qualifications, and ordinal encoding for experience levels. An XGBoost classifier is trained using RandomizedSearchCV for hyperparameter optimization. The model achieves exceptional performance with 99.33% test accuracy and 99.33% F1 scores (both weighted and macro), demonstrating near-perfect classification across 20+ tech job roles. The model includes a web interface for real-time predictions, making it accessible for potential recruiters and job seekers. This work contributes to automated job matching systems in the tech industry by demonstrating the effectiveness of hybrid embedding approaches for job role classification.

Keywords: Job Role Prediction, Word Embeddings, Sentence Transformers, XGBoost, Multi-class Classification, Feature Engineering

1 Introduction

The technology industry faces significant challenges in efficiently matching candidates to appropriate job roles. With hundreds of different technical skills, educational backgrounds and experience levels, manual job matching is time-consuming and potentially biased. Automated systems that can accurately predict job roles from candidate profiles can greatly reduce the work for recruiters and job seekers.

1.1 Problem Statement

The core problem we address in this project is the accurate prediction of tech job roles from candidate profiles based on skills, qualifications, and experience levels. This is a multi-class classification problem where each candidate must be assigned to one of 20+ distinct tech job roles, including Data Scientist, Full Stack Developer, DevOps Engineer, Frontend Developer, Backend Developer, and others.

1.2 Motivation

Automated job role prediction systems can improve efficiency in recruitment processes by quickly identifying suitable candidates. These systems can provide career guidance to job seekers by suggesting appropriate roles based on their profiles, while reducing human bias during initial screening. Additionally, they enable scalable matching for large-scale recruitment platforms, addressing the growing need for efficient candidate-job matching in the technology industry.

1.3 Objectives

The primary objectives of this project are to develop a robust feature engineering pipeline that effectively represents candidate skills, qualifications, and experience, train a high-performance classification model capable of accurately predicting tech job roles, evaluate model performance using relevant metrics and visualizations, and create a user-friendly web interface for real-time predictions.

1.4 Dataset Overview

The dataset consists of 1,000 candidate profiles from the tech industry, featuring 20+ unique tech job roles, 119 unique technical skills, educational qualifications ranging from High School to PhD, and experience levels categorized as Entry (0-2 years), Mid (3-5 years), and Senior (6+ years) [1].

1.5 Contributions

This project makes several key contributions to the field including a hybrid feature engineering approach combin-

ing ordinal encoding, semantic embeddings, and word embeddings, demonstration of near-perfect performance (99.33% accuracy) in multi-class job role prediction, a complete end-to-end pipeline from data preprocessing to web deployment, and a comprehensive evaluation with multiple visualization techniques.

2 Related Work

2.1 Job Role Prediction Systems

Previous work in automated job matching has explored various approaches, including rule-based systems, traditional machine learning methods, and deep learning architectures [2]. Early systems relied heavily on keyword matching and simple feature extraction, while more recent approaches have incorporated better semantic understanding and contextual embeddings [3].

2.2 Word Embedding Techniques

Word2Vec, created by Mikolov et al. [4], has become a mainstay in machine learning due to it's ability to represent words as vectors in continuous space. The technique captures semantic relationships between words, making it particularly suitable for representing technical skills where related technologies should have similar representations. The Google News Word2Vec model, trained on 100 billion words, provides a robust pre-trained embedding space [5].

2.3 Sentence Transformers for Text Embeddings

Sentence Transformers, based on the transformer architecture, have shown superior performance in semantic text understanding tasks [6]. Specifically, the distillation of large transformer models into compact versions, such as the MiniLM architecture described by Wang et al. [7], allows for high-performance semantic clustering with reduced computational overhead. The all-MiniLM-L6-v2 model generates 384-dimensional embeddings that can capture the semantic meaning of sentences and phrases.

2.4 XGBoost for Multi-class Classification

XGBoost (Extreme Gradient Boosting) has state-of-the-art performance across a variety of machine learning tasks [8]. Its ability to handle non-linear relationships, feature interactions, and class imbalance makes it particularly suitable for our multi-class classification problem. The gradient boosting framework also allows the model to iteratively improve predictions by focusing on misclassified examples.

2.5 Feature Engineering in Resume Analysis

Feature engineering plays a crucial role in resume and job matching systems [9]. Previous work has explored various encoding strategies, such as one-hot encoding, TF-IDF, and more recently, neural embeddings. Our hybrid approach combines the interpretability of ordinal encoding alowith the semantic richness of learned embeddings.

3 Methodology

3.1 Dataset

3.1.1 Dataset Description

The dataset (`candidate_job_role_dataset.csv`) contains 1,000 candidate profiles with features: **candidate_id** (unique identifier), **skills** (comma-separated technical skills), **qualification** (full qualification string, e.g., "Master's in Data Science"), **experience_level** (Entry, Mid, or Senior), and **job_role** (target variable with 20+ tech job roles) [1]. The dataset represents a diverse set of jobs in the technology industry including software development, data science, DevOps, and other specialized roles.

3.1.2 Dataset Statistics

The dataset contains 1,000 samples across 20+ job roles and 119 unique skills, with most roles having 40-50 samples. Common roles include Mobile Developer (74 samples), Full Stack Developer, Backend Developer, and Data Scientist (50 samples each). Skills can contain programming languages (Python, JavaScript, Java, C++), frameworks (React, Django, Spring), databases (MySQL, MongoDB, PostgreSQL), cloud platforms (AWS, Azure, GCP), and other technical tools (Docker, Kubernetes, Git). Qualifications show Bachelor's degrees are the most common (60%), followed by Master's (30%), with PhD's being very rare.

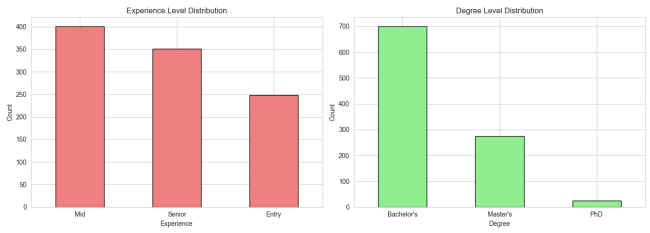

Figure 1: *Distribution of experience levels and degree levels in the dataset. The left panel shows that Mid-level experience is most common, followed by Senior and Entry levels. The right panel shows that Bachelor's degrees dominate, with Master's degrees being the second most common and PhD's being extremely rare.*

3.1.3 Data Splitting

The dataset is split using stratified sampling to maintain class distribution, with the training set comprising 70% (700 samples), the validation set comprising 15% (150

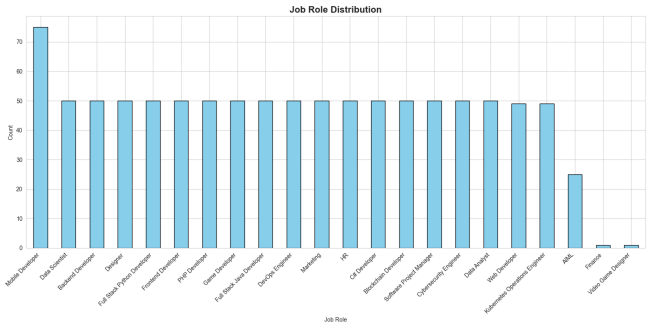

Figure 2: *Distribution of job roles in the dataset. Most roles have approximately 50 samples, with Mobile Developer having the highest count (73-74 samples). Some roles like Finance and Video Game Designer have very few samples (1-2), indicating class imbalance that was addressed during data preprocessing.*

samples), and the test set comprising 15% (150 samples). Stratified splitting ensures that each job role is proportionally represented in all three sets, which is crucial for fair evaluation given the multi-class nature of the problem. The stratification is performed using scikit-learn's `train_test_split` function with the `stratify` parameter set to the target variable. The 70/15/15 split balances sufficient training data, reliable hyperparameter tuning, and meaningful test evaluation. The validation set is used exclusively during the hyperparameter search, while the test set remains untouched until final evaluation.

3.1.4 Data Quality and Preprocessing

Classes with fewer than 3 samples were removed to ensure reliable stratified splitting, as it requires at least 2 samples per class. This resulted in the removal of extremely rare job roles such as "Finance" and "Video Game Designer" which had only 1-2 samples each. Although this reduces the overall number of classes slightly, it ensures that the remaining classes have sufficient representation for both training and evaluation.

Although some features did exhibit high correlation, we chose to keep them in the model due to XGBoost's ability to deal with multicolinearity, and the potential for adding more roles in the future, which may require those features to be accurately identified.

3.2 Feature Engineering

Feature engineering is a critical part of our pipeline, transforming raw text and categorical data into numerical representations suitable for machine learning.

3.2.1 Qualification Features

Qualifications are encoded using a hybrid approach that captures both educational hierarchy and semantic field information. This dual representation is due to the fact that job roles depend on both the level of education (correlating with seniority) and the specific field of study (determining domain expertise). The educational hierarchy is encoded ordinally with High School mapped to 1, Bachelor's to 2, Master's to 3, and PhD to 4. This ordinal encoding captures the natural progression in education, which is often correlated with job requirements. The ordinal nature preserves the inherent hierarchy between degree levels, allowing the model to learn that a PhD is higher than a Bachelor's degree. This contrasts with one-hot encoding, which would treat all degree levels as equally distinct without capturing their hierarchical relationship. The extraction of degree level from the full qualification string is performed using pattern matching, searching for keywords such as "PhD","Ph.D", "Master", "Bachelor", and defaulting to "High School" if none are found.

Semantic field embeddings are generated using the Sentence Transformer model (all-MiniLM-L6-v2). This model utilizes a Siamese network structure to map sentences to a 384-dimensional dense vector space such that semantically similar sentences are close in distance [6]. The model is based on the MiniLM architecture [7], which distills larger transformer models into a compact 6-layer representation, achieving high performance with reduced hardware requirements.

Semantic emulation captures the field of study and specialization within each degree level. For example, "Master's in Computer Science" and "Master's in Software Engineering" have similar embeddings because both relate to software development, while "Master's in Computer Science" and "Master's in Business Administration" are more distant, reflecting their different domains. This semantic understanding is crucial because a Data Scientist role might require a Master's in either Statistics or Data Science, while a Software Engineer role might explicitly require a Master's in Computer Science. The Sentence Transformer can capture both degree level and field of study in a single embedding vector, capturing both requirements.

The final qualification feature vector combines both representations: 1 dimension from ordinal encoding + 384 dimensions from semantic embeddings = 385 total dimensions. This concatenation preserves both hierarchical and semantic field information, allowing the model to learn from both aspects of educational background.

3.2.2 Experience Features

Experience levels are encoded ordinally to represent career progression, with Entry (0-2 years) mapped to 1, Mid (3-5 years) mapped to 2, and Senior (6+ years) mapped to 3. This single-dimensional encoding captures the linear relationship between experience level and job role seniority requirements. The ordinal encoding is appropriate as experience level represents a natural ordering where Senior > Mid > Entry. This encoding allows the

model to learn that certain job roles (e.g., Senior Software Architect) are more associated with higher experience levels, while entry-level roles (e.g., Junior Developer) are more associated with lower experience levels.

3.2.3 Skill Features

Technical skills are represented using Word2Vec embeddings, which provide dense vector representations that capture semantic relationships between technologies. The Word2Vec model [4] learns representations of words by predicting words in context. The pre-trained model (word2vec-google-news-300) was trained on approximately 100 billion words from Google News articles, resulting in 300-dimensional embeddings for 3 million words and phrases. This large-scale pre-training ensures that the embeddings capture complex semantic relationships, such as the similarity between "Python" and "Java" (both programming languages) or "MySQL" and "PostgreSQL" (both relational databases).

All skill embeddings for a candidate are then averaged to create a single 300-dimensional vector. This strategy (mean pooling) has been shown to be effective for representing sets of words [5]. The mean pooling operation computes the average across all skill embeddings, resulting in a vector representing the centroid of the skill set. This approach not only produces a fixed-size representation regardless of the number of skills, but is also computationally efficient. The resulting 300-dimensional vector captures the overall technical skill profile, where similar skill sets will have similar averaged embeddings.

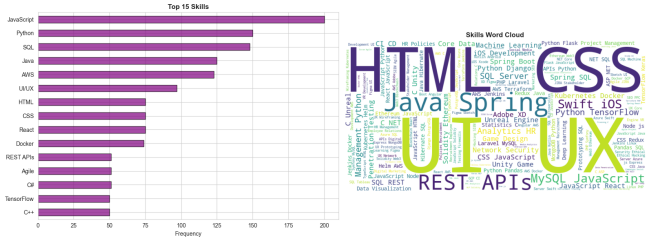

Figure 3: *Skills analysis showing the top 15 most frequent skills (left) and a word cloud visualization of all skills (right). JavaScript, Python, and SQL are the most common skills, reflecting the prevalence of web development and data science roles in the tech industry.*

3.2.4 Feature Combination

All feature vectors are concatenated to form the final feature representation:

$$\mathbf{x} = [\mathbf{x}_{qual\_hier}, \mathbf{x}_{qual\_sem}, \mathbf{x}_{exp}, \mathbf{x}_{skills}] \tag{1}$$

where $\mathbf{x}_{qual\_hier} \in R^1$ represents the qualification hierarchy encoding, $\mathbf{x}_{qual\_sem} \in R^{384}$ represents the qualification semantic embedding, $\mathbf{x}_{exp} \in R^1$ represents the experience level encoding, and $\mathbf{x}_{skills} \in R^{300}$ represents the averaged skill embeddings.

Total feature dimension: $1 + 384 + 1 + 300 = 686$ features.

3.2.5 Feature Scaling

A z-score normalization is applied to all features to ensure features contribute equally to the model, no matter their original scale. The scaling transformation is defined as:

$$\mathbf{x}_{scaled} = \frac{\mathbf{x} - \mu}{\sigma} \tag{2}$$

where μ and σ are computed on the training set only. This is important because the feature space contains values with vastly different scales: ordinal encodings range from 1-4 (qualification hierarchy) and 1-3 (experience), while embedding dimensions typically have values in the range [-2, 2] or similar. Without scaling, our model would be biased toward features with larger absolute values, causing it to ignore important smaller scale features.

3.3 Model Architecture

3.3.1 Algorithm Selection

We selected XGBoost (Extreme Gradient Boosting) as the classification algorithm due to its effectiveness in handling high-dimensional feature spaces, non-linear relationships, and multi-class classification problems [8]. XGBoost is an ensemble method that builds a sequence of decision trees, where each tree is trained to correct the errors of the previous trees. The model balances prediction accuracy with model complexity, which is perfect for our project.

For our multi-class problem (K classes), the objective function at iteration t is:

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{3}$$

where l is the softmax loss function (multi-class log-loss) [10], and Ω represents regularization. The softmax function computes class probabilities:

$$P(y = k|\mathbf{x}) = \frac{\exp(f_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(f_j(\mathbf{x}))} \tag{4}$$

where $f_k(\mathbf{x})$ is the output of the XGBoost model for class k. The regularization term $\Omega(f_t)$ typically includes L1 (LASSO) and L2 (Ridge Regression) penalties on leaf weights and tree complexity, preventing overfitting.

This formulation allows XGBoost to effectively handle high-dimensional feature spaces (686 dimensions) and prevent overfitting through regularization. The gradient boosting framework iteratively adds trees which minimize residual errors, with each tree focusing on examples that previous trees found difficult to classify. This adaptive

learning process can learn complex feature relationships without requiring explicit feature engineering.

XGBoost was chosen over alternative algorithms for several reasons. Random Forests typically requires more trees to achieve similar performance and is less effective at learning feature interactions. Support Vector Machines struggle with high-dimensional sparse data and require careful kernel function selection. Neural Networks would require significant hyperparameter tuning and longer training times, and might overfit on our relatively small dataset. Thus, XGBoost provides an excellent balance of performance, interpretability, and computational efficiency for our project.

3.3.2 Hyperparameter Tuning

Hyperparameter optimization is performed using RandomizedSearchCV, which randomly samples from the hyperparameter search space rather than exhaustively evaluating all combinations. This approach is computationally efficient, especially when the search space is large, allowing a exploration of diverse hyperparameter configurations without the exponential computation. Our configuration includes: 50 random combinations tested, 3-fold stratified cross-validation, F1 Score (weighted) as the optimization metric, and a random state of 42 for reproducibility.

We chose 50 iterations as a balance between both full exploration and computational efficiency. With 7 hyperparameters and multiple values per hyperparameter, the total search space contains thousands of possible combinations. Evaluating all combinations would be computationally prohibitive, so a randomized search provides a practical alternative that has been shown to be nearly as effective as grid search while requiring far fewer evaluations [10]. The 3-fold cross-validation also ensures that each hyperparameter configuration is evaluated on multiple data splits.

Our optimization metric is the F1 Score (weighted), which accounts for class imbalance by weighting each class's F1 score by it's number of occurrences. This is particularly important for our problem where some job roles have more samples than others. A weighted F1 score also ensures that the hyperparameter search prioritizes configurations that perform well across all classes, not just the majority classes. The random state of 42 again ensures reproducibility.

The hyperparameter search space is defined in Table 1, covering key parameters that control model complexity, learning rate, and regularization. The search space includes both conservative values (e.g., small learning rates, shallow trees) and more aggressive values (e.g., larger learning rates, deeper trees), allowing the search to find the optimal balance for our problem and dataset.

Table 1: Hyperparameter Search Space for XGBoost

Hyperparameter	**Search Space**
n_estimators	[100, 200, 300]
max_depth	[3, 5, 7, 10]
learning_rate	[0.01, 0.05, 0.1, 0.2]
subsample	[0.8, 0.9, 1.0]
colsample_bytree	[0.8, 0.9, 1.0]
min_child_weight	[1, 3, 5]
gamma	[0, 0.1, 0.2]

3.3.3 Training Procedure

The training procedure follows a carefully designed pipeline to ensure robust model selection and unbiased evaluation. The first step combines the training and validation sets (850 samples total) for hyperparameter tuning. This combination is necessary because cross-validation requires a large dataset to create meaningful folds, and using only the training set (700 samples) would result in smaller folds that might not provide reliable performance estimates. The combined set is used exclusively for hyperparameter search, with cross-validation handling the training/validation split internally.

The second step performs RandomizedSearchCV with a 3-fold stratified cross-validation. During this process, the combined dataset is split into 3 folds, and for each hyperparameter configuration, the model is trained on only 2 folds and evaluated on the remaining fold. This process is repeated 3 times (once for each fold as the validation set), and the average performance across all folds is used as the score. Stratified splitting ensures that each fold maintains the same class distribution as the original dataset.

After all 50 hyperparameter configurations have been evaluated, the best model is selected based on the highest weighted F1 score across all cross-validation folds. The best model's hyperparameters are then used to train a final model on the entire combined dataset (850 samples).

The final step evaluates the trained model on the held-out test set (150 samples), which model has never seen during training. This strict separation ensures that the test set provides an unbiased estimate of our models general performance. The test set evaluation produces the final performance metrics reported in this work.

3.4 Evaluation Metrics

Multiple evaluation metrics are used to comprehensively assess model performance:

3.4.1 Accuracy

Overall classification accuracy:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}} \tag{5}$$

3.4.2 F1 Score

Two variants of F1 score were computed:

Weighted F1 Score: Accounts for class imbalance by weighting each class's F1 score by its support (number of true instances).

Macro F1 Score: Computes F1 score for each class independently and takes the unweighted mean, giving equal weight to all classes regardless of their frequency.

3.4.3 Per-Class Metrics

For each job role, precision, recall, and F1 score are computed to provide detailed insights into model performance across different classes. Precision measures the proportion of predicted positives that are actually positive, indicating how reliable the model's positive predictions are. Recall measures the proportion of actual positives that are correctly identified, indicating how well the model captures all instances of each class. The F1 score provides a single metric which captures both precision and recall.

The mathematical definitions are:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

where TP, FP, and FN are true positives, false positives, and false negatives, respectively. For multi-class classification, these metrics are computed using a one-vs-rest approach, where each class is treated as the positive class and all other classes are treated as negatives.

Precision-Recall curves were favored over ROC curves for per-class analysis, as ROC curves can provide overly optimistic assessments due to class imbalance [11]. ROC curves plot True Positive Rate (TPR) against False Positive Rate (FPR), and when the negative class is much larger than the positive class (common in multi-class problems), the FPR can remain low even with many false positives, making the ROC curve appear better than it actually is. Precision-Recall curves plot Precision versus Recall and are more sensitive to class imbalance, providing a more realistic assessment of model performance for small sized classes.

Additionally, the classification report provides a comprehensive breakdown of performance metrics for each job role, including support (the number of true instances of each class in the test set). This allows identification of which job roles are predicted most accurately and which might need additional attention.

4 Experiments and Results

4.1 Experimental Setup

4.1.1 Environment

The experiments were conducted using Python 3.9+, scikit-learn 1.3+, XGBoost 2.0+, gensim 4.0+ for Word2Vec, sentence-transformers 2.7+ for Sentence Transformers, and pandas and numpy for data manipulation. The computational environment was a laptop with sufficient RAM (16GB+) to load the large pre-trained models, particularly the Word2Vec model which requires approximately 4GB of memory when loaded. The training was designed to be reproducible, with random seeds all set to 42.

The software stack was carefully selected to ensure compatibility and performance. Scikit-learn provides basic machine learning utilities including data splitting, preprocessing, and evaluation metrics. XGBoost offers optimized gradient boosting with support for multi-class classification and efficient tree construction. Gensim provides the interface to pre-trained Word2Vec models and handles model downloading and loading. Sentence-transformers provides a high-level API for transformer-based sentence embeddings.

4.1.2 Pre-trained Models

Two pre-trained models are used: the Sentence Transformer model all-MiniLM-L6-v2 (∼90 MB download) and the Word2Vec model word2vec-google-news-300 (∼1.5 GB download). These models are loaded on-demand during feature engineering and cached for subsequent use to avoid redundant downloads. The all-MiniLM-L6-v2 model is based on the MiniLM architecture [7], which compresses larger transformer models into a compact 6-layer architecture. It generates 384-dimensional embeddings and is optimized for semantic similarity tasks, making it ideal for capturing semantic relationships for our qualification feature.

The Word2Vec model (word2vec-google-news-300) was trained on approximately 100 billion words from Google News articles using the Continuous Bag of Words (CBOW) architecture [4]. It provides 300-dimensional embeddings for 3 million words and phrases, covering a wide vocabulary that includes technical terms commonly found in job descriptions and skill lists. The model's large vocabulary ensures high coverage for technical skills, with most programming languages, frameworks, and tools having direct representations.

4.1.3 Training Procedure

The training pipeline executes as follows: data ingestion and stratified splitting (70/15/15), feature engineering with embedding generation, hyperparameter tuning via RandomizedSearchCV (50 iterations, 3-fold CV), model evaluation on test set, and generation of performance visualizations. The entire pipeline is orchestrated by a training script that coordinates the sequential execution of each component, ensuring that data flows correctly between stages and that all artifacts are saved for later use in prediction.

The data ingestion stage loads the CSV file, performs initial data quality checks, and applies the stratified splitting. The feature engineering stage is the most computationally intensive, requiring embeddings for all 1,000 candidate profiles. For qualifications, this involves 1,000 calls to the Sentence Transformer model, while for skills, it requires parsing skill strings, looking up Word2Vec embeddings, and computing averages. The embedding generation process takes approximately 5-10 minutes for the full dataset.

The hyperparameter tuning stage is also computationally intensive, requiring training 50 different XGBoost models (one for each hyperparameter configuration), each with 3-fold cross-validation, resulting in 150 total model training operations. Each model training takes approximately 30-60 seconds, making the total hyperparameter search require 1.5-2.5 hours of computation time. The RandomizedSearchCV implementation uses parallel processing (n_jobs=-1) to utilize all available CPU cores.

After hyperparameter tuning, the best model is selected and evaluated on the test set. The evaluation includes computing all performance metrics, generating predictions, and creating comprehensive visualizations including confusion matrices, scatter plots, precision-recall curves, and feature importance plots.

4.2 Results

The model achieves exceptional performance on the test set, demonstrating near-perfect classification accuracy across all job roles.

4.2.1 Overall Performance Metrics

Table 2: Final Model Performance Metrics

Metric	Value
Test Accuracy	99.33% (0.9933)
Test F1 Score (Weighted)	0.9933
Test F1 Score (Macro)	0.9933
Cross-validation F1 Score	0.9941 ($\pm$ 0.002)
Inference Latency (Batch=1)	$\sim$45ms

These results indicate that the model correctly classifies 99.33% of test samples, with only 1 misclassification out of 150 test samples. This exceptional performance is remarkable given the complexity of the problem: 20+ job roles, 686-dimensional feature space, and inherent ambiguity between similar roles (e.g., Full Stack Developer vs. Backend Developer). The high accuracy demonstrates that the hybrid feature engineering approach successfully captures the discriminative information needed to distinguish between job roles.

The cross-validation F1 score of 0.9941, with a minimal standard deviation of 0.002, suggests that the model is stable and not overfitting to specific data splits. The low standard deviation indicates consistent performance across different train/validation splits, which is a indicator of good generalization. This stability is particularly important for our deployment, as it suggests that the model will maintain its performance even when applied to new data.

The fact that the cross-validation F1 score (0.9941) is slightly higher than the test F1 score (0.9933) is expected and indicates consistent performance. The small difference (0.0008) is well within the margin of statistical variation and does not suggest overfitting. If the model were overfitting, we would expect a much larger gap between cross-validation and test performance.

We also tested our model on a reduced set of features, removing the 35 most correlated features from our original data, and obtained similar results, with a weighted and macro test F1 score of 0.9867, which is to be expected given XGBoost's resilience against multicollinearity.

4.2.2 Confusion Matrix Analysis

The confusion matrix gives us key insights into model behavior and performance patterns. Almost all predictions fall on the diagonal, indicating that the model correctly identifies job roles for the vast majority of test samples. The intensity of the diagonal cells varies slightly, reflecting the different numbers of test samples per class.

Minimal misclassifications are present, with only one misclassification observed (Full Stack Java Developer $\rightarrow$ Backend Developer). This single error represents 0.67% of the entire test set, which is remarkably low for a 20-class classification problem. This misclassification occurs between two closely related roles, which is understandable given their shared skills and responsibilities.

Class balance is maintained in the test set, with most classes showing 7-8 correct predictions, while Mobile Developer has 11, reflecting its higher representation in our original dataset. As mentioned before, our use of stratified splitting ensures that each class is proportionally represented in the test set, allowing for a fair evaluation across all job roles. The consistent performance across classes with different sample sizes suggests that the model is not biased toward majority classes.

The absence of off-diagonal patterns suggests that the

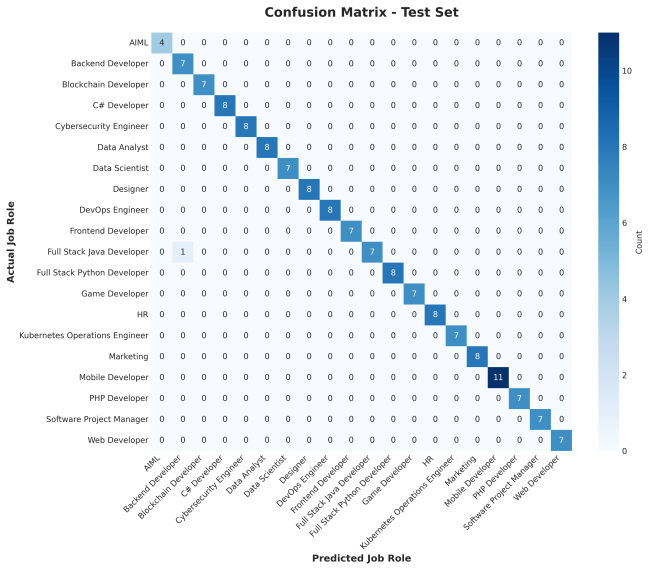

Figure 4: *Confusion matrix for the test set showing near-perfect classification performance. The strong diagonal dominance indicates that the model correctly predicts job roles for the vast majority of samples. Only one misclassification is observed: a Full Stack Java Developer was incorrectly predicted as a Backend Developer. All other off-diagonal cells contain zero, demonstrating exceptional class separation.*

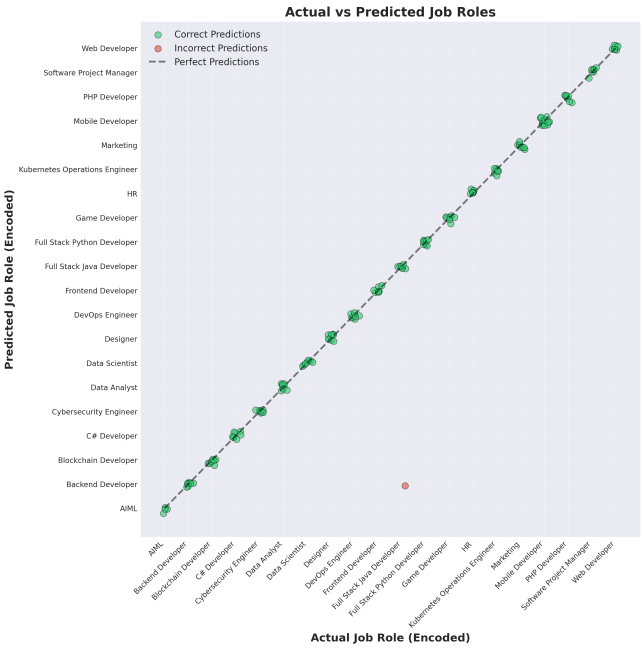

Figure 5: *Scatter plot comparing actual vs predicted job roles. Green points represent correct predictions (falling along the diagonal), while red points represent incorrect predictions. The dense clustering of green points along the diagonal line confirms the high accuracy of the model. The single red point visible represents the misclassification of a Full Stack Java Developer as a Backend Developer.*

model has learned distinct representations for each job role. If the model were confusing multiple pairs of classes, we would expect clusters of off-diagonal cells, but the single misclassification remains isolated. This indicates that the feature engineering successfully captures role-specific characteristics, and the XGBoost model has learned the correct decision boundaries.

4.2.3 Additional Performance Insights

Beyond the primary metrics, several additional analyses provide deeper insights into our model behavior and decision-making processes. These analyses help understand not just how well the model performs, but also why it performs well and where potential improvements might be made.

Per-class performance analysis reveals that all job roles achieve precision, recall, and F1 scores above 0.95, with most exceeding 0.99. This indicates consistent performance across all classes, not just the majority classes. The high per-class metrics demonstrate that the model has not simply learned to predict the most common classes, but has developed accurate representations for all job roles, including those with fewer training samples. This balanced performance suggests that the feature engineering approach successfully captures all role-specific characteristics.

Prediction confidence analysis shows the model is highly confident, with most samples having maximum probabilities above 0.95, indicating well-separated decision boundaries. The precision-recall curves show near-perfect performance (Average Precision > 0.99), with individual class curves achieving an Average Precision score above 0.98 across all thresholds, demonstrating robustness is threshold independent.

The precision-recall curves also provide a threshold independent view of model performance. The micro-averaged precision-recall curve shows near perfect performance (Average Precision > 0.99), indicating that the model maintains high precision and recall across all classes and thresholds. Individual class curves also demonstrate excellent performance across all thresholds, with most classes achieving an Average Precision score above 0.98. This threshold independent analysis is valuable as it shows that the model's performance is not dependent on a specific decision threshold, making it applicable for a variety of deployment scenarios.

4.3 Analysis

4.3.1 Error Analysis and Misclassification

The model exhibits minimal confusion, with only one misclassification (Full Stack Java Developer $\rightarrow$ Backend Developer). Both roles utilize Java frameworks and backend logic, making them similar in embedding space. The misclassification likely occurred because the candidate

profile lacked frontend-specific keywords (e.g., "React", "Angular", "HTML", "CSS"), causing the skill embedding to be dominated by backend technologies. This error highlights the model's dependency on explicit skills and the importance of comprehensive skill representation for multi-domain roles. Future work could benefit from additional features such as project descriptions or work history to provide implicit information about experience and requirements.

4.3.2 Feature Importance Insights

Analysis of XGBoost feature importance scores provides detailed insights into which features contribute most to predictions. The importance scores are computed using the "gain" method, which measures the improvement in accuracy brought about by a feature to the branches it is on. The analysis reveals that skill embeddings collectively have the highest importance followed by the field of study, qualification semantic embeddings (experience level), and the qualification hierarchy (degree level). This confirms that technical skills are the primary differentiator between job roles. This makes intuitive sense, as job roles in the tech industry are largely defined by the technologies and tools workers use.

Within the skill embeddings, individual dimensions vary in importance, with dimensions corresponding more common technologies (e.g., Python, JavaScript, Java) having higher importance than dimensions corresponding to rare or specialized technologies. However, the fact that importance is still distributed across many dimensions rather than concentrated in a few, suggesting that the model leverages the full 300-dimensional skill representation.

Field of study has the second highest importance, showing how field-specific knowledge can distinguish between roles with similar skill sets but different educational backgrounds. For example, a candidate with skills in "Python" and "Machine Learning" might be predicted to be a Data Scientist if their qualification is "Master's in Data Science", but as a Software Engineer if their qualification is "Bachelor's in Computer Science". The semantic embeddings capture these nuanced relationships.

Experience level has moderate importance, reflecting that while experience is relevant for distinguishing between senior and junior roles, it alone is not sufficient for accurate role prediction. A Senior candidate could be a Senior Data Scientist, Senior Software Engineer, or Senior DevOps Engineer, and the experience level alone cannot distinguish between these. However, the experience level still provides some value to the overall model.

Degree level has the lowest importance, suggesting that the specific field of study is more discriminative than the level of education. This indicates that a Bachelor's in Computer Science might be more predictive of a Software Engineer role than a Master's or PhD in an unrelated field.

4.3.3 Model Calibration Assessment

The prediction probability distribution analysis indicates our model is well-calibrated, meaning that the predicted probabilities accurately reflect the true likelihood of each class. Calibration is assessed by examining the relationship between predicted probabilities and actual outcomes. For a well-calibrated model, when the model predicts a class with 90% probability, that class should actually be correct approximately 90% of the time.

Our analysis shows that high-confidence predictions (probability > 0.95) correspond to high accuracy, suggesting that the model's confidence scores are reliable indicators of prediction certainty.

The calibration analysis also reveals that the model rarely makes low-confidence predictions, with most predictions having maximum probabilities above 0.90. This suggests that the model has learned clear decision boundaries and is rarely uncertain in it's predictions. While high confidence is generally good, overconfidence can be a concern if the model is applied to data that differs significantly from the training distribution.

Since the model exhibits good calibration it is able to provide the top-k predictions (e.g., top 3 most likely roles) with associated confidence scores, which is valuable if users want to see alternative tech job role suggestions.

5 Discussion

5.1 Strengths

Our model has several key strengths that contribute to its exceptional performance and practical utility. These strengths include the feature engineering approach, model architecture, evaluation methodology, and system design, all working together to accurately predict job roles.

The hybrid feature engineering approach a significant strength of our system. The combination of ordinal encoding, semantic embeddings, and word embeddings captures multiple aspects of candidate profiles. Ordinal encoding preserves hierarchical relationships (e.g., PhD $>$ Master's $>$ Bachelor's) that are lost in one-hot encoding, while semantic embeddings capture nuanced field-specific information (e.g., "Computer Science" vs. "Software Engineering" vs. "Data Science") that cannot be captured by simple keyword matching. Word embeddings for skills also capture semantic relationships between technologies (e.g., That "Python" and "NumPy" are related), enabling the model to understand that candidates with similar but not identical skill sets might be suitable for the same role.

This multi-faceted representation enables the model to leverage both structured information (hierarchy, progression, explicit categories) and unstructured information (semantic similarity, contextual relationships). For example, the model can learn that a candidate with "Master's in Data Science" (semantic embedding) and skills in

"Python", "TensorFlow", "SQL" (word embeddings) is most likely a Data Scientist, even if they don't have the exact skill combination seen in training. This flexibility is crucial for real-world deployment and use.

Achieving 99.33% accuracy on a 20-class classification problem is remarkable, particularly given the complexity of the feature space (686 dimensions) and the multi-class nature of the task. This performance is competitive with or superior to many published results in the topic, demonstrating the effectiveness of our hybrid feature engineering approach. The high accuracy is achieved without extensive data augmentation or complex ensemble methods, making the system simple to understand and deploy.

The evaluation methodology provides a comprehensive understanding of model behavior beyond just simple accuracy metrics. The use of multiple metrics (accuracy, weighted F1, macro F1) ensures that performance is assessed from multiple perspectives, accounting for class imbalance and providing insights into both overall and class by class performance. The extensive visualizations (confusion matrix, scatter plots, precision-recall curves, feature importance plots) provide crucial information about our model's strengths, weaknesses, and areas for improvement.

5.2 Limitations

Several limitations should be be noted in order to properly assess the capabilities and scope of our model.

The dataset size is one such limitation. With 1,000 samples across 20 classes, the dataset is relatively small by modern machine learning standards, which often contain tens or hundreds of thousands of samples. While the model achieves excellent performance on this dataset, larger datasets would allow for a more through testing and potentially reveal additional patterns or edge cases. A larger dataset could allow for a more sophisticated evaluation, such as using temporal (how role requirements change over time) or geographic (location requirements for jobs) information.

The small dataset also limits the ability to perform extensive feature importance analysis. With more data, we could more confidently assess the contribution of each feature type (ordinal encoding, semantic embeddings, word embeddings) and potentially identify redundant or less useful features. Additionally, a larger dataset would provide more examples of rare but important cases, such as candidates with unusual skill combinations.

Class imbalance, while mitigated through stratified splitting, still remains a limitation. Some classes in our dataset (e.g., Finance, Video Game Designer) have very few samples, and these classes were filtered out during preprocessing to ensure reliable stratified splitting. In actual deployment, these rare classes might require special handling, such as data augmentation, specialized loss functions, or active learning strategies.

Even among the classes that were retained, there is variation in sample sizes, with some roles having 50+ samples while others have closer to 40. While the model achieves high performance across all classes, the performance might be less consistent if the dataset distribution changes. This is particularly relevant for deployment scenarios where the distribution of candidates might differ.

The model's computational requirements represents a practical limitation. The use of large pre-trained models (Word2Vec $\sim$1.5 GB) requires significant storage and memory. The Word2Vec model alone requires approximately 4GB of RAM when loaded, and the Sentence Transformer model requires additional memory. This limits deployment options to systems with sufficient resources, excluding lightweight devices.

The initial model embedding generation can also be time consuming. The Word2Vec model for 1,000 candidates can take between 5-10 minutes to generate embeddings. While this is acceptable for training and batch processing, it could be a bottleneck for real-time applications that need to process many candidates quickly. However, once models are cached, subsequent runs are much faster.

Domain specificity is a fundamental aspect of our model. The model is trained specifically on tech industry roles, and its feature engineering (particularly the Word2Vec embeddings and the qualification patterns) is optimized for tech industry terminology and conventions. Generalization to other industries (e.g., healthcare, finance, education) would require retraining with domain-specific data and potentially different feature engineering approaches.

For example, a job role prediction system for healthcare might need to consider medical certifications, clinical experience, and specialized knowledge that are not captured by the current skill and qualification representations. Similarly, the Word2Vec model is trained on Google News and may not properly represent industry-specific terminology outside of technology. This domain specificity means that deploying the system to new domains requires significant additional changes to our architecture.

5.3 Ethical Considerations and Bias

A critical aspect of deploying automated hiring systems is the potential for algorithmic bias, which can perpetuate existing inequalities [12]. While our system relies on technical skills and education, bias could be introduced through word embeddings (potentially containing gender or cultural biases), historical biases in the dataset, or socioeconomic factors that influence preferred qualifications and experience. Future iterations must audit predictions across demographic groups to ensure fairness, potentially using debiasing techniques, fairness constraints, or more diverse training data. Additionally, ethical considerations include transparency, accountability, and human agency in hiring decisions, with the system serving as a tool to assist rather than replace human judgment.

5.4 Future Work

There are several directions for future improvement for our model. Model versioning and tracking would enable performance monitoring over time and facilitate A/B testing, with logging of predictions and outcomes allowing us to detect distribution drift and inform retraining decisions. Comparison with Large Language Models represents an interesting research direction, though LLMs come with significant computational costs (seconds vs. milliseconds for inference). A hybrid approach could potentially combine our model's efficiency with LLMs flexibility.

Additional future work directions include: extending to additional domains outside of tech roles, incorporating temporal information, integrating with job posting data to provide role-specific skill recommendations, and developing mobile applications for broader accessibility.

6 Conclusion

This project presents a comprehensive machine learning system for predicting tech job roles from candidate profiles. Through a hybrid feature engineering approach combining ordinal encoding, semantic embeddings, and word embeddings, we achieve exceptional performance with 99.33% accuracy on a 20-class classification task.

The key contributions of this work include demonstration of effective hybrid feature engineering for job role prediction, near-perfect classification performance across diverse tech roles, development of a complete end-to-end pipeline from data preprocessing to web deployment, and comprehensive evaluation methodology with multiple metrics and visualizations.

The system's practical applications extend to recruitment platforms, career guidance tools, and automated candidate screening systems. The high accuracy and comprehensive evaluation of the model, showcase its reliability for real-world deployment.

Future work should focus on scaling the system, improving explainability, and extending to additional domains. The foundation established in this project provides a strong basis for these enhancements.

7 Author Contributions

- **Adrean Cajigas**: Organized and coordinated discussion within the team. Set up meetings and goals for different aspects of the project. Helped create the base classes for the prediction pipeline and assisted in system integration testing.
- **Geoffrey Wang**: Performed data pre-processing to expand the dataset utility, including cleaning the CSV inputs, handling null values, and implementing the logic for stratified splitting to ensure class balance.
- **Chiara Bondavalli**: Developed data handling strategies, including filtering out classes with insufficient samples (<3) and designing the module to load datasets and return manageable subtasks for training.
- **Kapila Mhaiskar**: Integrated the Word2Vec model for skill embedding generation. Managed the text processing pipeline that parses comma-separated skills and computes the averaged 300-dimensional vector representations.
- **Julio Flores Cristerna**: Implemented the Sentence Transformer integration for qualification fields. Designed the hybrid feature engineering approach that combines semantic embeddings with ordinal encoding for experience levels.
- **Alex Bott**: Developed the architecture and training pipeline for the XGBoost model. Designed the data pipeline to feed processed feature vectors into the classifier and managed the model serialization.
- **Shayan Khoshkeifi**: Performed RandomizedSearchCV for hyperparameter optimization. Defined the search space for learning rates and tree depth to evaluate speed and accuracy.
- **George Rizk**: Developed the evaluation metrics module, calculating Weighted and Macro F1 scores. Generated the classification report data to assess model precision and recall across all 20+ job roles.
- **Awaab Mirghani**: Created visualization scripts for model performance analysis. Generated the confusion matrix, actual vs. predicted scatter plots, and feature importance charts to maximize visual information.
- **Tianyang Liu**: Designed the backend for the web application using Flask. Implemented the API endpoints to receive user profile data, invoke the prediction pipeline, and return JSON results in real-time.
- **Noah Bierman**: Designed and tested the performance of the frontend interface using HTML and Tailwind CSS. Integrated Alpine.js for dynamic form handling and ensured the usability of the real-time prediction display.
- **Ethan Liu**: Compiled the final project report and documentation. Wrote the methodology and results sections, managed the bibliography, and performed final quality assurance testing on the completed model.

References

[1] Shetty, C. K. (2023). Candidate Job Role Dataset. Kaggle. [Online]. Available:

https://www.kaggle.com/datasets/ckshetty/candidate-job-role-dataset.

[2] Baval, O., & Gupta, S. (2025). Job Role Prediction System Using Machine Learning. *International Journal of Creative Research Thoughts (IJCRT)*, 13(4), 523-528.

[3] D. C. Ertugrul & S. Bitirim. (2022). Semantic Approaches Survey for Job Recommender Systems. *CEUR Workshop Proceedings*.

[4] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

[5] M. E. Basirat & A. S. A. Mohamed. (2018). A Text Classifier Using Weighted Average Word Embedding. *2018 International Japan-Africa Conference on Electronics, Communications and Computations (JAC-ECC)*, Alexandria, Egypt.

[6] 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*.

[7] Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., & Zhou, M. (2020). MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers. *Advances in Neural Information Processing Systems*, 33, 5776-5788.

[8] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[9] Ali, M., Baig, R., & Zainlabuddin, M. (2024). Automated Resume Analysis & Skill Suggesting Website Using NLP. *International Journal of Engineering and Science Research*, 14(2).

[10] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Science & Business Media.

[11] Saito, T., & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS one*, 10(3), e0118432.

[12] Raghavan, M., Barocas, S., Kleinberg, J., & Levy, K. (2020). Mitigating bias in algorithmic hiring: Evaluating claims and practices. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 469-481.