```python
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
import seaborn as sns
import time
import warnings
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, LSTM, Embedding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
import tensorflow as tf

# Suppress warnings
warnings.filterwarnings('ignore')

# Download required NLTK packages
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)

# Function to clean text
def clean_text(text):
    """Clean the text by removing special characters, numbers, and extra whitespace."""
    if isinstance(text, str):
        # Convert to lowercase
        text = text.lower()
        # Remove special characters and numbers
        text = re.sub(r'[^a-zA-Z\s]', '', text)
        # Remove extra whitespace
        text = re.sub(r'\s+', ' ', text).strip()
        return text
    return ""

# TF-IDF with RNN (LSTM)
def tfidf_rnn(X_train_text, X_test_text, y_train, y_test):
    print("\n=== TF-IDF with RNN ===")
    start_time = time.time()

    # Extract TF-IDF features
    tfidf_vectorizer = TfidfVectorizer(max_features=5000)
    X_train = tfidf_vectorizer.fit_transform(X_train_text).toarray()
    X_test = tfidf_vectorizer.transform(X_test_text).toarray()

    # Reshape for RNN (samples, timesteps, features)
    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

    # Build RNN model
    model = Sequential([
        LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True),
        Dropout(0.3),
        LSTM(64),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

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

    # Early stopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Train model
    history = model.fit(
        X_train, y_train,
        epochs=15,
        batch_size=64,
        validation_split=0.1,
        callbacks=[early_stopping],
        verbose=1
    )
```

```
    # Evaluate model
    y_pred_prob = model.predict(X_test)
    y_pred = (y_pred_prob > 0.5).astype(int)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    cm = confusion_matrix(y_test, y_pred)

    # Print results
    print(f"Accuracy: {accuracy:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(cm)

    # Training time
    train_time = time.time() - start_time
    print(f"Training time: {train_time:.2f} seconds")

    # Plot training history
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('TF-IDF RNN Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('TF-IDF RNN Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.tight_layout()
    plt.savefig('tfidf_rnn_history.png')

    return {
        'accuracy': accuracy,
        'report': report,
        'confusion_matrix': cm,
        'train_time': train_time,
        'model': model,
        'history': history.history
    }

# Word2Vec with RNN (LSTM)
def word2vec_rnn(X_train_text, X_test_text, y_train, y_test):
    print("\n=== Word2Vec with RNN ===")
    start_time = time.time()

    # Tokenize text
    tokenizer = Tokenizer(num_words=10000)
    tokenizer.fit_on_texts(X_train_text)

    X_train_seq = tokenizer.texts_to_sequences(X_train_text)
    X_test_seq = tokenizer.texts_to_sequences(X_test_text)

    # Pad sequences
    max_len = 200
    X_train_pad = pad_sequences(X_train_seq, maxlen=max_len)
    X_test_pad = pad_sequences(X_test_seq, maxlen=max_len)

    # Vocabulary size
    vocab_size = len(tokenizer.word_index) + 1

    # Build RNN model with embedding layer
    model = Sequential([
        Embedding(vocab_size, 100, input_length=max_len),
        LSTM(128, return_sequences=True),
        Dropout(0.3),
        LSTM(64),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

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

```
    # Early stopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Train model
    history = model.fit(
        X_train_pad, y_train,
        epochs=15,
        batch_size=64,
        validation_split=0.1,
        callbacks=[early_stopping],
        verbose=1
    )

    # Evaluate model
    y_pred_prob = model.predict(X_test_pad)
    y_pred = (y_pred_prob > 0.5).astype(int)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    cm = confusion_matrix(y_test, y_pred)

    # Print results
    print(f"Accuracy: {accuracy:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(cm)

    # Training time
    train_time = time.time() - start_time
    print(f"Training time: {train_time:.2f} seconds")

    # Plot training history
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Word2Vec RNN Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Word2Vec RNN Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.tight_layout()
    plt.savefig('word2vec_rnn_history.png')

    return {
        'accuracy': accuracy,
        'report': report,
        'confusion_matrix': cm,
        'train_time': train_time,
        'model': model,
        'history': history.history
    }

# GloVe-like with RNN (LSTM)
def glove_rnn(X_train_text, X_test_text, y_train, y_test):
    print("\n=== GloVe-like with RNN ===")
    start_time = time.time()

    # Extract GloVe-like features using character n-grams
    tfidf_vectorizer = TfidfVectorizer(
        analyzer='char_wb',
        ngram_range=(2, 5),
        max_features=10000
    )
    X_train_tfidf = tfidf_vectorizer.fit_transform(X_train_text)
    X_test_tfidf = tfidf_vectorizer.transform(X_test_text)

    # Apply SVD to reduce dimensions
    svd = TruncatedSVD(n_components=100, random_state=42)
    X_train = svd.fit_transform(X_train_tfidf.toarray())
    X_test = svd.transform(X_test_tfidf.toarray())

    # Reshape for RNN (samples, timesteps, features)
```

```
    # Reshape for RNN (samples, timesteps, features)
    X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
    X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))

    # Build RNN model
    model = Sequential([
        LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=True),
        Dropout(0.3),
        LSTM(64),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

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

    # Early stopping
    early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

    # Train model
    history = model.fit(
        X_train, y_train,
        epochs=15,
        batch_size=64,
        validation_split=0.1,
        callbacks=[early_stopping],
        verbose=1
    )

    # Evaluate model
    y_pred_prob = model.predict(X_test)
    y_pred = (y_pred_prob > 0.5).astype(int)

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    cm = confusion_matrix(y_test, y_pred)

    # Print results
    print(f"Accuracy: {accuracy:.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred))
    print("Confusion Matrix:")
    print(cm)

    # Training time
    train_time = time.time() - start_time
    print(f"Training time: {train_time:.2f} seconds")

    # Plot training history
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('GloVe-like RNN Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('GloVe-like RNN Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')
    plt.tight_layout()
    plt.savefig('glove_rnn_history.png')

    return {
        'accuracy': accuracy,
        'report': report,
        'confusion_matrix': cm,
        'train_time': train_time,
        'model': model,
        'history': history.history
    }

# Main function
def main():
    """Main function to classify AI-generated vs. Human-generated text using RNN with different embeddings."""
    print("AI-Generated vs. Human-Generated Text Classification")
```

```
    print("=" * 50)

    # Set random seeds for reproducibility
    np.random.seed(42)
    tf.random.set_seed(42)

    # Load dataset from Kaggle
    print("Loading dataset...")
    # Assuming the dataset is downloaded and available locally
    df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/NLP_LAB/Training_Essay_Data (1).csv/Training_Essay_Data.csv")

    print(f"Dataset shape: {df.shape}")
    print(f"Column names: {df.columns.tolist()}")
    print(f"Label distribution: {df['generated'].value_counts().to_dict()}")

    # Clean and preprocess text
    print("Cleaning and preprocessing text...")
    df['clean_text'] = df['text'].apply(clean_text)

    # Split data
    print("Splitting data into train and test sets...")
    X_train_text, X_test_text, y_train, y_test = train_test_split(
        df['clean_text'], df['generated'], test_size=0.2, random_state=42, stratify=df['generated']
    )

    # Dictionary to store all results
    results = {}

    # Run all models
    try:
        # TF-IDF with RNN
        results['TF-IDF RNN'] = tfidf_rnn(X_train_text, X_test_text, y_train, y_test)

        # Word2Vec with RNN
        results['Word2Vec RNN'] = word2vec_rnn(X_train_text, X_test_text, y_train, y_test)

        # GloVe-like with RNN
        results['GloVe-like RNN'] = glove_rnn(X_train_text, X_test_text, y_train, y_test)
    except Exception as e:
        print(f"Error during model training: {e}")

    # Compare results
    print("\nComparison of all models:")
    print("=" * 50)

    comparison = {}
    for model_name, model_results in results.items():
        comparison[model_name] = {
            'accuracy': model_results['accuracy'],
            'f1_score': model_results['report']['weighted avg']['f1-score'],
            'train_time': model_results['train_time']
        }

    comparison_df = pd.DataFrame(comparison).T
    comparison_df = comparison_df.sort_values('accuracy', ascending=False)
    print(comparison_df)

    # Visualize comparison
    plt.figure(figsize=(10, 6))
    sns.barplot(x=comparison_df.index, y=comparison_df['accuracy'])
    plt.title('Accuracy Comparison of RNN Models with Different Embeddings')
    plt.ylabel('Accuracy')
    plt.ylim(0.6, 1.0)  # Set y-axis to start from 0.6 for better visualization
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig('model_comparison.png')

    print("\nBest performing model:")
    best_model = comparison_df.index[0]
    print(f"{best_model} with accuracy: {comparison_df.loc[best_model, 'accuracy']:.4f}")

    # Check if accuracy meets the 70% threshold
    if comparison_df['accuracy'].max() >= 0.70:
        print("\nAccuracy threshold of 70% achieved!")
    else:
        print("\nAccuracy threshold of 70% not achieved.")

    print("\nProgram completed successfully.")


if __name__ == "__main__":
    main()
```

```
AI-Generated vs. Human-Generated Text Classification
==================================================
Loading dataset...
Dataset shape: (29145, 2)
Column names: ['text', 'generated']
Label distribution: {0: 17508, 1: 11637}
Cleaning and preprocessing text...
Splitting data into train and test sets...

=== TF-IDF with RNN ===
Epoch 1/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - accuracy: 0.9020 - loss: 0.3073 - val_accuracy: 0.9936 - val_loss: 0.0194
Epoch 2/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 7s 8ms/step - accuracy: 0.9955 - loss: 0.0154 - val_accuracy: 0.9944 - val_loss: 0.0111
Epoch 3/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9982 - loss: 0.0044 - val_accuracy: 0.9957 - val_loss: 0.0134
Epoch 4/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - accuracy: 0.9989 - loss: 0.0029 - val_accuracy: 0.9953 - val_loss: 0.0154
Epoch 5/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 1.0000 - loss: 2.3898e-04 - val_accuracy: 0.9953 - val_loss: 0.0179
183/183 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
Accuracy: 0.9962
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3502
           1       0.99      1.00      1.00      2327

    accuracy                           1.00      5829
   macro avg       1.00      1.00      1.00      5829
weighted avg       1.00      1.00      1.00      5829

Confusion Matrix:
[[3490   12]
 [  10 2317]]
Training time: 44.59 seconds

=== Word2Vec with RNN ===
Epoch 1/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 13s 26ms/step - accuracy: 0.7716 - loss: 0.4884 - val_accuracy: 0.9605 - val_loss: 0.1171
Epoch 2/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 9s 25ms/step - accuracy: 0.9718 - loss: 0.0931 - val_accuracy: 0.9846 - val_loss: 0.0591
Epoch 3/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.9716 - loss: 0.0888 - val_accuracy: 0.9773 - val_loss: 0.0814
Epoch 4/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 8s 25ms/step - accuracy: 0.9859 - loss: 0.0556 - val_accuracy: 0.9867 - val_loss: 0.0422
Epoch 5/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 10s 24ms/step - accuracy: 0.9931 - loss: 0.0291 - val_accuracy: 0.9854 - val_loss: 0.0418
Epoch 6/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 8s 24ms/step - accuracy: 0.9933 - loss: 0.0234 - val_accuracy: 0.9708 - val_loss: 0.0992
Epoch 7/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - accuracy: 0.9938 - loss: 0.0206 - val_accuracy: 0.9850 - val_loss: 0.0429
Epoch 8/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - accuracy: 0.9924 - loss: 0.0232 - val_accuracy: 0.9893 - val_loss: 0.0373
Epoch 9/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - accuracy: 0.9966 - loss: 0.0122 - val_accuracy: 0.9850 - val_loss: 0.0675
Epoch 10/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 10s 23ms/step - accuracy: 0.9901 - loss: 0.0337 - val_accuracy: 0.9897 - val_loss: 0.0287
Epoch 11/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 8s 25ms/step - accuracy: 0.9980 - loss: 0.0081 - val_accuracy: 0.9884 - val_loss: 0.0501
Epoch 12/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 10s 25ms/step - accuracy: 0.9974 - loss: 0.0082 - val_accuracy: 0.9906 - val_loss: 0.0291
Epoch 13/15
328/328 ━━━━━━━━━━━━━━━━━━━━ 8s 23ms/step - accuracy: 0.9973 - loss: 0.0090 - val_accuracy: 0.9919 - val_loss: 0.0314
183/183 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
Accuracy: 0.9864
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      3502
           1       0.98      0.99      0.98      2327

    accuracy                           0.99      5829
   macro avg       0.98      0.99      0.99      5829
weighted avg       0.99      0.99      0.99      5829

Confusion Matrix:
[[3443   59]
 [  20 2307]]
Training time: 134.74 seconds
```