User Guide: Running the Conductance Prediction Model Locally.

Summary:

This script provides an automated pipeline for predicting conductance in twisted bilayer graphene systems (50x50nm) using machine learning. The process integrates data normalization, clustering with a Self-Organizing Map (SOM), and regression with Gradient Boosting Regressors (GBR).

The workflow ensures that any new input follows the same transformation pipeline as the training data, maintaining consistency in predictions. The key steps include downloading pre-trained models and scalers, clustering new input data based on learned patterns, selecting the appropriate GBR model for the identified cluster, and producing a final conductance prediction. This version includes predictions for all angles present in the training dataset, allowing for a comprehensive modeling of conductance behavior across different TBG configurations.

By structuring the workflow in an automated and modular way, this approach significantly reduces computational costs compared to traditional simulations while preserving accuracy in conductance predictions.

1. Install Required Dependencies

- Before running the script, ensure you have the required Python packages installed. This will ensure that the correct versions of each package are installed, maintaining compatibility with the script. You can do this by running the following command in terminal:

pip install numpy==2.0.0 joblib==1.4.2 scikit-learn==1.6.0 minisom==2.3.3 pandas==2.2.2

2. Download Required Files

- Users need to download the trained models and scalers from Google Drive. The script will handle this automatically.

- Download these User Guide from Github to use the links bellow.

Required Files:

File Name	Google Drive Link
minisom_clusterizer.pkl	Download
scaler_X_standard.pkl	Download
scaler_X_minmax.pkl	Download
scaler_y_standard.pkl	Download
scaler_y_minmax.pkl	Download

GBR Models for Each Cluster	Google Drive Link
gbr_model_cluster_0.pkl	Download
gbr_model_cluster_1.pkl	Download
gbr_model_cluster_2.pkl	Download
gbr_model_cluster_3.pkl	Download
gbr_model_cluster_4.pkl	Download
gbr_model_cluster_5.pkl	Download
gbr_model_cluster_6.pkl	Download
gbr_model_cluster_7.pkl	Download
gbr_model_cluster_8.pkl	Download

- After downloading, place all the files in the **same folder** as the Python script, users should name the folder as **"Predicted_Conductance".**

3. Define Output Folder

- In the script, user's can set the directory where the output files will be saved.

4. Making Predictions with a New Input

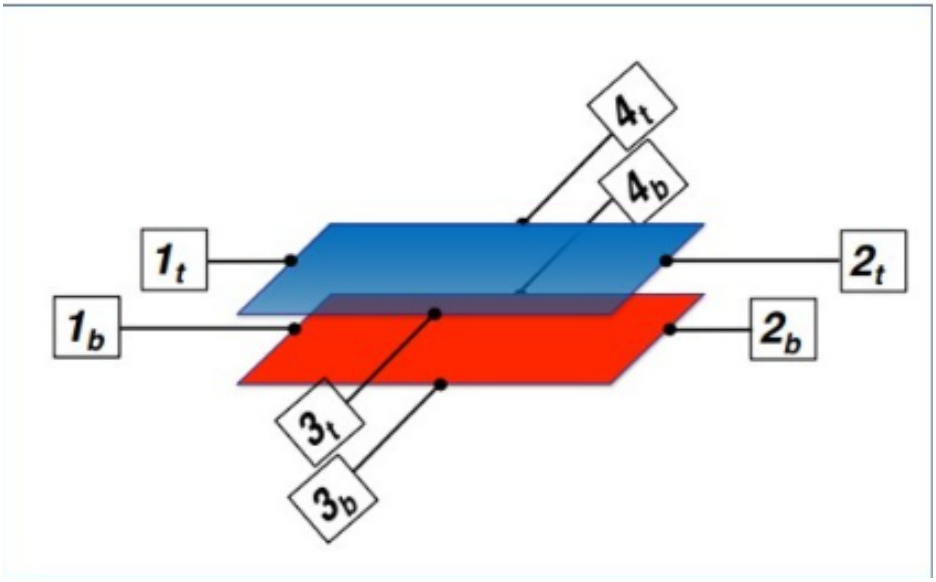
- After running the code, users can input a new angle configuration to predict conductance. To perform a prediction, type the angle values in the variable.

- The energy (E) will be in the range of 0.285 to 0.306. Values outside this range can lead to unreliable predictions as the model has not been trained beyond these limits.

- Angle (θ) (twist angle between graphene layers), must be within the range 1.17° to 4°. Any value outside this range produce inaccurate results. Values outside this range can lead to unreliable predictions as the model has not been trained beyond these limits.

- Arrival and Exit Pairs were indexed according to the following mapping: The "u" (up) and "d" (down) indicate the layer of the bilayer graphene where the contact is located.

TBG device simulation



5. Run the Script

📌 Python Script for Conductance Prediction

```
import os

import joblib

import pandas as pd

import numpy as np

from minisom import MiniSom

import urllib.request


# 📌 Load Pre-trained MiniSom Model

som = joblib.load("minisom_clusterizer.pkl")

som_x, som_y = som._weights.shape[:2]


# 📌 Load Scalers

scaler_X_standard = joblib.load("scaler_X_standard.pkl")

scaler_X_minmax = joblib.load("scaler_X_minmax.pkl")

scaler_y_standard = joblib.load("scaler_y_standard.pkl")

scaler_y_minmax = joblib.load("scaler_y_minmax.pkl")


# 📌 Load GBR Models for Each Cluster

models = {i: joblib.load(f"gbr_model_cluster_{i}.pkl") for i in range(9)}


print("✅ Models and scalers loaded!")


# 📌 Function to generate all valid contact pairs
```

```
def generate_contact_pairs():
    return [(i, j) for i in range(1, 9) for j in range(1, 9) if i != j]

# 📌 User selects the prediction angle BEFORE clustering
new_angle = float(input("\n📌 Enter an angle for prediction (between 1.17 and 4.0): "))

# 📌 Generate all possible contact pairs and energy values
contact_pairs = generate_contact_pairs()
data = []
for arrival, exit_ in contact_pairs:
    energy_values = np.linspace(0.285, 0.306, 206)
    for energy in energy_values:
        data.append([energy, new_angle, arrival, exit_])

df = pd.DataFrame(data, columns=["Energy", "Angle", "ArrivalPair", "ExitPair"])
print(f"\n✅ DataFrame created with {len(df)} samples!")  # Should contain 9,888 samples

# 📌 Normalize the data
df_standardized = scaler_X_standard.transform(df)
df_normalized = scaler_X_minmax.transform(df_standardized)

# 📌 Cluster the samples using the trained MiniSom
clusters = [som.winner(sample) for sample in df_normalized]
df["Cluster"] = [x[0] * som_y + x[1] for x in clusters]

print("\n📌 Clustering completed!")
```

```
# 📌 Predict conductance using the GBR model for each cluster
predicted_conductance = []
for i, sample in enumerate(df_normalized):
    cluster = df["Cluster"].iloc[i]
    model = models.get(cluster)

    if model:
        pred_norm = model.predict(sample.reshape(1, -1))
        pred_std = scaler_y_minmax.inverse_transform(pred_norm.reshape(-1, 1))
        pred_real = scaler_y_standard.inverse_transform(pred_std)
        predicted_conductance.append(pred_real[0][0])
    else:
        predicted_conductance.append(None)

df["PredictedConductance"] = predicted_conductance

print("\n✅ Prediction process completed!")

# 📌 Solicitar ao usuário o diretório para salvar os arquivos
while True:
    save_directory = input("\n📌 Enter the directory where you want to save the output files: ").strip()

    # Verifica se o caminho não está vazio
    if save_directory:
        try:
            os.makedirs(save_directory, exist_ok=True)  # Cria a pasta, se não existir
            print(f"\n✅ Files will be saved in: {save_directory}")
```

```
            break
        except Exception as e:
            print(f"❌ Error: {e}. Please enter a valid directory.")
    else:
        print("❌ Invalid input. Please enter a valid directory path.")



# 📌 Contact mapping
arrival_mapping = {'1u': 1, '2u': 2, '3u': 3, '4u': 4, '1d': 5, '2d': 6, '3d': 7, '4d': 8}
reverse_mapping = {v: k for k, v in arrival_mapping.items()}

# 📌 Iterate through each ArrivalPair/ExitPair and save individual files
saved_files = []
for (arrival, exit_), group_df in df.groupby(["ArrivalPair", "ExitPair"]):
    arrival_name = reverse_mapping[arrival]
    exit_name = reverse_mapping[exit_]
    angle_str = str(new_angle).replace(".", "p")

    filename = f"G{arrival_name}{exit_name}_AA50x50_{angle_str}_0T_th.dat"
    output_file = os.path.join(save_directory, filename)

    # 📌 Save only Energy and PredictedConductance columns, without header
    df_to_save = group_df[["Energy", "PredictedConductance"]]
    df_to_save.to_csv(output_file, sep=" ", index=False, header=False)

    print(f"✅ File saved: {output_file}")
    saved_files.append(output_file)
```

```
print("\n✅ All files have been generated and saved successfully!")



# 📌 Caminho onde os arquivos de condutância foram salvos

octave_script_filename = "matriz_resistencia_8c_matheus.m"

octave_script_path = os.path.join(save_directory, octave_script_filename)


# 📌 Verifica se o arquivo já existe na pasta

if not os.path.exists(octave_script_path):

    octave_script_url =
"https://raw.githubusercontent.com/MatheusHGK/GBR_TBG/main/matriz_resistencia_8c_matheus.m"


    try:

        urllib.request.urlretrieve(octave_script_url, octave_script_path)

        print(f"\n✅ Downloaded {octave_script_filename} to {save_directory}")

    except Exception as e:

        print(f"❌ Error downloading the script: {e}")

else:

    print(f"\n📌 {octave_script_filename} already exists in {save_directory}. Download skipped.")


# 📌 Mensagem final com instruções

print("\n📌 Next Steps:")

print(f"1️⃣ Open the script '{octave_script_filename}' in a text editor.")

print("2️⃣ Modify the following parameters according to your dataset:")

print("   - ang: Set this to the correct dataset angle.")

print("   - I1, I2: Define the leads where the current enters (I1) and exits (I2), assuming I2 has zero voltage.")

print("   - V1, V2: Specify the leads for voltage measurement.")
```

```
print("\n3️⃣ Ensure the filenames in the script match the actual dataset filenames.")

print("   Example: If the dataset uses 3.5°, rename files to G2u1u_AA50x50_3p5_0T_th0.dat")

print("\n✅ Once these steps are completed, run the script in Octave to compute resistance values.")
```

6. Resistance Calculation

6.1 Overview of the Methodology

- After generating the predicted conductance files, the next step is to calculate the resistance values. The resistance computation follows a structured process that integrates the conductance predictions into an Octave script designed for this purpose.

6.2 Running the Resistance Calculation in Octave

Step 1: Ensure the Script is in the Correct Directory

- When running the Python script for conductance prediction, the **Octave script** (matriz_resistencia_8c_matheus.m) is automatically saved in the same directory as the “predicted condutances.dat files”.

Step 2: Modifying the Input Parameters

- Before running the script, it is necessary to update key parameters within the **Octave script** to match your specific dataset.

- Open the **matriz_resistencia_8c_matheus.m** file in a text editor and locate the following lines:

```
ang = 3.0; % Set this to the correct dataset angle

I1 = 2;    % Define the lead where current enters

I2 = 4;    % Define the lead where current exits (assumed to have voltage 0)

V1 = 1;    % Define the first voltage measurement lead

V2 = 5;    % Define the second voltage measurement lead
```

- Modify these values based on your simulation requirements.

Step 3: Ensuring Filenames Match the Dataset

- The script reads .dat files based on predefined naming conventions. If your dataset was generated with a different angle, update all occurrences of the angle in the script accordingly.

For example:

- If your dataset uses **3.5°**, ensure the filenames match: G2u1u_AA50x50_3p5_0T_th0.dat
- If your dataset uses **5.0°**, ensure the filenames match: G2u1u_AA50x50_5p0_0T_th0.dat

Having incorrect filenames will result in errors when running the script.