In [5]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

In [6]:
```python
# Load the dataset
file_path = "D:/Capstone_2025/code/preprocessed_data.csv"  # Update with your actual path
df = pd.read_csv(file_path)
df
```

Out[6]:

	Year	Month	Day	Hour	Clearsky DHI	Clearsky DNI	Temperature	Clearsky GHI	cloud fill flag	Cloud Type	...	Surface Albedo	Pressure	Precipitable Water	Win Directio
0	2006	1	1	0	0	0	0.108614	0	0	0	...	0.15	778	0.9	8
1	2006	1	1	1	0	0	0.093633	0	0	0	...	0.15	779	1.0	8
2	2006	1	1	2	0	0	0.082397	0	0	0	...	0.15	779	1.0	9
3	2006	1	1	3	0	0	0.112360	0	0	0	...	0.15	780	1.0	9
4	2006	1	1	4	36	484	0.254682	96	1	3	...	0.15	780	1.0	9
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
13431	2022	1	31	19	0	0	0.363296	0	0	1	...	0.14	779	1.5	12
13432	2022	1	31	20	0	0	0.359551	0	0	1	...	0.14	778	1.5	18
13433	2022	1	31	21	0	0	0.348315	0	0	0	...	0.14	777	1.5	21
13434	2022	1	31	22	0	0	0.329588	0	0	3	...	0.14	777	1.4	18
13435	2022	1	31	23	0	0	0.303371	0	0	0	...	0.14	777	1.4	16

13436 rows × 27 columns

In [7]:
```python
# Select relevant features
features = ['Temperature', 'Wind Speed', 'Surface Albedo', 'Cloud Type', 'Pressure', 'Precipitable Water']
target_ghi = 'Clearsky GHI'  # Target variable for prediction
target_dni = 'Clearsky DNI'
```

In [8]:
```python
# Drop missing values (if any)
df = df.dropna()
```

In [9]:
```
# Split data into features (X) and target (y)
X = df[features]
y_ghi = df[target_ghi]
y_dni = df[target_dni]

# Split into training and test sets
X_train, X_test, y_train_ghi, y_test_ghi = train_test_split(X, y_ghi, test_size=0.2, random_state=42)
X_train, X_test, y_train_dni, y_test_dni = train_test_split(X, y_dni, test_size=0.2, random_state=42)
```

In [10]:
```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Random Forest

In [11]:
```
# Train Random Forest Model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train_scaled, y_train_ghi)

# Predictions
y_pred_ghi_rf = rf_model.predict(X_test_scaled)

# Evaluate
print("Random Forest (GHI) - MAE:", mean_absolute_error(y_test_ghi, y_pred_ghi_rf))
print("Random Forest (GHI) - RMSE:", mean_squared_error(y_test_ghi, y_pred_ghi_rf, squared=False))
print("Random Forest (GHI) - R2 Score:", r2_score(y_test_ghi, y_pred_ghi_rf))
```

```
Random Forest (GHI) - MAE: 61.59804538690477
Random Forest (GHI) - RMSE: 127.70602041251989
Random Forest (GHI) - R2 Score: 0.8365030532250054

C:\Users\reshm\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: FutureWarning: 'squared' is d
eprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the funct
ion'root_mean_squared_error'.
  warnings.warn(
```

XGBOOST Model

In [12]:
```
# Train XGBoost Model
xgb_model = xgb.XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_model.fit(X_train_scaled, y_train_ghi)

# Predictions
y_pred_ghi_xgb = xgb_model.predict(X_test_scaled)

# Evaluate
print("XGBoost (GHI) - MAE:", mean_absolute_error(y_test_ghi, y_pred_ghi_xgb))
print("XGBoost (GHI) - RMSE:", mean_squared_error(y_test_ghi, y_pred_ghi_xgb, squared=False))
print("XGBoost (GHI) - R2 Score:", r2_score(y_test_ghi, y_pred_ghi_xgb))
```

```
XGBoost (GHI) - MAE: 61.6754375093927
XGBoost (GHI) - RMSE: 126.01739455921303
XGBoost (GHI) - R2 Score: 0.8407982587814331

C:\Users\reshm\anaconda3\Lib\site-packages\sklearn\metrics\_regression.py:492: FutureWarning: 'squared' is d
eprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the funct
ion'root_mean_squared_error'.
  warnings.warn(
```

Train LSTM Model

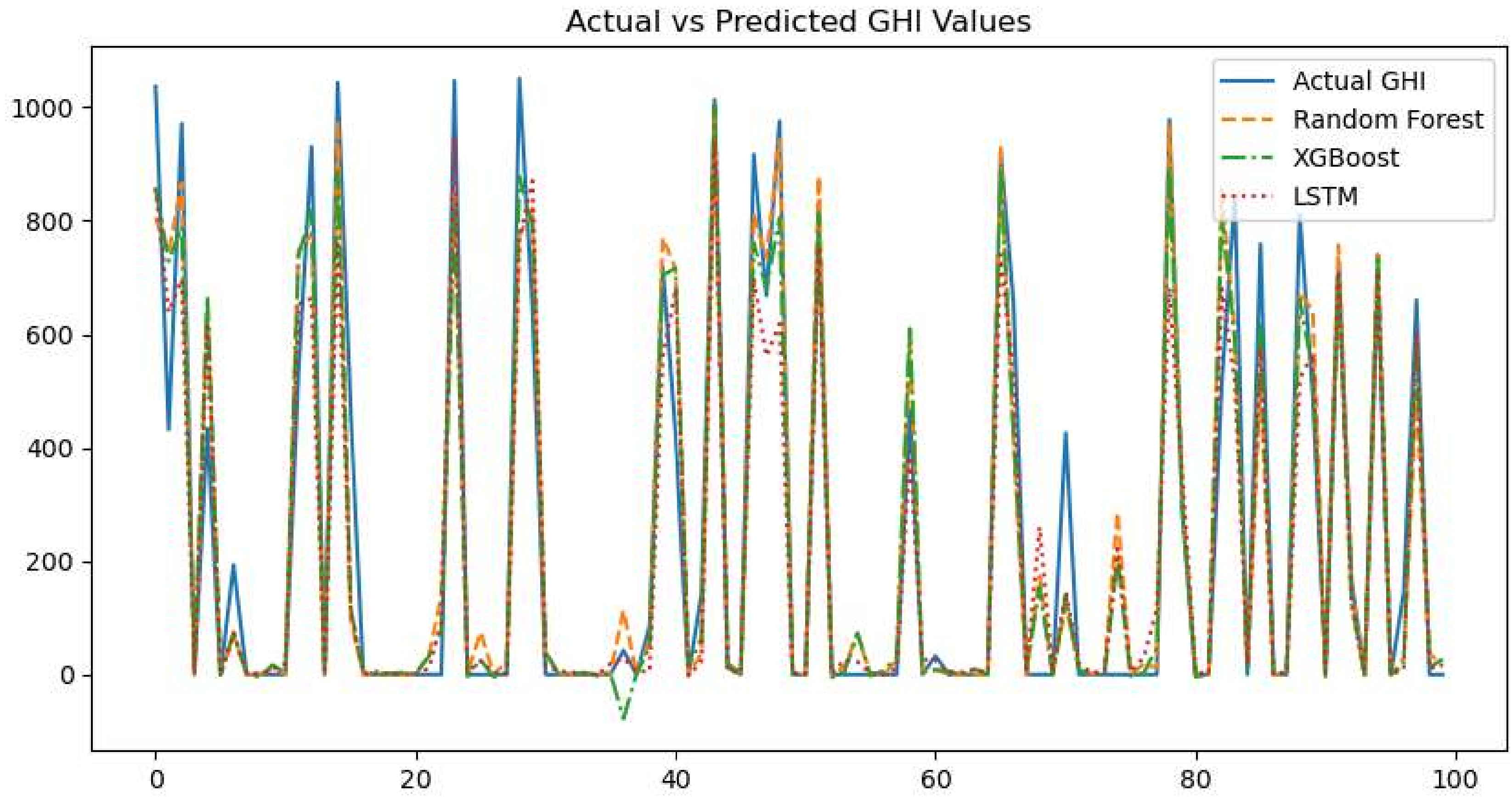
In [13]:
```
# Reshape input for LSTM (samples, time steps, features)
X_train_lstm = np.reshape(X_train_scaled, (X_train_scaled.shape[0], 1, X_train_scaled.shape[1]))
X_test_lstm = np.reshape(X_test_scaled, (X_test_scaled.shape[0], 1, X_test_scaled.shape[1]))
```

In [14]:
```python
# Define LSTM model
lstm_model = Sequential([
    LSTM(50, activation='relu', return_sequences=True, input_shape=(1, X_train_scaled.shape[1])),
    Dropout(0.2),
    LSTM(50, activation='relu'),
    Dropout(0.2),
    Dense(1)
])

# Compile model
lstm_model.compile(optimizer='adam', loss='mse')

# Train LSTM
lstm_model.fit(X_train_lstm, y_train_ghi, epochs=50, batch_size=32, validation_data=(X_test_lstm, y_test_ghi)

# Predictions
y_pred_ghi_lstm = lstm_model.predict(X_test_lstm)

# Evaluate
print("LSTM (GHI) - MAE:", mean_absolute_error(y_test_ghi, y_pred_ghi_lstm))
print("LSTM (GHI) - RMSE:", mean_squared_error(y_test_ghi, y_pred_ghi_lstm, squared=False))
print("LSTM (GHI) - R2 Score:", r2_score(y_test_ghi, y_pred_ghi_lstm))
```

```
Epoch 14/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 20092.2871 - val_loss: 18428.0742
Epoch 15/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 21851.2500 - val_loss: 18503.1484
Epoch 16/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 21390.9102 - val_loss: 18281.2500
Epoch 17/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 21019.2031 - val_loss: 18206.8105
Epoch 18/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 20869.4238 - val_loss: 18067.9668
Epoch 19/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - loss: 20521.9688 - val_loss: 17963.3535
Epoch 20/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 20174.7598 - val_loss: 17947.5273
Epoch 21/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 20241.7480 - val_loss: 17823.5898
Epoch 22/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 21042.4844 - val_loss: 17831.1289
Epoch 23/50
336/336 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 20988.4980 - val_loss: 17615.2031
```

Visualize the prediction

In [15]:
```
# Plot actual vs predicted GHI values
plt.figure(figsize=(10,5))
plt.plot(y_test_ghi.values[:100], label="Actual GHI", linestyle="-")
plt.plot(y_pred_ghi_rf[:100], label="Random Forest", linestyle="--")
plt.plot(y_pred_ghi_xgb[:100], label="XGBoost", linestyle="-.")
plt.plot(y_pred_ghi_lstm[:100], label="LSTM", linestyle=":")
plt.legend()
plt.title("Actual vs Predicted GHI Values")
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: