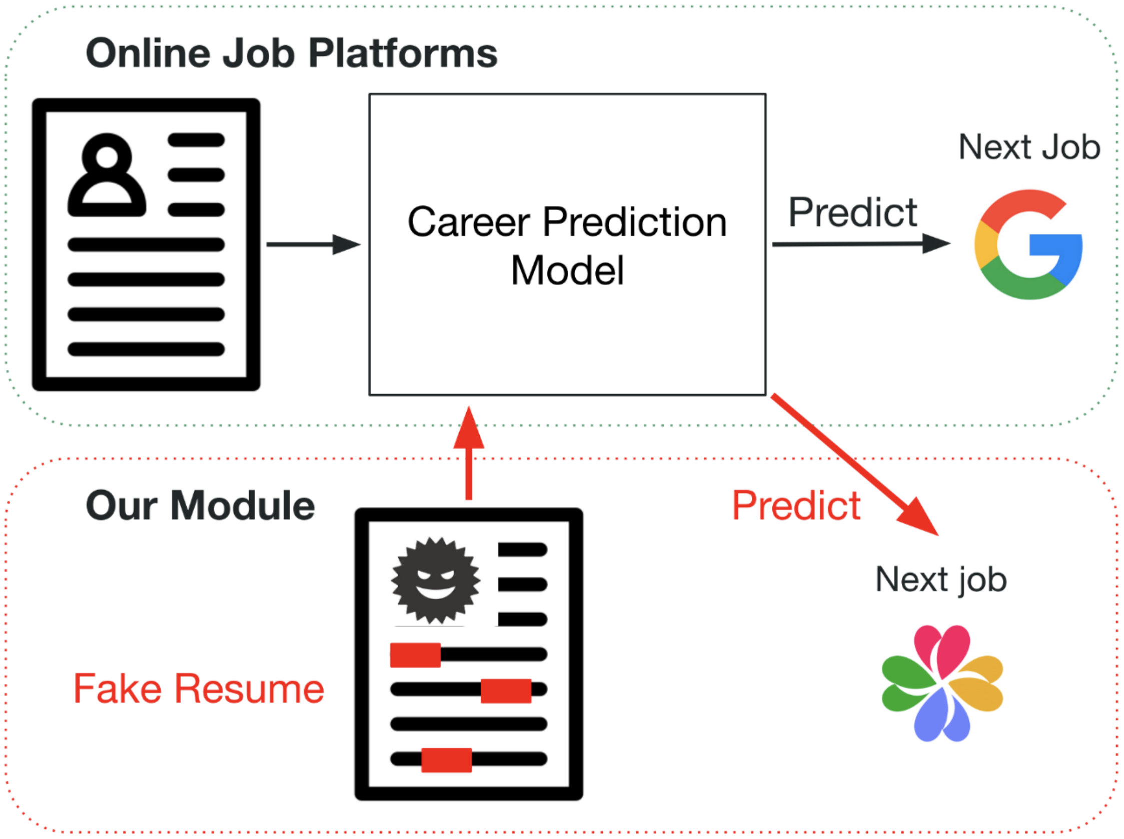
Fake Resume Classification with Novel ABACUS based LSTM



```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from datasets import load_dataset
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
```

```python
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.utils import resample
from sklearn.metrics import classification_report, roc_auc_score
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from wordcloud import WordCloud

np.random.seed(42)
tf.random.set_seed(42)

sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)
import matplotlib

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

def clean_text(text):
    if pd.isna(text):
        return ""
    text = text.lower()
    text = re.sub(r'\b[\w\.-]+@[\w\.-]+\.\w+\b', '', text)
    text = re.sub(r'https?://\S+|www\.\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stop_words]
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return ' '.join(tokens)

try:
    dataset = load_dataset("AzharAli05/Resume-Screening-Dataset")
    df = dataset['train'].to_pandas()
except Exception as e:
    print(f"Error loading dataset: {e}")
    exit()

df['Resume_clean'] = df['Resume'].apply(clean_text)
df['Job_Description_clean'] = df['Job_Description'].apply(clean_text)
df['Reason_for_decision_clean'] = df['Reason_for_decision'].apply(clean_text)
```

```
print("Original Decision distribution:",
df['Decision'].value_counts(dropna=False).to_dict())
df['Decision'] = df['Decision'].str.lower().map({'select': 1, 'reject':
0}).fillna(0).astype(int)
print("Mapped Decision distribution:",
df['Decision'].value_counts().to_dict())

def flag_fake_resume(reason_clean, resume_clean, decision):
    if not reason_clean or not resume_clean:
        return 0
    fake_keywords = [
        'inconsistent', 'exaggerated', 'unverified', 'false', 'fabricated',
'misleading',
        'overstated', 'questionable', 'unsubstantiated', 'implausible',
'discrepancy',
        'overqualified', 'unrealistic', 'suspicious', 'invalid', 'unreliable'
    ]
    if decision == 0 and any(keyword in reason_clean for keyword in
fake_keywords):
        return 1
    buzzwords = ['expert', 'worldclass', 'leading', 'unparalleled',
'visionary', 'guru']
    resume_words = word_tokenize(resume_clean)
    buzzword_count = sum(resume_words.count(bw) for bw in buzzwords)
    if buzzword_count > 5:
        return 1
    return 0

df['is_fake'] = df.apply(lambda x:
flag_fake_resume(x['Reason_for_decision_clean'], x['Resume_clean'],
x['Decision']), axis=1)

def generate_fake_resume(original_resume, role):
    buzzwords = ['expert', 'worldclass', 'leading', 'unparalleled',
'visionary']
    large_companies = ['tesla', 'google', 'microsoft', 'amazon', 'facebook']
    fake_resume = original_resume + " " + "
".join(np.random.choice(buzzwords, size=10))
    fake_resume += f" chief {role.lower()} at
{np.random.choice(large_companies)} years experience developed aidriven
blockchain solution"
    return clean_text(fake_resume)

n_fake = int(len(df) * 0.5 / (1 - 0.5)) - df['is_fake'].sum()
if n_fake > 0:
    fake_resumes = df[['Resume_clean', 'Role']].sample(n_fake,
random_state=42, replace=True)
    fake_resumes['Resume_clean'] = fake_resumes.apply(lambda x:
```

```
generate_fake_resume(x['Resume_clean'], x['Role']), axis=1)
    fake_df = pd.DataFrame({
        'Role': fake_resumes['Role'],
        'Resume': fake_resumes['Resume_clean'],
        'Resume_clean': fake_resumes['Resume_clean'],
        'Decision': 0,
        'Reason_for_decision': 'Generated fake resume with exaggerated
claims',
        'Reason_for_decision_clean': 'generated fake resume exaggerated
claim',
        'Job_Description': df['Job_Description'].sample(n_fake,
random_state=42, replace=True),
        'Job_Description_clean': df['Job_Description_clean'].sample(n_fake,
random_state=42, replace=True),
        'is_fake': 1
    })
    df = pd.concat([df, fake_df], ignore_index=True)

tfidf = TfidfVectorizer(max_features=5000, stop_words='english')
resume_tfidf = tfidf.fit_transform(df['Resume_clean'])
job_tfidf = tfidf.transform(df['Job_Description_clean'])
similarity_scores = [cosine_similarity(resume_tfidf[i], job_tfidf[i])[0][0]
for i in range(len(df))]
df['similarity_score'] = similarity_scores
df.loc[df['similarity_score'] < 0.25, 'is_fake'] = 1

max_words = 5000
max_len = 50
beads_per_column = 5
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(df['Resume_clean'] + ' ' +
df['Job_Description_clean'])
X_sequences = tokenizer.texts_to_sequences(df['Resume_clean'] + ' ' +
df['Job_Description_clean'])
X_padded = pad_sequences(X_sequences, maxlen=max_len, padding='post',
truncating='post')

tfidf_abacus = TfidfVectorizer(max_features=max_words, stop_words='english')
X_tfidf = tfidf_abacus.fit_transform(df['Resume_clean'] + ' ' +
df['Job_Description_clean']).toarray()
X_abacus = np.zeros((X_padded.shape[0], max_len, beads_per_column))
for i in range(X_padded.shape[0]):
    for t in range(max_len):
        word_idx = X_padded[i, t]
        if word_idx > 0:
            word = tokenizer.index_word.get(word_idx, '')
            if word in tfidf_abacus.vocabulary_:
                tfidf_idx = tfidf_abacus.vocabulary_[word]
                bead_value = X_tfidf[i, tfidf_idx]
                bead_idx = min(int(bead_value * beads_per_column),
beads_per_column-1)
                X_abacus[i, t, bead_idx] = 1

y_fake = df['is_fake'].values
y_decision = df['Decision'].values

df_fake_0 = df[df['is_fake'] == 0]
df_fake_1 = df[df['is_fake'] == 1]
if len(df_fake_0) == 0 or len(df_fake_1) == 0:
    print("Warning: One class missing in is_fake. Skipping balancing for
is_fake.")
    X_fake_train, X_fake_test, y_fake_train, y_fake_test = train_test_split(
        X_abacus, y_fake, test_size=0.2, random_state=42
    )
else:
    n_majority = max(len(df_fake_0), len(df_fake_1))
    df_fake_0_resampled = resample(df_fake_0, replace=True,
n_samples=n_majority, random_state=42) if len(df_fake_0) < n_majority else
df_fake_0
    df_fake_1_resampled = resample(df_fake_1, replace=True,
n_samples=n_majority, random_state=42) if len(df_fake_1) < n_majority else
df_fake_1
    df_fake_balanced = pd.concat([df_fake_0_resampled, df_fake_1_resampled])
    X_fake_sequences =
tokenizer.texts_to_sequences(df_fake_balanced['Resume_clean'] + ' ' +
df_fake_balanced['Job_Description_clean'])
    X_fake_padded = pad_sequences(X_fake_sequences, maxlen=max_len,
padding='post', truncating='post')
    X_fake_tfidf = tfidf_abacus.transform(df_fake_balanced['Resume_clean'] +
' ' + df_fake_balanced['Job_Description_clean']).toarray()
    X_fake_abacus = np.zeros((X_fake_padded.shape[0], max_len,
beads_per_column))
    for i in range(X_fake_padded.shape[0]):
        for t in range(max_len):
            word_idx = X_fake_padded[i, t]
            if word_idx > 0:
                word = tokenizer.index_word.get(word_idx, '')
                if word in tfidf_abacus.vocabulary_:
                    tfidf_idx = tfidf_abacus.vocabulary_[word]
                    bead_value = X_fake_tfidf[i, tfidf_idx]
                    bead_idx = min(int(bead_value * beads_per_column),
beads_per_column-1)
                    X_fake_abacus[i, t, bead_idx] = 1
    y_fake_balanced = df_fake_balanced['is_fake'].values
    X_fake_train, X_fake_test, y_fake_train, y_fake_test = train_test_split(
        X_fake_abacus, y_fake_balanced, test_size=0.2,
stratify=y_fake_balanced, random_state=42
    )

df_dec_0 = df[df['Decision'] == 0]
```

```
df_dec_1 = df[df['Decision'] == 1]
if len(df_dec_0) == 0 or len(df_dec_1) == 0:
    print("Warning: One class missing in Decision. Skipping balancing for
Decision.")
    X_dec_train, X_dec_test, y_dec_train, y_dec_test = train_test_split(
        X_abacus, y_decision, test_size=0.2, random_state=42
    )
else:
    n_majority_dec = max(len(df_dec_0), len(df_dec_1))
    df_dec_0_resampled = resample(df_dec_0, replace=True,
n_samples=n_majority_dec, random_state=42) if len(df_dec_0) < n_majority_dec
else df_dec_0
    df_dec_1_resampled = resample(df_dec_1, replace=True,
n_samples=n_majority_dec, random_state=42) if len(df_dec_1) < n_majority_dec
else df_dec_1
    df_dec_balanced = pd.concat([df_dec_0_resampled, df_dec_1_resampled])
    X_dec_sequences =
tokenizer.texts_to_sequences(df_dec_balanced['Resume_clean'] + ' ' +
df_dec_balanced['Job_Description_clean'])
    X_dec_padded = pad_sequences(X_dec_sequences, maxlen=max_len,
padding='post', truncating='post')
    X_dec_tfidf = tfidf_abacus.transform(df_dec_balanced['Resume_clean'] + '
' + df_dec_balanced['Job_Description_clean']).toarray()
    X_dec_abacus = np.zeros((X_dec_padded.shape[0], max_len,
beads_per_column))
    for i in range(X_dec_padded.shape[0]):
        for t in range(max_len):
            word_idx = X_dec_padded[i, t]
            if word_idx > 0:
                word = tokenizer.index_word.get(word_idx, '')
                if word in tfidf_abacus.vocabulary_:
                    tfidf_idx = tfidf_abacus.vocabulary_[word]
                    bead_value = X_dec_tfidf[i, tfidf_idx]
                    bead_idx = min(int(bead_value * beads_per_column),
beads_per_column-1)
                    X_dec_abacus[i, t, bead_idx] = 1
    y_dec_balanced = df_dec_balanced['Decision'].values
    X_dec_train, X_dec_test, y_dec_train, y_dec_test = train_test_split(
        X_dec_abacus, y_dec_balanced, test_size=0.2, stratify=y_dec_balanced,
random_state=42
    )

def build_abacus_network(max_len, beads_per_column):
    model = Sequential([
        LSTM(128, input_shape=(max_len, beads_per_column),
return_sequences=True),
        Dropout(0.3),
        LSTM(64),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

if len(np.unique(y_fake_train)) > 1:
    model_fake = build_abacus_network(max_len, beads_per_column)
    model_fake.fit(X_fake_train, y_fake_train, epochs=20, batch_size=32,
validation_split=0.2, verbose=1)
else:
    print("Skipping training for is_fake: Only one class present.")

if len(np.unique(y_dec_train)) > 1:
    model_decision = build_abacus_network(max_len, beads_per_column)
    model_decision.fit(X_dec_train, y_dec_train, epochs=20, batch_size=32,
validation_split=0.2, verbose=1)
else:
    print("Skipping training for Decision: Only one class present.")

if len(np.unique(y_fake_train)) > 1:
    y_fake_pred = (model_fake.predict(X_fake_test) > 0.5).astype(int)
    print("Fake Resume Classification Report:")
    print(classification_report(y_fake_test, y_fake_pred))
    if len(np.unique(y_fake_test)) > 1:
        print("ROC-AUC (Fake):", roc_auc_score(y_fake_test,
model_fake.predict(X_fake_test)))
    else:
        print("ROC-AUC (Fake): Skipped due to single class in test set")
else:
    print("No evaluation for is_fake: Model not trained.")

if len(np.unique(y_dec_train)) > 1:
    y_dec_pred = (model_decision.predict(X_dec_test) > 0.5).astype(int)
    print("Selection/Rejection Classification Report:")
    print(classification_report(y_dec_test, y_dec_pred))
    if len(np.unique(y_dec_test)) > 1:
        print("ROC-AUC (Decision):", roc_auc_score(y_dec_test,
model_decision.predict(X_dec_test)))
    else:
        print("ROC-AUC (Decision): Skipped due to single class in test set")
else:
    print("No evaluation for Decision: Model not trained.")

def visualize_abacus_words(features, title, tokenizer, sample_idx, X_padded):
    plt.figure(figsize=(12, 4))
    bead_positions = features
    word_labels = []
    for t in range(min(10, features.shape[0])):
        word_idx = X_padded[sample_idx, t]
        word = tokenizer.index_word.get(word_idx, 'PAD') if word_idx > 0 else
'PAD'
        word_labels.append(word)
        for row in range(features.shape[1]):
            color = 'red' if bead_positions[t, row] else 'gray'
            plt.scatter(t, row, c=color, s=100)
    plt.xticks(range(min(10, features.shape[0])), word_labels, rotation=45,
ha='right')
    plt.yticks(range(features.shape[1]), [f"Bead {i+1}" for i in
range(features.shape[1])])
    plt.title(title)
    plt.grid(True)
    plt.tight_layout()
    plt.show()

sample_idx = 0
visualize_abacus_words(X_abacus[sample_idx], f"Abacus Word Activation (Fake:
{y_fake[sample_idx]})", tokenizer, sample_idx, X_padded)
visualize_abacus_words(X_abacus[sample_idx], f"Abacus Word Activation
(Decision: {y_decision[sample_idx]})", tokenizer, sample_idx, X_padded)

top_roles = df['Role'].value_counts().nlargest(10).index
df['Role_plot'] = df['Role'].where(df['Role'].isin(top_roles), 'Other')
top_reasons = df['Reason_for_decision'].value_counts().nlargest(10).index
df['Reason_plot'] =
df['Reason_for_decision'].where(df['Reason_for_decision'].isin(top_reasons),
'Other')

plt.figure()
sns.countplot(data=df, x='Role_plot', hue='is_fake',
order=top_roles.append(pd.Index(['Other'])))
plt.title('Count of Real vs. Fake Resumes by Role (Top 10 + Other)')
plt.xticks(rotation=45, ha='right')
plt.xlabel('Role')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

plt.figure()
sns.countplot(data=df, x='Decision', hue='is_fake')
plt.title('Count of Real vs. Fake Resumes by Decision')
plt.xlabel('Decision')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

plt.figure()
sns.countplot(data=df, x='Reason_plot', hue='is_fake',
order=top_reasons.append(pd.Index(['Other'])))
```

```python
plt.title('Count of Real vs. Fake Resumes by Reason for Decision (Top 10 +
Other)')
plt.xticks(rotation=45, ha='right')
plt.xlabel('Reason for Decision')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

plt.figure()
sns.countplot(data=df, x='is_fake')
plt.title('Count of Real vs. Fake Resumes')
plt.xlabel('Is Fake (0 = Real, 1 = Fake)')
plt.ylabel('Count')
plt.tight_layout()
plt.show()

plt.figure()
df['Decision'].value_counts().plot.pie(autopct='%1.1f%%', colors=['#4CAF50',
'#F44336'], labels=['Reject', 'Select'])
plt.title('Proportion of Select vs. Reject Decisions')
plt.ylabel('')
plt.tight_layout()
plt.show()

plt.figure()
df['is_fake'].value_counts().plot.pie(autopct='%1.1f%%', labels=['Real',
'Fake'], colors=['#4CAF50', '#F44336'])
plt.title('Proportion of Real vs. Fake Resumes')
plt.ylabel('')
plt.tight_layout()
plt.show()

plt.figure()
df['Role_plot'].value_counts().nlargest(5).plot.pie(autopct='%1.1f%%')
plt.title('Proportion of Top 5 Roles')
plt.ylabel('')
plt.tight_layout()
plt.show()

plt.figure()
df['Reason_plot'].value_counts().nlargest(5).plot.pie(autopct='%1.1f%%')
plt.title('Proportion of Top 5 Reasons for Decision')
plt.ylabel('')
plt.tight_layout()
plt.show()

df['resume_word_count'] = df['Resume_clean'].apply(lambda x:
len(word_tokenize(str(x))))
df['job_word_count'] = df['Job_Description_clean'].apply(lambda x:
```

```
len(word_tokenize(str(x))))

plt.figure()
sns.histplot(data=df, x='resume_word_count', hue='is_fake', bins=50)
plt.title('Resume Word Count Distribution by Real/Fake')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

plt.figure()
sns.histplot(data=df, x='job_word_count', hue='is_fake', bins=50)
plt.title('Job Description Word Count Distribution by Real/Fake')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()

def generate_wordcloud(text, title):
    wordcloud = WordCloud(width=800, height=400, background_color='white',
max_words=100).generate(text)
    plt.figure()
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.title(title)
    plt.axis('off')
    plt.tight_layout()
    plt.show()

real_resumes = ' '.join(df[df['is_fake'] == 0]['Resume_clean'])
fake_resumes = ' '.join(df[df['is_fake'] == 1]['Resume_clean'])
generate_wordcloud(real_resumes, 'Word Cloud for Real Resumes')
generate_wordcloud(fake_resumes, 'Word Cloud for Fake Resumes')

real_jobs = ' '.join(df[df['is_fake'] == 0]['Job_Description_clean'])
fake_jobs = ' '.join(df[df['is_fake'] == 1]['Job_Description_clean'])
generate_wordcloud(real_jobs, 'Word Cloud for Real Job Descriptions')
generate_wordcloud(fake_jobs, 'Word Cloud for Fake Job Descriptions')

real_reasons = ' '.join(df[df['is_fake'] == 0]['Reason_for_decision_clean'])
fake_reasons = ' '.join(df[df['is_fake'] == 1]['Reason_for_decision_clean'])
generate_wordcloud(real_reasons, 'Word Cloud for Real Reasons for Decision')
generate_wordcloud(fake_reasons, 'Word Cloud for Fake Reasons for Decision')

# Step 13: Print diagnostics
print("is_fake distribution:", df['is_fake'].value_counts().to_dict())
print("Decision distribution:", df['Decision'].value_counts().to_dict())
print("Top 10 Roles:", df['Role'].value_counts().head(10).to_dict())
print("Top 10 Reasons:",
df['Reason_for_decision'].value_counts().head(10).to_dict())
```

```
print("is_fake test distribution:",
pd.Series(y_fake_test).value_counts().to_dict())
if len(np.unique(y_dec_test)) > 0:
    print("Decision test distribution:",
pd.Series(y_dec_test).value_counts().to_dict())
else:
    print("Decision test distribution: Empty due to missing classes")
```

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /usr/share/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
Original Decision distribution: {'reject': 5114, 'select': 5060}
Mapped Decision distribution: {0: 5114, 1: 5060}
Epoch 1/20
```

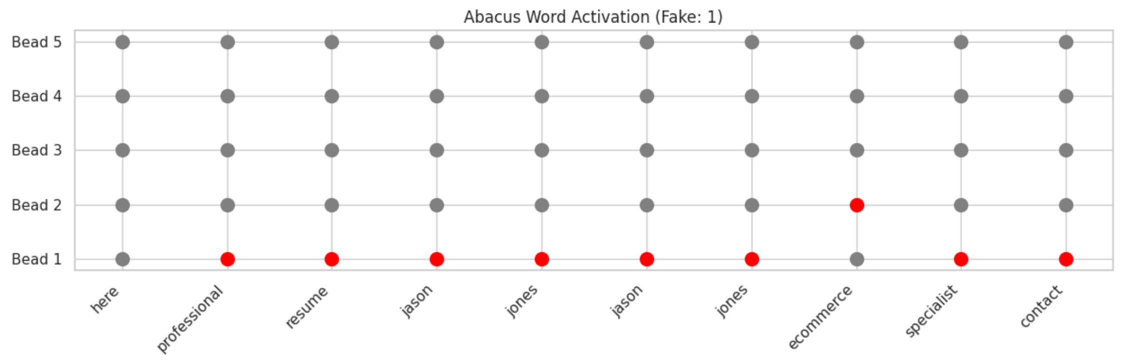
```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.
When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(**kwargs)
```

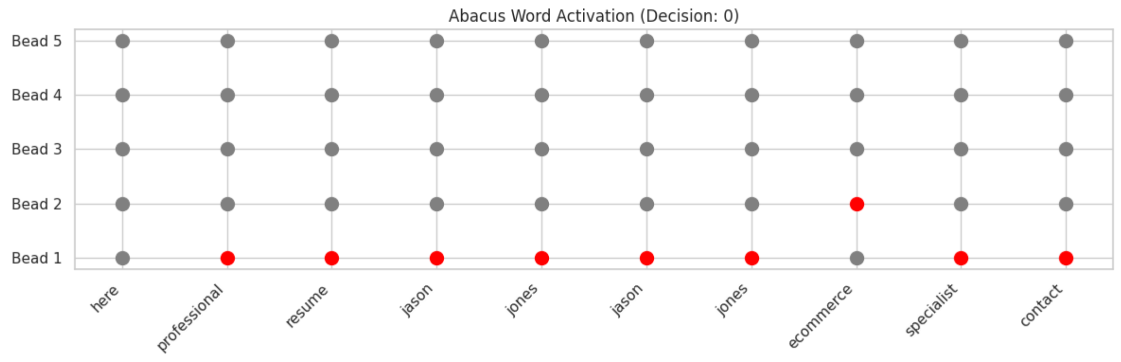
```
702/702 ━━━━━━━━━━━━━━━━━━━━ 52s 66ms/step - accuracy: 0.5519 - loss: 0.6841
- val_accuracy: 0.5743 - val_loss: 0.6796
Epoch 2/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 47s 67ms/step - accuracy: 0.5776 - loss: 0.6778
- val_accuracy: 0.6177 - val_loss: 0.6603
Epoch 3/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 47s 66ms/step - accuracy: 0.6342 - loss: 0.6529
- val_accuracy: 0.6517 - val_loss: 0.6328
Epoch 4/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 47s 66ms/step - accuracy: 0.6315 - loss: 0.6488
- val_accuracy: 0.6405 - val_loss: 0.6281
Epoch 5/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 47s 67ms/step - accuracy: 0.7123 - loss: 0.5661
- val_accuracy: 0.7660 - val_loss: 0.4872
Epoch 6/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 66ms/step - accuracy: 0.7649 - loss: 0.4913
- val_accuracy: 0.7876 - val_loss: 0.4657
Epoch 7/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 47s 67ms/step - accuracy: 0.7786 - loss: 0.4799
- val_accuracy: 0.7933 - val_loss: 0.4554
Epoch 8/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 47s 66ms/step - accuracy: 0.7807 - loss: 0.4681
- val_accuracy: 0.7853 - val_loss: 0.4709
Epoch 9/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 66ms/step - accuracy: 0.7784 - loss: 0.4687
- val_accuracy: 0.7936 - val_loss: 0.4449
```

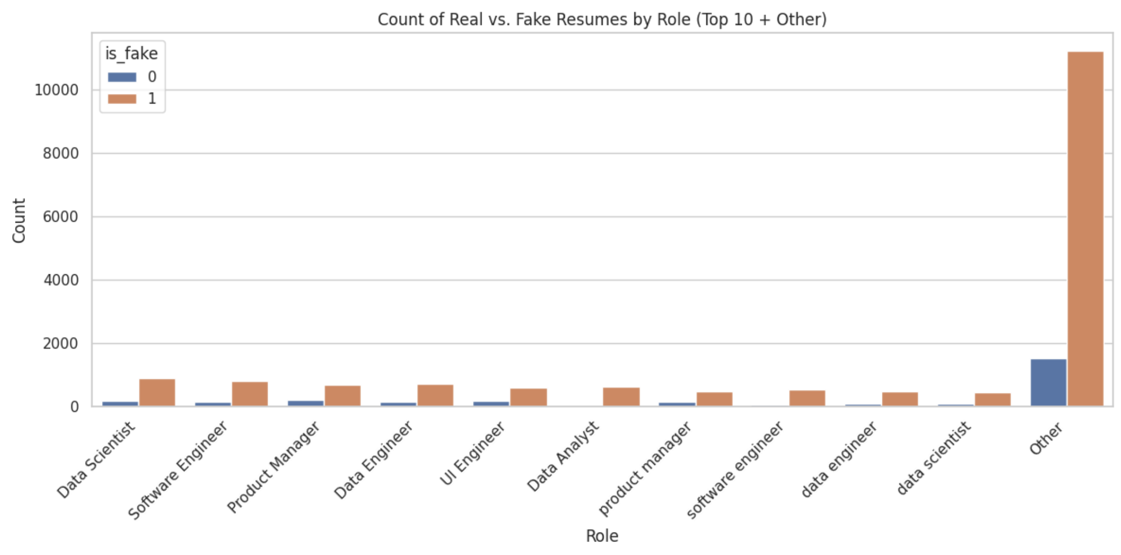
```
Epoch 10/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 65ms/step - accuracy: 0.7895 - loss: 0.4556
- val_accuracy: 0.8079 - val_loss: 0.4289
Epoch 11/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 65ms/step - accuracy: 0.7968 - loss: 0.4417
- val_accuracy: 0.8034 - val_loss: 0.4281
Epoch 12/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 66ms/step - accuracy: 0.8037 - loss: 0.4349
- val_accuracy: 0.8134 - val_loss: 0.4186
Epoch 13/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 65ms/step - accuracy: 0.8003 - loss: 0.4462
- val_accuracy: 0.8212 - val_loss: 0.4068
Epoch 14/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 65ms/step - accuracy: 0.8117 - loss: 0.4169
- val_accuracy: 0.8253 - val_loss: 0.4015
Epoch 15/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 65ms/step - accuracy: 0.8130 - loss: 0.4140
- val_accuracy: 0.8264 - val_loss: 0.3991
Epoch 16/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 45s 65ms/step - accuracy: 0.8201 - loss: 0.4053
- val_accuracy: 0.8203 - val_loss: 0.4010
Epoch 17/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 65ms/step - accuracy: 0.8243 - loss: 0.3977
- val_accuracy: 0.8262 - val_loss: 0.3906
Epoch 18/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 45s 65ms/step - accuracy: 0.8304 - loss: 0.3911
- val_accuracy: 0.8244 - val_loss: 0.3916
Epoch 19/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 46s 65ms/step - accuracy: 0.8307 - loss: 0.3828
- val_accuracy: 0.8321 - val_loss: 0.3760
Epoch 20/20
702/702 ━━━━━━━━━━━━━━━━━━━━ 45s 65ms/step - accuracy: 0.8338 - loss: 0.3769
- val_accuracy: 0.8283 - val_loss: 0.3808
Epoch 1/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 45s 66ms/step - accuracy: 0.5014 - loss: 0.6943
- val_accuracy: 0.5094 - val_loss: 0.6932
Epoch 2/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 40s 65ms/step - accuracy: 0.5062 - loss: 0.6933
- val_accuracy: 0.5061 - val_loss: 0.6931
Epoch 3/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.5059 - loss: 0.6932
- val_accuracy: 0.5061 - val_loss: 0.6930
Epoch 4/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.5044 - loss: 0.6931
- val_accuracy: 0.5123 - val_loss: 0.6930
Epoch 5/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.5072 - loss: 0.6933
- val_accuracy: 0.5102 - val_loss: 0.6925
Epoch 6/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 40s 65ms/step - accuracy: 0.5137 - loss: 0.6928
```

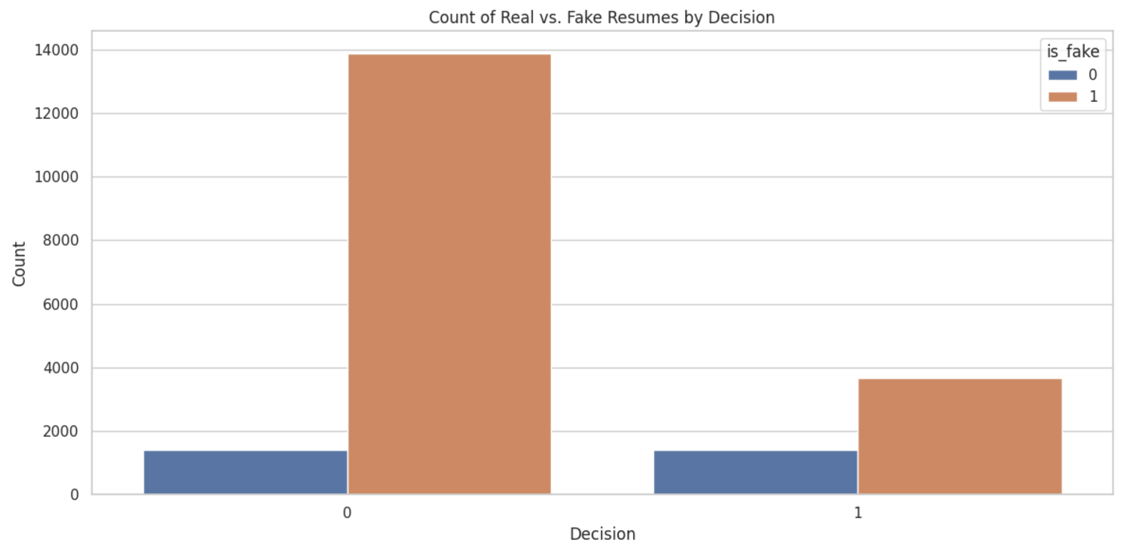
```
- val_accuracy: 0.5117 - val_loss: 0.6926
Epoch 7/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.5802 - loss: 0.6595
- val_accuracy: 0.7500 - val_loss: 0.5571
Epoch 8/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.7500 - loss: 0.5597
- val_accuracy: 0.7537 - val_loss: 0.5502
Epoch 9/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.7550 - loss: 0.5505
- val_accuracy: 0.7553 - val_loss: 0.5420
Epoch 10/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.7587 - loss: 0.5434
- val_accuracy: 0.7535 - val_loss: 0.5425
Epoch 11/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 40s 65ms/step - accuracy: 0.7575 - loss: 0.5398
- val_accuracy: 0.7533 - val_loss: 0.5409
Epoch 12/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.7573 - loss: 0.5392
- val_accuracy: 0.7545 - val_loss: 0.5440
Epoch 13/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.7581 - loss: 0.5372
- val_accuracy: 0.7520 - val_loss: 0.5451
Epoch 14/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 39s 64ms/step - accuracy: 0.7573 - loss: 0.5358
- val_accuracy: 0.7565 - val_loss: 0.5394
Epoch 15/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 40s 65ms/step - accuracy: 0.7587 - loss: 0.5346
- val_accuracy: 0.7510 - val_loss: 0.5434
Epoch 16/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 40s 65ms/step - accuracy: 0.7595 - loss: 0.5333
- val_accuracy: 0.7531 - val_loss: 0.5441
Epoch 17/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 41s 67ms/step - accuracy: 0.7593 - loss: 0.5312
- val_accuracy: 0.7522 - val_loss: 0.5455
Epoch 18/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 41s 67ms/step - accuracy: 0.7602 - loss: 0.5294
- val_accuracy: 0.7543 - val_loss: 0.5440
Epoch 19/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 40s 66ms/step - accuracy: 0.7598 - loss: 0.5287
- val_accuracy: 0.7543 - val_loss: 0.5407
Epoch 20/20
612/612 ━━━━━━━━━━━━━━━━━━━━ 40s 66ms/step - accuracy: 0.7593 - loss: 0.5276
- val_accuracy: 0.7555 - val_loss: 0.5385
220/220 ━━━━━━━━━━━━━━━━━━━━ 6s 24ms/step
Fake Resume Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.87      0.84      3510
           1       0.86      0.80      0.83      3510
```

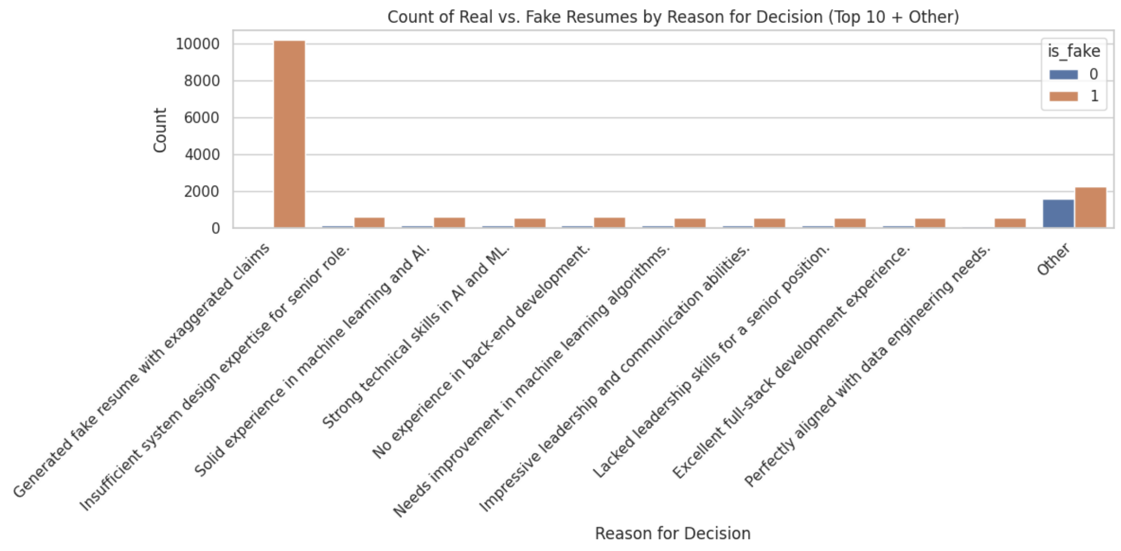
```
    accuracy                           0.83      7020
   macro avg       0.83      0.83      0.83      7020
weighted avg       0.83      0.83      0.83      7020

220/220 ━━━━━━━━━━━━━━━━━━━━ 5s 22ms/step
ROC-AUC (Fake): 0.9111811186597512
192/192 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step
Selection/Rejection Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.66      0.74      3058
           1       0.72      0.88      0.79      3057

    accuracy                           0.77      6115
   macro avg       0.78      0.77      0.76      6115
weighted avg       0.78      0.77      0.76      6115

192/192 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step
ROC-AUC (Decision): 0.8079580407402155
```

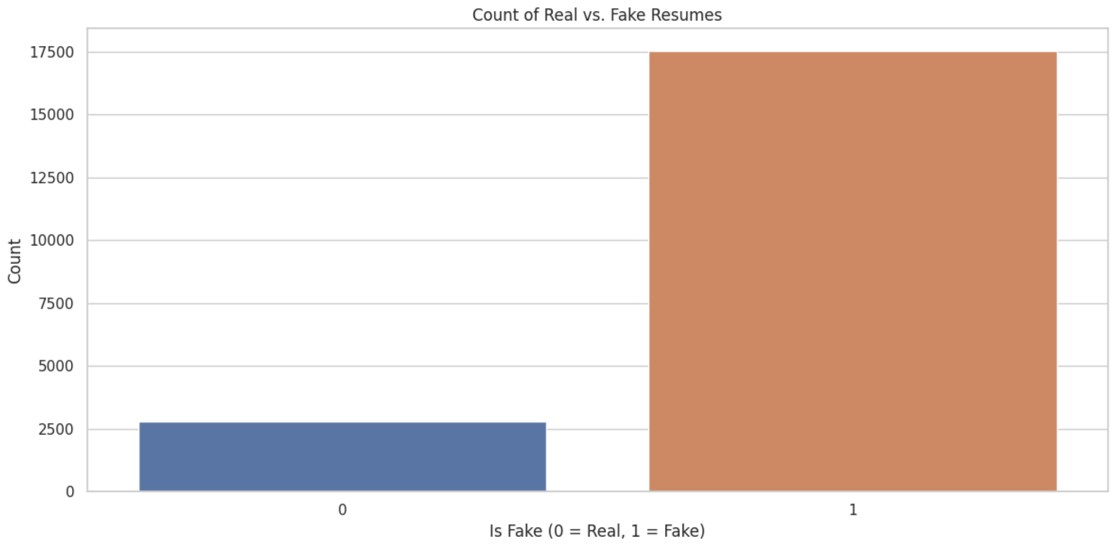


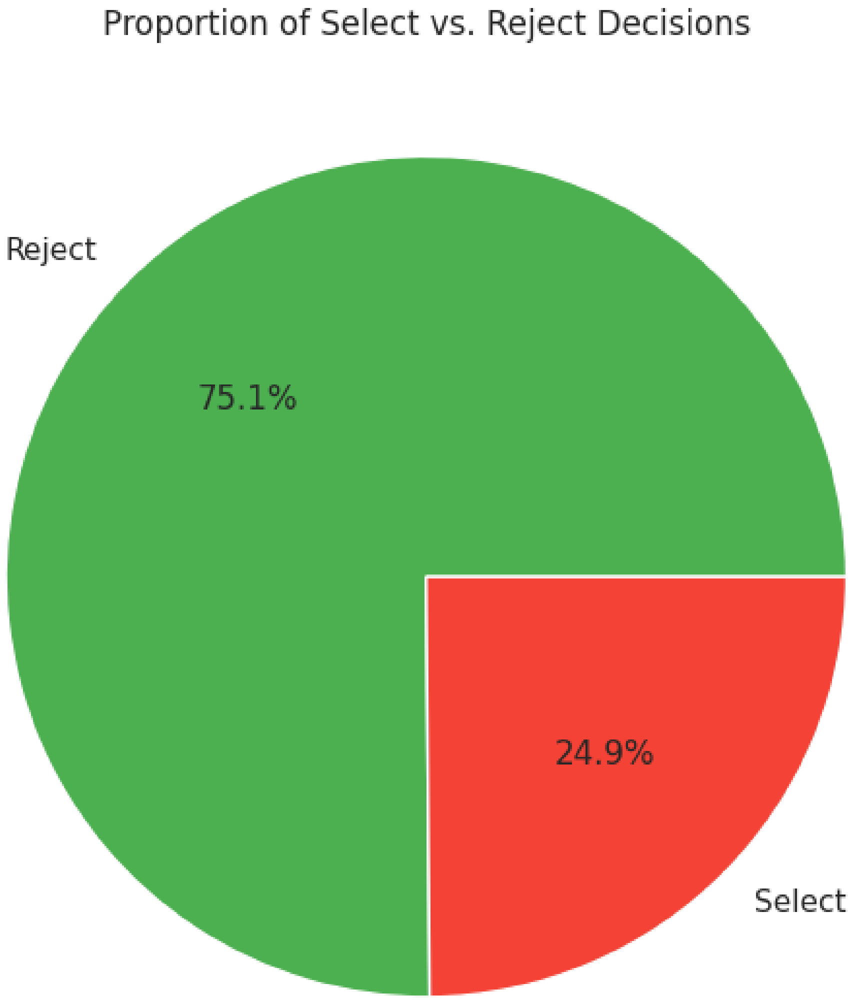


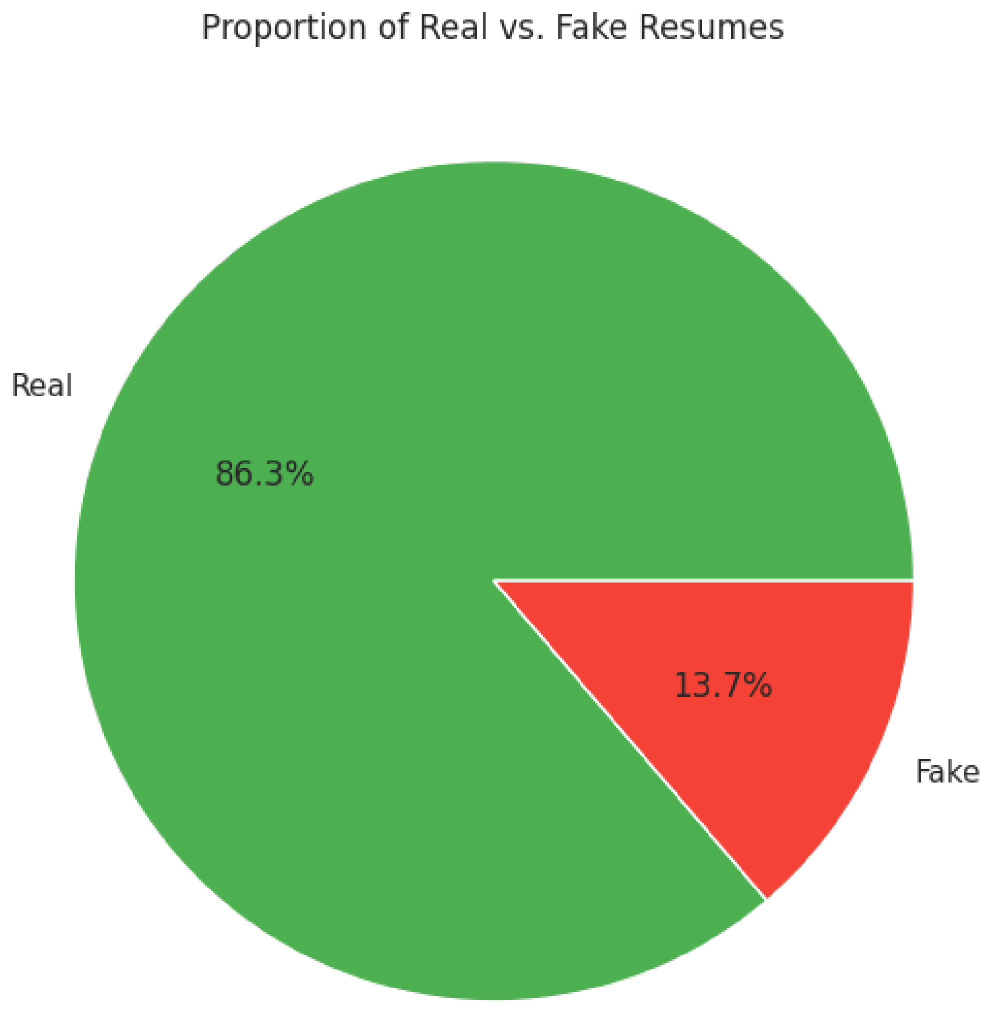

Count of Real vs. Fake Resumes by Role (Top 10 + Other)
is_fake
0
1
Count
Role
Data Scientist
Software Engineer
Product Manager
Data Engineer
UI Engineer
Data Analyst
product manager
software engineer
data engineer
data scientist
Other

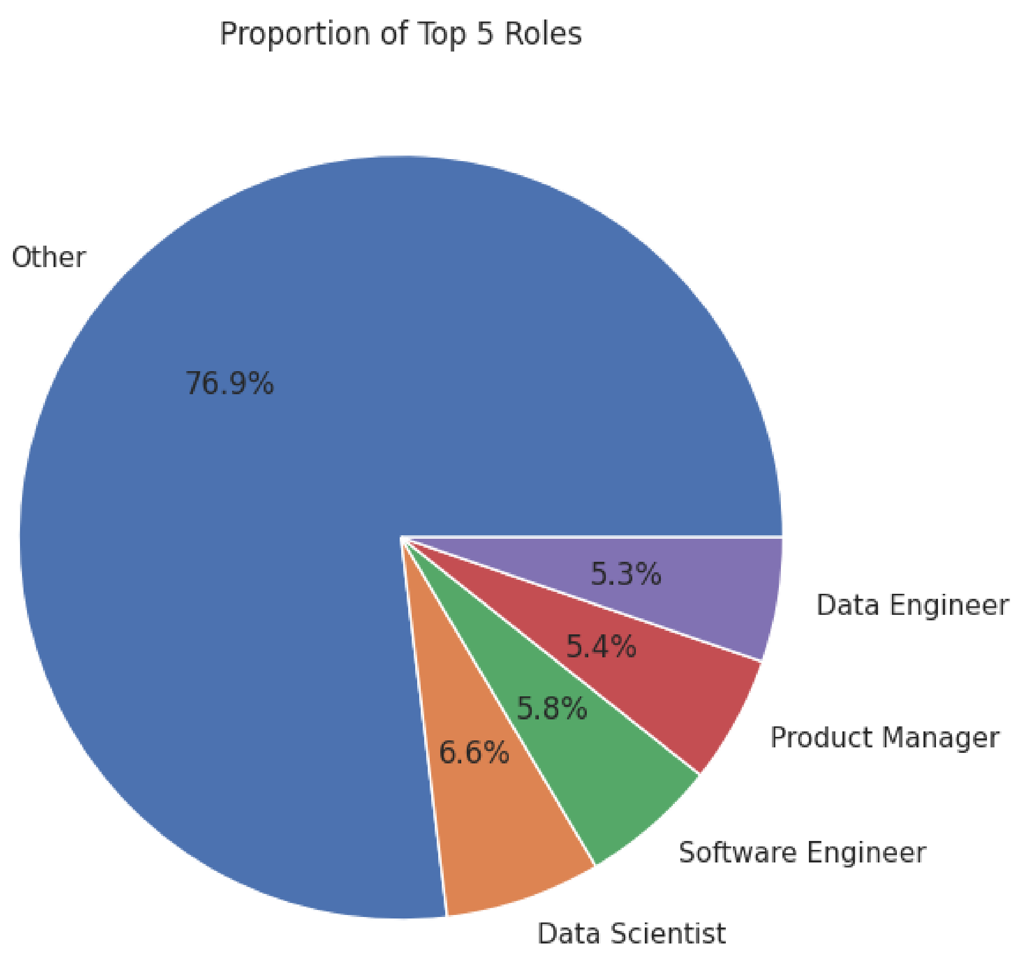

Count of Real vs. Fake Resumes by Decision
is_fake
0
1
Count
Decision

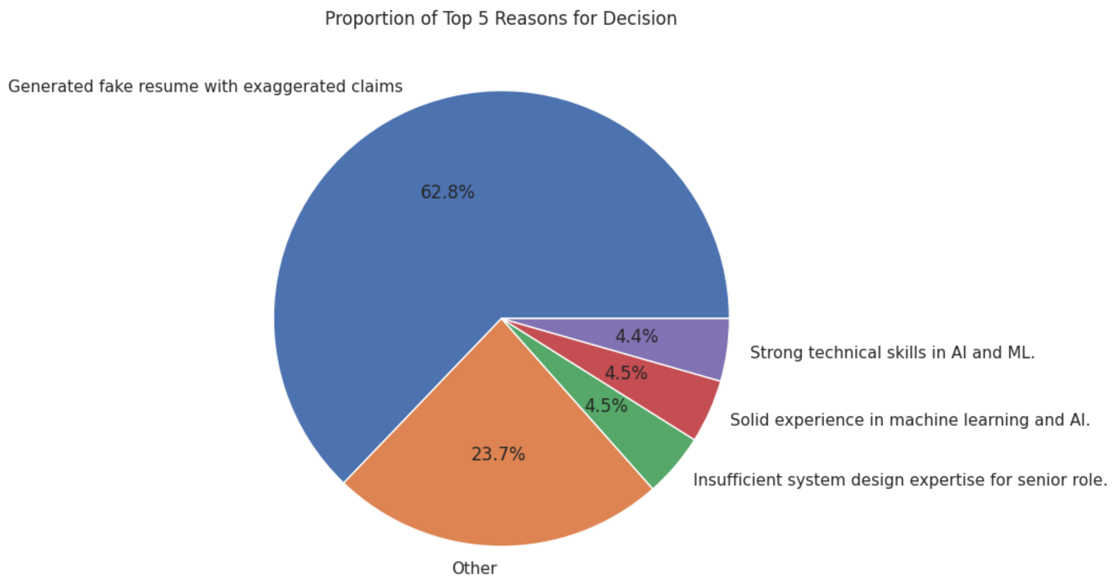

Count of Real vs. Fake Resumes by Reason for Decision (Top 10 + Other)
is_fake
0
1
Count
Reason for Decision
Generated fake resume with exaggerated claims
Insufficient system design expertise for senior role.
Solid experience in machine learning and AI.
Strong technical skills in AI and ML.
No experience in back-end development.
Needs improvement in machine learning algorithms.
Impressive leadership and communication abilities.
Lacked leadership skills for a senior position.
Excellent full-stack development experience.
Perfectly aligned with data engineering needs.
Other


Count of Real vs. Fake Resumes
Count
17500
15000
12500
10000
7500
5000
2500
0
0
1
Is Fake (0 = Real, 1 = Fake)

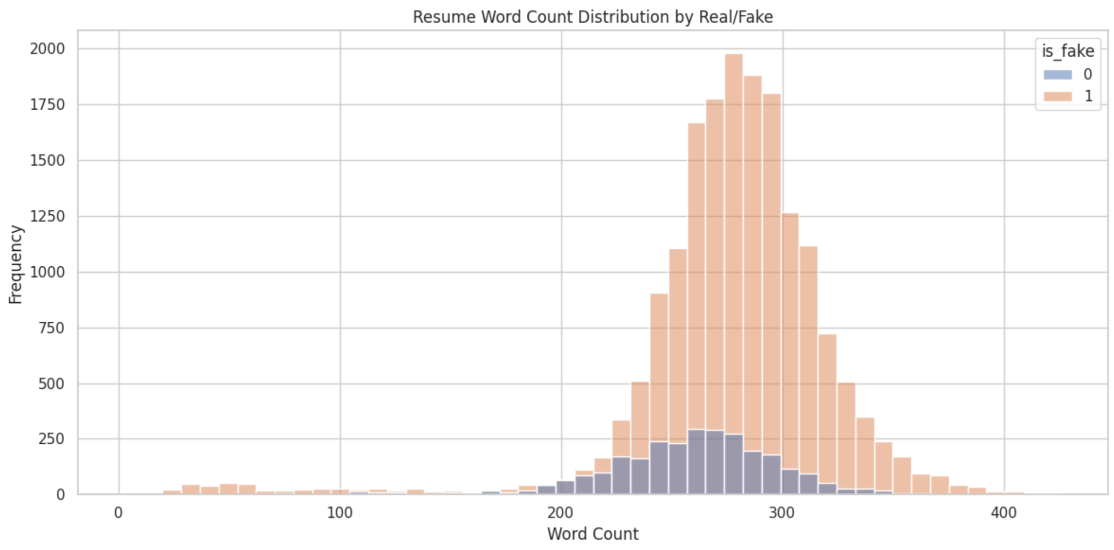

Proportion of Select vs. Reject Decisions
Reject
75.1%
24.9%
Select


Proportion of Real vs. Fake Resumes
Real
86.3%
13.7%
Fake


Proportion of Top 5 Roles
Other
76.9%
5.3%
Data Engineer
5.4%
Product Manager
5.8%
Software Engineer
6.6%
Data Scientist

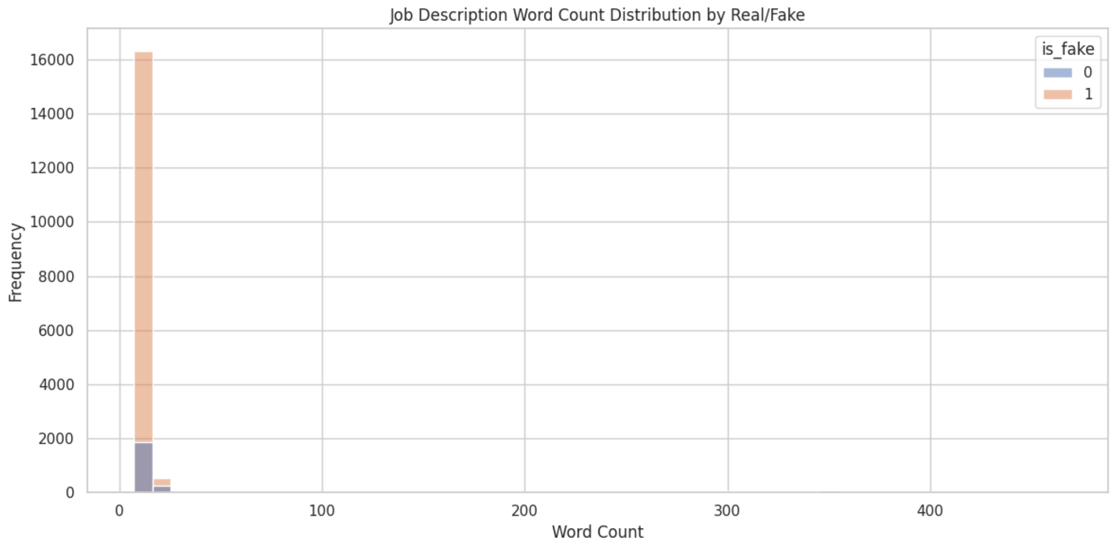

Proportion of Top 5 Reasons for Decision
Generated fake resume with exaggerated claims
62.8%
4.4%
Strong technical skills in AI and ML.
4.5%
Solid experience in machine learning and AI.
4.5%
Insufficient system design expertise for senior role.
23.7%
Other

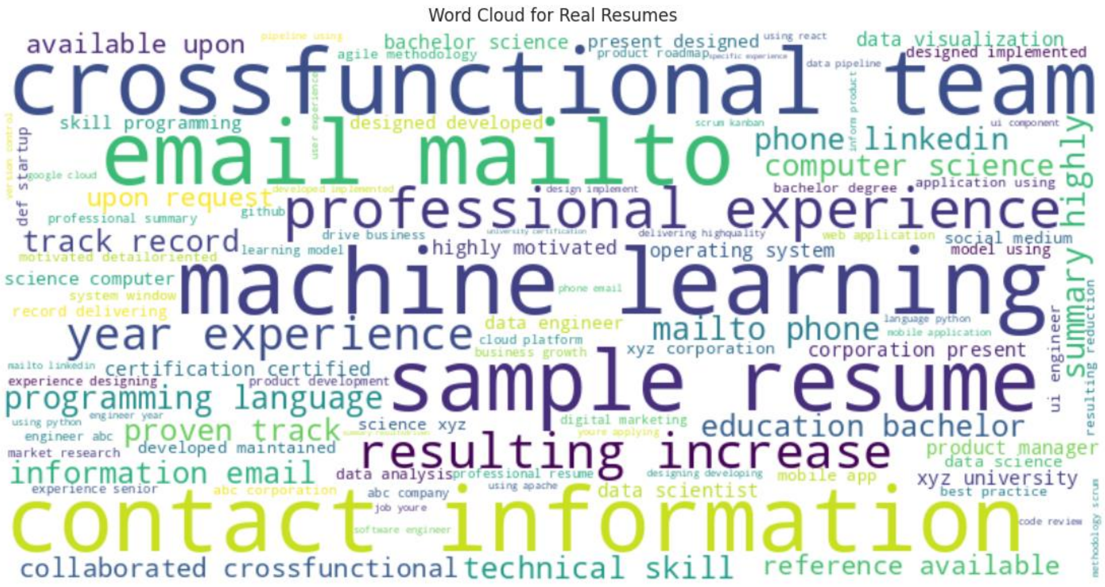
```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need to pass
a length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need to pass
a length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need to pass
a length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```



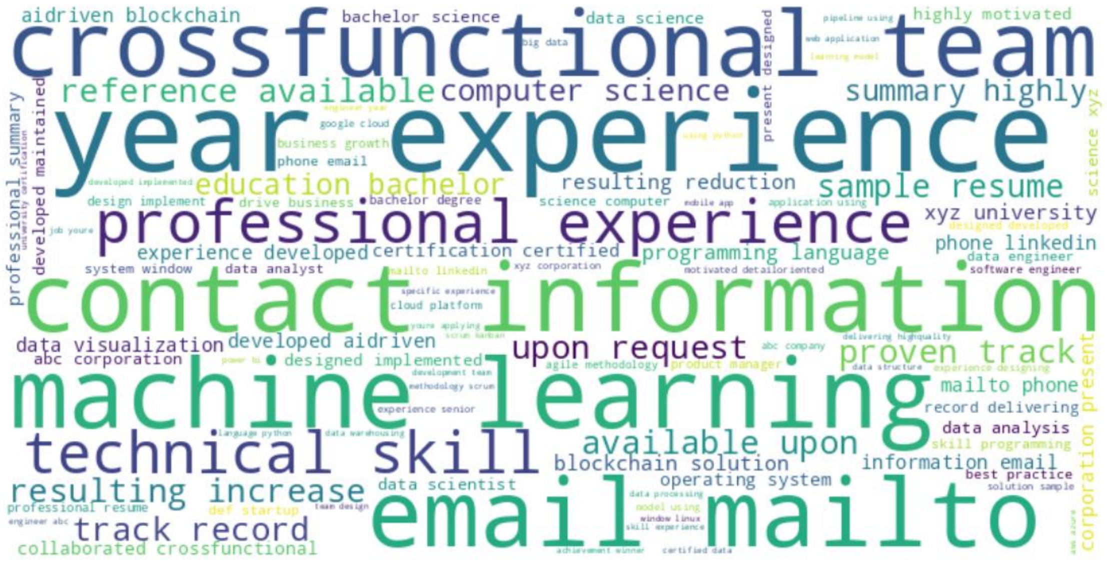
```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need to pass
a length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need to pass
a length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
```

```
  data_subset = grouped_data.get_group(pd_key)
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1075:
FutureWarning: When grouping with a length-1 list-like, you will need to pass
a length-1 tuple to get_group in a future version of pandas. Pass `(name,)`
instead of `name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```

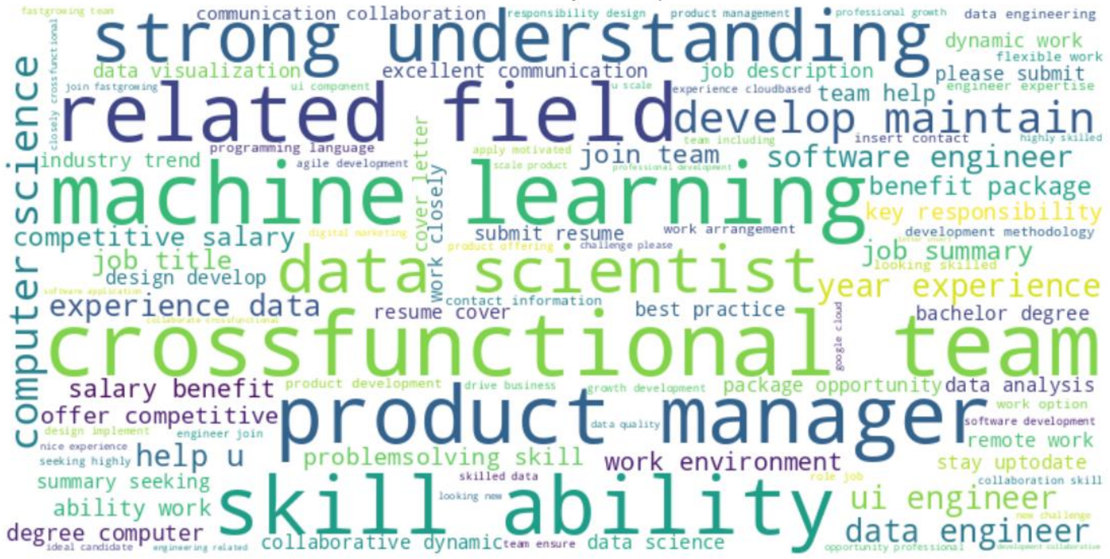




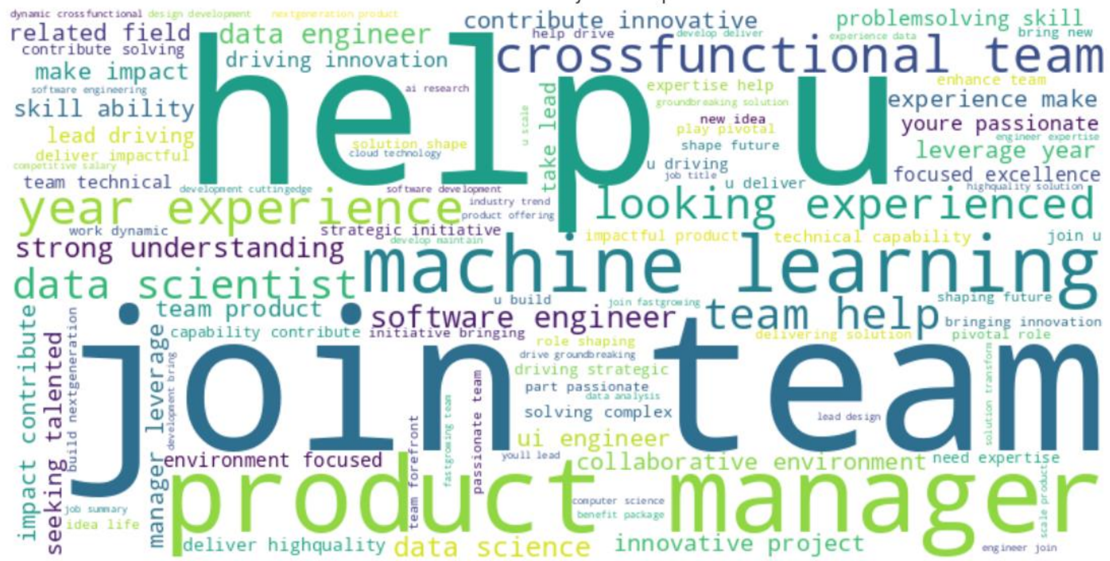
Word Cloud for Fake Resumes



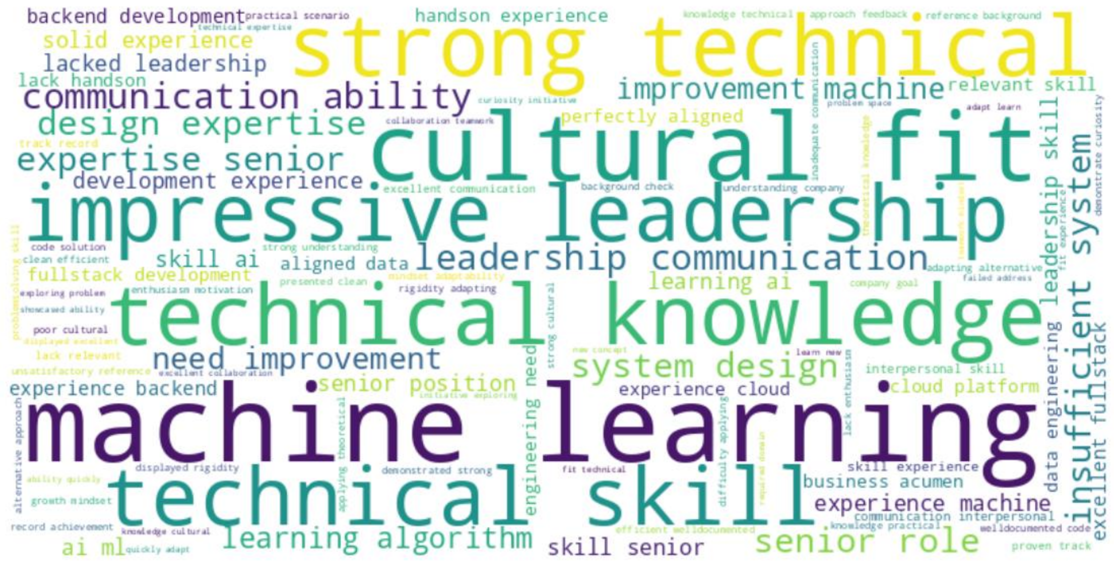
Word Cloud for Real Job Descriptions



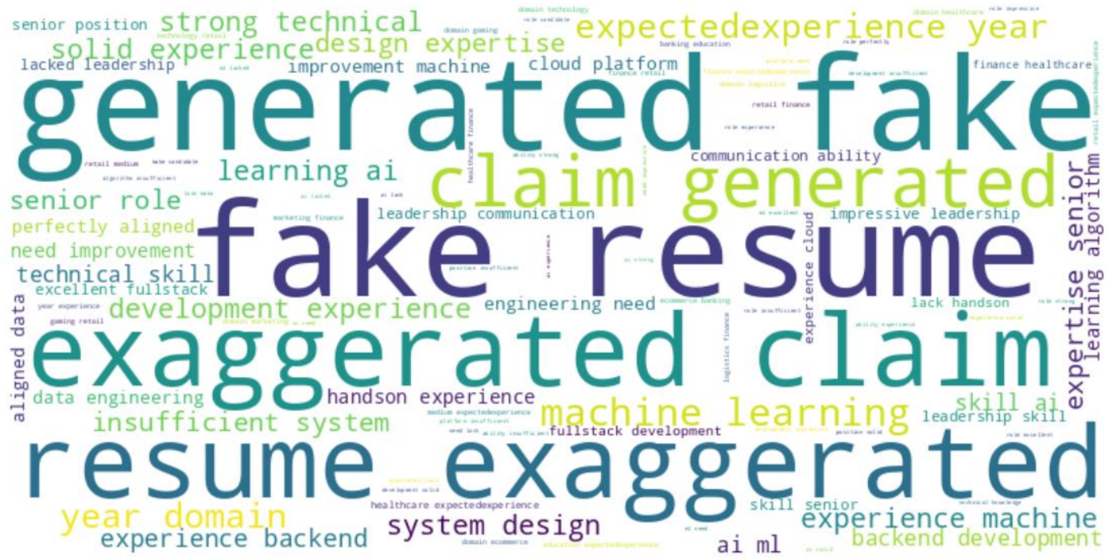
Word Cloud for Fake Job Descriptions



Word Cloud for Real Reasons for Decision



Word Cloud for Fake Reasons for Decision



```
is_fake distribution: {1: 17549, 0: 2797}
Decision distribution: {0: 15286, 1: 5060}
Top 10 Roles: {'Data Scientist': 1090, 'Software Engineer': 962, 'Product
Manager': 894, 'Data Engineer': 871, 'UI Engineer': 774, 'Data Analyst': 638,
'product manager': 628, 'software engineer': 612, 'data engineer': 584, 'data
scientist': 561}
Top 10 Reasons: {'Generated fake resume with exaggerated claims': 10172,
'Insufficient system design expertise for senior role.': 730, 'Solid
experience in machine learning and AI.': 728, 'Strong technical skills in AI
and ML.': 718, 'No experience in back-end development.': 717, 'Needs
improvement in machine learning algorithms.': 707, 'Impressive leadership and
communication abilities.': 704, 'Lacked leadership skills for a senior
position.': 685, 'Excellent full-stack development experience.': 677,
'Perfectly aligned with data engineering needs.': 668}
is_fake test distribution: {1: 3510, 0: 3510}
Decision test distribution: {0: 3058, 1: 3057}
```