

# 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.

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.

### Required Files:

File Name	Google Drive Link
minisom_clusterizer.pkl	<a href="#">Download</a>
scaler_X_standard.pkl	<a href="#">Download</a>
scaler_X_minmax.pkl	<a href="#">Download</a>
scaler_y_standard.pkl	<a href="#">Download</a>
scaler_y_minmax.pkl	<a href="#">Download</a>

GBR Models for Each Cluster	Google Drive Link
gbr_model_cluster_0.pkl	<a href="#">Download</a>
gbr_model_cluster_1.pkl	<a href="#">Download</a>
gbr_model_cluster_2.pkl	<a href="#">Download</a>
gbr_model_cluster_3.pkl	<a href="#">Download</a>
gbr_model_cluster_4.pkl	<a href="#">Download</a>
gbr_model_cluster_5.pkl	<a href="#">Download</a>
gbr_model_cluster_6.pkl	<a href="#">Download</a>
gbr_model_cluster_7.pkl	<a href="#">Download</a>
gbr_model_cluster_8.pkl	<a href="#">Download</a>

- After downloading, place all the files in the **same folder** as the Python script.

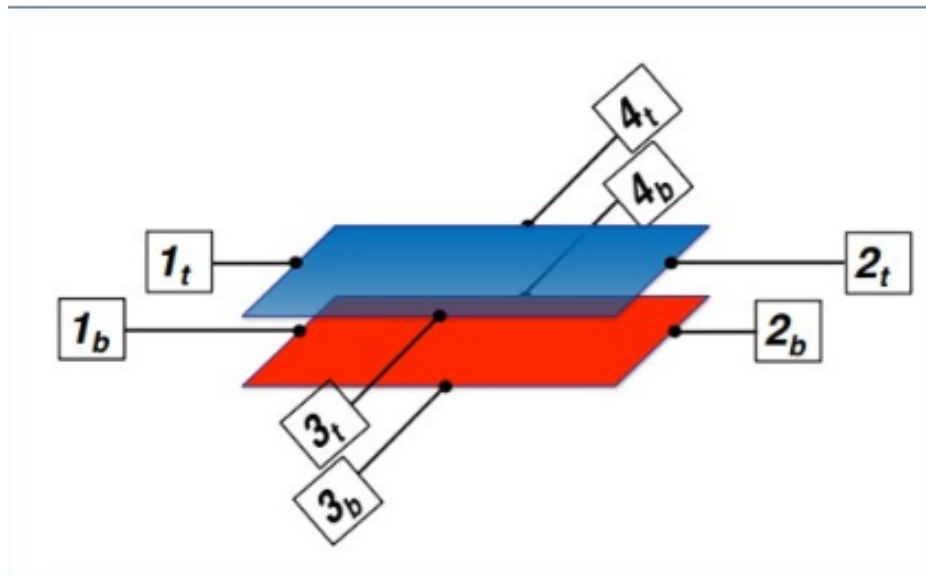
### 3. Define Output Folder

- In the script in section 4, modify this line to set the directory where the output files will be saved:  
`save_directory=r"C:\Users\YOUR_USERNAME\Desktop\Predicted_Conductance"`. If the folder does not exist, the script will create it automatically.

### 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 ( $\theta$ ) (twist angle between graphene layers), must be within the range  $1.17^\circ$  to  $4^\circ$ . 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

- if the user is running the script from a command-line terminal, save the following Python script as `predict_conductance.py` and execute it with:  
**`python predict_conductance.py`**

### Python Script for Conductance Prediction


```
import os

import joblib

import pandas as pd

import numpy as np

from minisom import MiniSom

#  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!")
```

```
# 💡 Save results to .dat files in the correct format
```

```
save_directory = r"C:\Users\YOUR_USERNAME\Desktop\Predicted_Conductance" # Change this if needed
```

```
os.makedirs(save_directory, exist_ok=True)
```

```
# 💡 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}_OT_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!")
```