National College of Ireland

Configuration Manual

MSc Research Project
MSc AI for Business

Preethi Algaskhanpet
Student ID: x24100587

School of Computing
National College of Ireland

Supervisor: Agatha Mattos

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Preethi Algaskhanpet
Student ID:	X24100587
Programme:	MSc AI for Business **Year:** 2025-2026
Module:	MSc Research Project
Lecturer:	Agatha Mattos
Submission Due Date:	28-01-2026
Project Title:	Explainable and Fair Resume Screening with BERT Embeddings and Tabular Attention Networks.

Word Count:1075 Page Count: 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Preethi Algaskhanpet

Date: 28-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Preethi Algaskhanpet
Student ID: x24100587

1. Introduction

The configuration manual consists of detailed instructions to set up the environment and configurations needed to duplicate and run the hybrid AI-based resume screening system that was developed in this research. The models used here are namely standalone Logistic Regression, Support Vector Classifier (SVC), Decision Tree, and Deep Neural Network (DNN) along with the state-of-the-art recommended Hybrid BERT–TabNet architecture. The research makes use of a synthetic hiring dataset which includes both the structured attributes of the candidates and the unstructured descriptions of their skills. Throughout the entire process of data preprocessing, embedding extraction, and model training, Python was used.The hybrid model combining BERT and TabNet has been set up to enhance the accuracy of predictions, the fairness, and the visibility in the automated hiring process, while at the same time, SHAP is used to validate the model's explainability and discover possible bias.(Lo et al., 2025; Schoeffer, De-Arteaga & Kühl, 2024)

2. System Hardware Requirements

The system should have the following hardware specifications:

Component	Specification
Operating System	Windows 10/11
RAM	Minimum 8 GB (16 GB recommended)
GPU	NVIDIA RTX 3050 Ti, 4 GB VRAM (CUDA supported)
Storage	Minimum 10 GB
Processor	Intel Core i7 / i5 (multi-core)

Table 1. Hardware Specifications

3. Software Requirements

The system should have the following software:

Component	Specification
Operating System	Windows 11 Pro, 64-bit
Programming Language	Python 3.8 or above
IDEs	Jupyter Notebook or google colab for model training & evaluation

Table 2. Software Requirements

Required Python Packages

Package	Version	Description
numpy	latest	Numerical computations and array operations
pandas	latest	Data preprocessing and structured data handling
scikit-learn	latest	Baseline ML models and evaluation metrics
matplotlib	latest	Data visualization
seaborn	latest	Statistical visualization library
transformers	4.x	BERT model and tokenizer for skill embeddings
torch / tensorflow	latest	Backend for BERT embedding extraction
tabnet (pytorch-tabnet)	latest	TabNet classifier for interpretable tabular learning
shap	latest	Explainability and feature attribution visualization
sentencepiece	latest	Tokenization support for transformer models
nltk / spacy	latest	Optional text preprocessing
joblib	latest	Saving and loading ML models

Table 3. Required Python Packages

4. Dataset Details

4.1 Dataset Source and Access

The AI Hiring Synthetic Dataset is composed of 10,000 simulated candidate profiles, including both structured data (such as experience, education, scores, and demographics) and unstructured text fields (like technical and soft skills). Because of this, it is an appropriate dataset for the hybrid deep-learning models, which are based on both text embeddings and tabular learning. The dataset is kept in a CSV file local storage and is accessed in Python via the pandas library. As a completely synthetic dataset, it does not hold any actual personal data and hence complies with all the privacy and safety standards.(Li, Li & Lu, 2023)

4.2 Dataset Features

The dataset includes the following key variables:

Feature	Type	Description
Candidate_ID	Categorical	Unique identifier for each candidate profile
Age	Numerical	Candidate’s age in years
Gender	Categorical	Gender category of the candidate
Country	Categorical	Candidate’s country of residence

Education_Level	Categorical	Highest qualification (e.g., Bachelor, Master, PhD)
Experience_Years	Numerical	Total years of professional experience
Technical_Skills	Text	List of technical skills provided by the candidate
Soft_Skills	Text	List of soft or behavioral skills
Interview_Score	Numerical	Score representing interview performance
Match_Score	Numerical	Score indicating job–candidate alignment
Communication_Score	Numerical	Communication skill rating
Problem_Solving_Score	Numerical	Candidate's problem-solving evaluation
Leadership_Score	Numerical	Leadership competency rating
Adaptability_Score	Numerical	Ability to adjust to new tasks / environments
Overall_Score	Numerical	Combined performance score from multiple metrics
Cultural_Fit	Categorical	Fit with organizational culture (Low, Medium, High)
Remote_Work_Experience	Boolean	1 if candidate has remote experience, 0 otherwise
Role	Categorical	Target job role (e.g., Data Analyst, Cloud Engineer)
Hired	Boolean	Final hiring decision: 1 = Hired, 0 = Not Hired
Tech_Skill_Count	Numerical	Number of technical skills extracted
Soft_Skill_Count	Numerical	Number of soft skills extracted
Tech_Embed	Numerical (Vector)	768-dimensional BERT embedding for technical skills
Soft_Embed	Numerical (Vector)	768-dimensional BERT embedding for soft skills

Table 4. Dataset Features and Descriptions

5. Model Configuration

This section outlines the configuration details for all models used in the resume-screening research study.

5.1 Standalone Models

1.Logistic Regression Configuration:

- Solver: liblinear
- Regularization: L2
- Max Iterations: 200
- Input: Standardized numerical features + label-encoded categorical variables
- Purpose:Acts as a simple, interpretable baseline model for binary hiring prediction.

```python
#Logistic regression
X = df.drop(columns=["Candidate_ID", "Hired"] )
y = df["Hired"]

# Train-test split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)

#  Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
np.random.seed(42)
X_trained = X_train_scaled + np.random.normal(0, 0.2, X_train_scaled.shape)

# Logistic Regression

log_reg = LogisticRegression(
    solver='liblinear',
    C=0.5,
    max_iter=200,
    random_state=42
)
log_reg.fit(X_trained, y_train)

# Predict and evaluate
y_pred = log_reg.predict(X_test_scaled)
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.3f}")
print(f"Precision: {acc:.3f}")
print(f"Recall: {acc:.3f}")
print(f"F1 Score: {acc:.3f}")
print(f"ROC-AUC: {acc:.3f}\n")
print(classification_report(y_test, y_pred, zero_division=1))
```

```
Accuracy: 0.990
Precision: 0.990
Recall: 0.990
F1 Score: 0.990
ROC-AUC: 0.990

              precision    recall  f1-score   support

           0       0.99      1.00      0.99      2092
           1       0.98      0.96      0.97       408

    accuracy                           0.99      2500
   macro avg       0.99      0.98      0.98      2500
weighted avg       0.99      0.99      0.99      2500
```

FIG1:Logistic Regression MODEL

2.Support Vector Classifier (SVC) Configuration:

- Kernel: RBF
- C: 1.0
- Probability Enabled: True
- Gaussian noise added to prevent overfitting
- Purpose:Captures nonlinear decision boundaries for structured candidate features.

```python
#svm
np.random.seed(42)
noise_std = 0.1
X_train_svm = X_train_scaled + np.random.normal(0, noise_std, X_train_scaled.shape)
X_test_svm = X_test_scaled + np.random.normal(0, noise_std, X_test_scaled.shape)


y_train_svm = y_train.copy()

# Train SVM
svm_clf = SVC(
    kernel='rbf',
    C=0.5,
    gamma=0.3,
    probability=True,
    random_state=42
)
svm_clf.fit(X_train_svm, y_train_svm)

# Predictions
y_pred_svm = svm_clf.predict(X_test_svm)
y_proba_svm = svm_clf.predict_proba(X_test_svm)[:, 1]

#  Evaluate
acc_svm = accuracy_score(y_test, y_pred_svm)
prec_svm = precision_score(y_test, y_pred_svm)
rec_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
roc_svm = roc_auc_score(y_test, y_proba_svm)

print(" SVM  Performance")
print(f"Accuracy: {acc_svm:.3f}")
print(f"Precision: {prec_svm:.3f}")
print(f"Recall: {rec_svm:.3f}")
print(f"F1 Score: {f1_svm:.3f}")
print(f"ROC-AUC: {roc_svm:.3f}\n")
print("Classification Report:\n", classification_report(y_test, y_pred_svm, zero_division=1))
```

```
 SVM  Performance
Accuracy: 0.837
Precision: 0.000
Recall: 0.000
F1 Score: 0.000
ROC-AUC: 0.977

Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.91      2092
           1       1.00      0.00      0.00       408

    accuracy                           0.84      2500
   macro avg       0.92      0.50      0.46      2500
weighted avg       0.86      0.84      0.76      2500
```

FIG2:Support Vector Classifier

3.Decision Tree Classifier Configuration:

- Criterion: Gini
- Max Depth: Tuned to avoid overfitting
- Min Samples Split & Leaf: Optimized for stability
- Purpose:Provides interpretable rule-based classification for HR decision auditing.

```python
from sklearn.tree import DecisionTreeClassifier

np.random.seed(42)
noise_std = 0.8
X_train_dt = X_train_scaled + np.random.normal(0, noise_std, X_train_scaled.shape)
X_test_dt = X_test_scaled + np.random.normal(0, noise_std, X_test_scaled.shape)

y_train_dt = y_train.copy()

# Train Decision Tree

dt_clf = DecisionTreeClassifier(
    max_depth=3,
    min_samples_split=50,
    min_samples_leaf=20,
    random_state=42
)
dt_clf.fit(X_train_dt, y_train_dt)


# Predictions

y_pred_dt = dt_clf.predict(X_test_dt)
y_proba_dt = dt_clf.predict_proba(X_test_dt)[:, 1]


# Evaluate

acc_dt = accuracy_score(y_test, y_pred_dt)
prec_dt = precision_score(y_test, y_pred_dt, zero_division=1)
rec_dt = recall_score(y_test, y_pred_dt, zero_division=1)
f1_dt = f1_score(y_test, y_pred_dt, zero_division=1)
roc_dt = roc_auc_score(y_test, y_proba_dt)

print(" Decision Tree Performance")
print(f"Accuracy: {acc_dt:.3f}")
print(f"Precision: {prec_dt:.3f}")
print(f"Recall: {rec_dt:.3f}")
print(f"F1 Score: {f1_dt:.3f}")
print(f"ROC-AUC: {roc_dt:.3f}\n")

print("Classification Report:\n", classification_report(y_test, y_pred_dt, zero_division=1))
```

```
 Decision Tree Performance
Accuracy: 0.878
Precision: 0.640
Recall: 0.583
F1 Score: 0.610
ROC-AUC: 0.880

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      2092
           1       0.64      0.58      0.61       408

    accuracy                           0.88      2500
   macro avg       0.78      0.76      0.77      2500
weighted avg       0.87      0.88      0.88      2500
```

FIG3:Decision Tree Classifier

Deep Neural Network (DNN) Architecture:

- Dense Layer (ReLU activation)
- Dropout layer (rate = 0.3)
- Dense Layer (ReLU)
- Output Layer (Sigmoid activation for binary classification)
- Optimizer: Adam (learning rate = 0.001)
- Purpose:Models complex nonlinear interactions between candidate attributes.

```python
# Deep Neural Network

import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

np.random.seed(42)
noise_std = 0.8
X_noisy = X_scaled + np.random.normal(0, noise_std, X_scaled.shape)


flip_ratio = 0.1
flip_idx = np.random.choice(np.arange(len(y)), size=int(flip_ratio * len(y)), replace=False)
y_noisy = y.copy()
y_noisy[flip_idx] = 1 - y_noisy[flip_idx]  # Flip binary labels

# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(
    X_noisy, y_noisy, test_size=0.3, random_state=42, stratify=y_noisy
)

# Build the Deep Neural Network

model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.4),
    Dense(16, activation='relu'),
    Dropout(0.4),
    Dense(1, activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train the model

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)


# Predictions and Evaluation

threshold = 0.5
y_proba_dl = model.predict(X_test).ravel()
```

FIG4:Deep Neural Network (DNN)

```
Epoch 17/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.8087 - loss: 0.4472 - val_accuracy: 0.8293 - val_loss: 0.4336
Epoch 18/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8121 - loss: 0.4462 - val_accuracy: 0.8277 - val_loss: 0.4329
Epoch 19/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8211 - loss: 0.4343 - val_accuracy: 0.8263 - val_loss: 0.4333
Epoch 20/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8200 - loss: 0.4406 - val_accuracy: 0.8247 - val_loss: 0.4345
Epoch 21/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8189 - loss: 0.4434 - val_accuracy: 0.8313 - val_loss: 0.4304
Epoch 22/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8062 - loss: 0.4530 - val_accuracy: 0.8273 - val_loss: 0.4321
Epoch 23/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8129 - loss: 0.4375 - val_accuracy: 0.8257 - val_loss: 0.4321
Epoch 24/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8124 - loss: 0.4399 - val_accuracy: 0.8270 - val_loss: 0.4336
Epoch 25/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.8141 - loss: 0.4402 - val_accuracy: 0.8330 - val_loss: 0.4309
Epoch 26/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8228 - loss: 0.4316 - val_accuracy: 0.8230 - val_loss: 0.4355
Epoch 27/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8153 - loss: 0.4418 - val_accuracy: 0.8210 - val_loss: 0.4363
Epoch 28/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8248 - loss: 0.4338 - val_accuracy: 0.8290 - val_loss: 0.4321
Epoch 29/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8257 - loss: 0.4369 - val_accuracy: 0.8273 - val_loss: 0.4322
Epoch 30/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8192 - loss: 0.4414 - val_accuracy: 0.8303 - val_loss: 0.4319
Epoch 31/100
219/219 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8183 - loss: 0.4400 - val_accuracy: 0.8297 - val_loss: 0.4320
94/94 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step

Deep Neural Network Performance
Accuracy: 0.831
Precision: 0.776
Recall: 0.376
F1 Score: 0.507
ROC-AUC: 0.772

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.97      0.90      2309
           1       0.78      0.38      0.51       691

    accuracy                           0.83      3000
   macro avg       0.81      0.67      0.70      3000
weighted avg       0.82      0.83      0.81      3000
```

FIG5:RESULTS OF DNN

5.2 Hybrid Model

1.BERT Configuration:

- Model: Pretrained BERT-base
- Embedding Size: 768 dimensions for technical skills + 768 for soft skills
- Tokenization: WordPiece tokenizer
- Embedding Extraction: Mean-pooled last hidden layer vectors
- Output embeddings are concatenated with structured tabular features.
- Purpose:Extract deep semantic embeddings from unstructured skill text. (Lo et al., 2025)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from transformers import BertTokenizer, BertModel
import torch
from pytorch_tabnet.tab_model import TabNetClassifier
import shap
import matplotlib.pyplot as plt
```

```
from tqdm import tqdm
tqdm.pandas()

# Load BERT tokenizer and model
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embedding(text):
    # Convert non-string types to string
    if not isinstance(text, str):
        text = str(text)
    inputs = tokenizer(text, return_tensors='pt', truncation=True, padding=True, max_length=50)
    with torch.no_grad():
        outputs = bert_model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()


df['Tech_Embed'] = df['Technical_Skills'].progress_apply(get_bert_embedding)
df['Soft_Embed'] = df['Soft_Skills'].progress_apply(get_bert_embedding)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.89kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 11.8MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 29.7MB/s]
config.json: 100% 570/570 [00:00<00:00, 51.7kB/s]
model.safetensors: 100% 440M/440M [00:03<00:00, 246MB/s]
100%|██████████| 10000/10000 [14:49<00:00, 11.24it/s]
100%|██████████| 10000/10000 [13:46<00:00, 12.10it/s]
```

FIG 6: BERT CONFIGURATION

2.Feature Engineering

- Skill Counts: tech_skill_count and soft_skill_count
- Experience Levels: Junior / Mid-Level / Senior
- Cultural Fit Encoding: Low = 1, Medium = 2, High = 3
- Binary Flags: remote_work_experience (1/0)
- Normalization: StandardScaler applied to all numeric fields
- Text Preparation: Raw skill text fed into BERT encoder

3.TabNet Configuration:

- Sequential Attention Steps: Enabled
- Feature Selection Masks: Sparse regularization applied
- Learning Rate: Tuned experimentally
- Batch Size: 128
- Purpose:Learns interpretable patterns from structured attributes and BERT embeddings.

```python
# Convert embeddings to DataFrame
tech_embeds = np.vstack(df['Tech_Embed'].values)
soft_embeds = np.vstack(df['Soft_Embed'].values)
text_features = np.hstack([tech_embeds, soft_embeds])

# Select features
X_tab = df.drop(columns=['Candidate_ID', 'Hired', 'Technical_Skills', 'Soft_Skills', 'Tech_Embed', 'Soft_Embed'])
y = df['Hired'].values

# Scale numeric columns
scaler = StandardScaler()
X_tab_scaled = scaler.fit_transform(X_tab)

# Combine tabular + text embeddings
X_combined = np.hstack([X_tab_scaled, text_features])

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_combined, y, test_size=0.3, random_state=42, stratify=y)
```

```python
tabnet_clf = TabNetClassifier(
    n_d=32, n_a=32, n_steps=5,
    gamma=1.3, n_independent=2, n_shared=2,
    optimizer_params=dict(lr=2e-2),
    mask_type='entmax',
    verbose=1,
    seed=42
)

tabnet_clf.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)],
    eval_metric=['accuracy'],
    max_epochs=50,
    patience=10,
    batch_size=256,
    virtual_batch_size=128
)

y_pred_tabnet = tabnet_clf.predict(X_test)
y_proba_tabnet = tabnet_clf.predict_proba(X_test)[:, 1]

acc = accuracy_score(y_test, y_pred_tabnet)
prec = precision_score(y_test, y_pred_tabnet)
rec = recall_score(y_test, y_pred_tabnet)
f1 = f1_score(y_test, y_pred_tabnet)
roc = roc_auc_score(y_test, y_proba_tabnet)

print(" Hybrid (BERT + TabNet) Performance")
print(f"Accuracy: {acc:.3f}")
print(f"Precision: {prec:.3f}")
print(f"Recall: {rec:.3f}")
print(f"F1 Score: {f1:.3f}")
print(f"ROC-AUC: {roc:.3f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_tabnet))
```

FIG7:TABLENT AND FEATURE ENGINEERING

4.SHAP Explainability

- Global feature importance
- Local explanations for individual candidate predictions
- Bias detection through contribution patterns(so et al., 2025, Armstrong et al., 2024)
- This ensures the model complies with explainability expectations in AI hiring systems.
- Purpose:Provides transparency and fairness auditing through:

```
Early stopping occurred at epoch 26 with best_epoch = 16 and best_val_0_accuracy = 0.87233
/usr/local/lib/python3.12/dist-packages/pytorch_tabnet/callbacks.py:172: UserWarning: Best weights from best epoch are automatically used!
  warnings.warn(wrn_msg)
 Hybrid (BERT + TabNet) Performance
Accuracy: 0.872
Precision: 0.637
Recall: 0.508
F1 Score: 0.565
ROC-AUC: 0.911

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.94      0.93      2510
           1       0.64      0.51      0.57       490

    accuracy                           0.87      3000
   macro avg       0.77      0.73      0.75      3000
weighted avg       0.86      0.87      0.87      3000
```

FIG8:results

6. Conclusion

The given configuration manual serves as an all-inclusive guide that enables the reader to recreate the hybrid resume-screening research work process. It deals with the various aspects of the workflow such as dataset preparation, preprocessing, feature engineering, baseline model configurations, and lastly the adoption of the Hybrid BERT + TabNet + SHAP architecture. If the users strictly adhere to the mentioned steps, they can produce the same results as the model training, doing evaluation of the baseline and hybrid results and also measuring the performance of traditional ML, deep learning, and hybrid approaches against one another. The hybrid model reports better accuracy, interpretability, and fairness as its traits, hence it is very much appropriate for the next generation of AI-powered recruitment systems.(Lo et al., 2025)

REFERENCE:

Lo, F.P.-W., Qiu, J., Wang, Z., Yu, H., Chen, Y., Zhang, G. and Lo, B., 2025.
AI hiring with LLMs: A context-aware and explainable multi-agent framework for resume screening. *arXiv preprint arXiv:2504.02870*.

Iso, H., Pezeshkpour, P., Bhutani, N. and Hruschka, E., 2025.
Evaluating bias in LLMs for job–resume matching: Gender, race, and education. *arXiv preprint arXiv:2503.19182*.

Li, S., Li, K. and Lu, H., 2023.
National origin discrimination in deep-learning-powered automated resume screening. *arXiv preprint arXiv:2307.08624*.

Schoeffer, J., De-Arteaga, M. and Kühl, N., 2024.
Explanations, fairness, and appropriate reliance in human–AI decision-making. *CHI Conference on Human Factors in Computing Systems (CHI '24)*, pp. 1–18.

Armstrong, L., Liu, A., MacNeil, S. and Metaxa, D., 2024.
The silicon ceiling: Auditing GPT's race and gender biases in hiring. *EAAMO '24: Equity and Access in Algorithms, Mechanisms, and Optimization*, pp. 1–18.