Model creation and Dataset

The aim of this file is to:

- Clean the input CSV files
- Turn them into Numpy format
- Train a 1D CNN model
- Make prediction using the trained weights
- using XAI techniques to justify models prediction

```
from utils import *
import torch
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split
from importlib import reload
import utils
reload(utils)  # Reloads the updated utils.py
```

```
<module 'utils' from 'c:\\Users\\amirt\\OneDrive\\Desktop\\RA work\\
Code\\utils.py'>
```

pre-processing

```
# input files
drug_files = {
    "AMP": {
        "path": "Data/Original/fp_comp_dataframe_AMP.csv",
        "drug_name": "AMP",
        "bond_type": "inward"
    },
    "COC": {
        "path": "Data/Original/fp_comp_dataframe_COC.csv",
        "drug_name": "COC",
        "bond_type": "outward"
    },
    "MDMA": {
        "path": "Data/Original/fp_comp_dataframe_MDMA.csv",
        "drug_name": "MDMA",
        "bond_type": "inward"
    }
}

# clean Raw Data
clean_and_save_drug_csv(drug_files,
output_dir="Data/superset/approach_1")
```

```
Saved cleaned CSV for AMP: Data/superset/approach_1\cleaned_AMP.csv
Saved cleaned CSV for COC: Data/superset/approach_1\cleaned_COC.csv
Saved cleaned CSV for MDMA: Data/superset/approach_1\cleaned_MDMA.csv
```

```
c:\Users\amirt\OneDrive\Desktop\RA work\Code\utils.py:45:
FutureWarning: errors='ignore' is deprecated and will raise in a
future version. Use to_numeric without passing `errors` and catch
exceptions explicitly instead
  df_cleaned = df_cleaned.apply(pd.to_numeric,
errors='ignore').fillna(0)
c:\Users\amirt\OneDrive\Desktop\RA work\Code\utils.py:45:
FutureWarning: errors='ignore' is deprecated and will raise in a
future version. Use to_numeric without passing `errors` and catch
exceptions explicitly instead
  df_cleaned = df_cleaned.apply(pd.to_numeric,
errors='ignore').fillna(0)
c:\Users\amirt\OneDrive\Desktop\RA work\Code\utils.py:45:
FutureWarning: errors='ignore' is deprecated and will raise in a
future version. Use to_numeric without passing `errors` and catch
exceptions explicitly instead
  df_cleaned = df_cleaned.apply(pd.to_numeric,
errors='ignore').fillna(0)
```

```
# Preparing NumPy Features
X, y, y_labels = prepare_numpy_data(["AMP", "COC"],

input_dir="Data/superset/approach_1",

output_dir="Data/superset/approach_1/num")
```

```
Saved NumPy arrays to: Data/superset/approach_1/num
```

```
#Defining Dataset & Dataloader
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

train_dataset = FingerprintDataset(X_train, y_train)
test_dataset = FingerprintDataset(X_test, y_test)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32)
```

```
# Defineing and Training The 1D CNN model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = CNN1D(input_length=X.shape[1], num_classes=len(y_labels))
train_model(model, train_loader, device=device, epochs=60, lr=0.001)
```

```
Epoch 1/60 - Loss: 22.4779
Epoch 2/60 - Loss: 21.9422
```

```
Epoch 3/60 - Loss: 20.9160
Epoch 4/60 - Loss: 19.8065
Epoch 5/60 - Loss: 18.1111
Epoch 6/60 - Loss: 15.1165
Epoch 7/60 - Loss: 11.8257
Epoch 8/60 - Loss: 8.5553
Epoch 9/60 - Loss: 5.8308
Epoch 10/60 - Loss: 4.1333
Epoch 11/60 - Loss: 3.0166
Epoch 12/60 - Loss: 2.4249
Epoch 13/60 - Loss: 1.8224
Epoch 14/60 - Loss: 1.4709
Epoch 15/60 - Loss: 1.2614
Epoch 16/60 - Loss: 1.0412
Epoch 17/60 - Loss: 0.8854
Epoch 18/60 - Loss: 0.8005
Epoch 19/60 - Loss: 0.6906
Epoch 20/60 - Loss: 0.6213
Epoch 21/60 - Loss: 0.5505
Epoch 22/60 - Loss: 0.5690
Epoch 23/60 - Loss: 0.4691
Epoch 24/60 - Loss: 0.4293
Epoch 25/60 - Loss: 0.3804
Epoch 26/60 - Loss: 0.3646
Epoch 27/60 - Loss: 0.3453
Epoch 28/60 - Loss: 0.3102
Epoch 29/60 - Loss: 0.2956
Epoch 30/60 - Loss: 0.2858
Epoch 31/60 - Loss: 0.2713
Epoch 32/60 - Loss: 0.2400
Epoch 33/60 - Loss: 0.2224
Epoch 34/60 - Loss: 0.2068
Epoch 35/60 - Loss: 0.2051
Epoch 36/60 - Loss: 0.1858
Epoch 37/60 - Loss: 0.1771
Epoch 38/60 - Loss: 0.1679
Epoch 39/60 - Loss: 0.1629
Epoch 40/60 - Loss: 0.1484
Epoch 41/60 - Loss: 0.1426
Epoch 42/60 - Loss: 0.1335
Epoch 43/60 - Loss: 0.1345
Epoch 44/60 - Loss: 0.1289
Epoch 45/60 - Loss: 0.1292
Epoch 46/60 - Loss: 0.1144
Epoch 47/60 - Loss: 0.1089
Epoch 48/60 - Loss: 0.1059
Epoch 49/60 - Loss: 0.1064
Epoch 50/60 - Loss: 0.0946
Epoch 51/60 - Loss: 0.0936
```

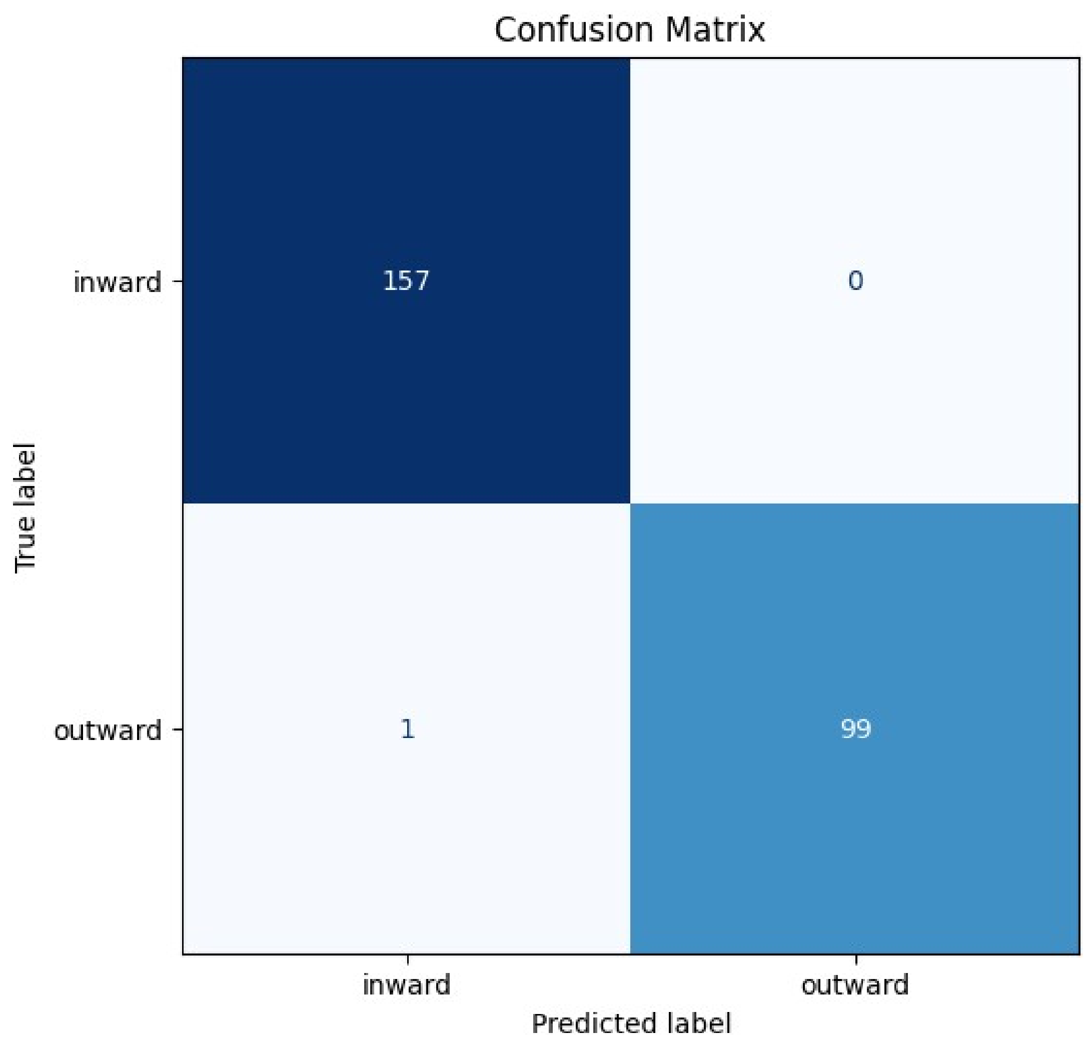
```
Epoch 52/60 - Loss: 0.0932
Epoch 53/60 - Loss: 0.0851
Epoch 54/60 - Loss: 0.0816
Epoch 55/60 - Loss: 0.0794
Epoch 56/60 - Loss: 0.0744
Epoch 57/60 - Loss: 0.0725
Epoch 58/60 - Loss: 0.0743
Epoch 59/60 - Loss: 0.0697
Epoch 60/60 - Loss: 0.0651
```

```python
# Model Evaluation
y_true, y_pred = evaluate_model(model, test_loader, device=device)

# Saving the model weights
save_model(model, "Data/superset/approach_1/model/cnn1d_weights.pth")
```

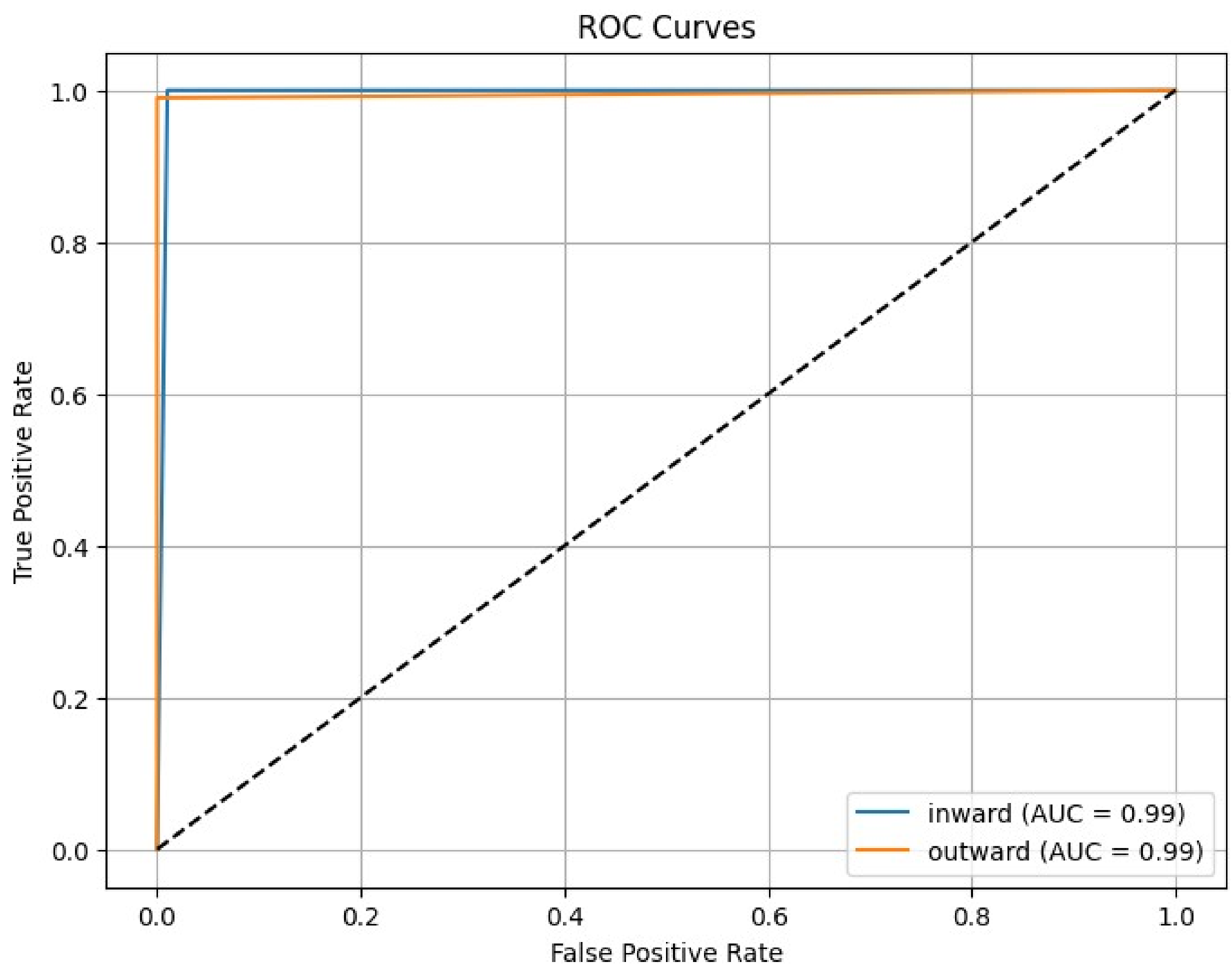
```
Model weights saved to:
Data/superset/approach_1/model/cnn1d_weights.pth
```

```python
from utils import plot_classification_metrics

# Model Diagnostics
plot_classification_metrics(y_true, y_pred, y_labels)
```

```
Total Predictions Per Class:
outward: 99 frames
inward: 158 frames
```

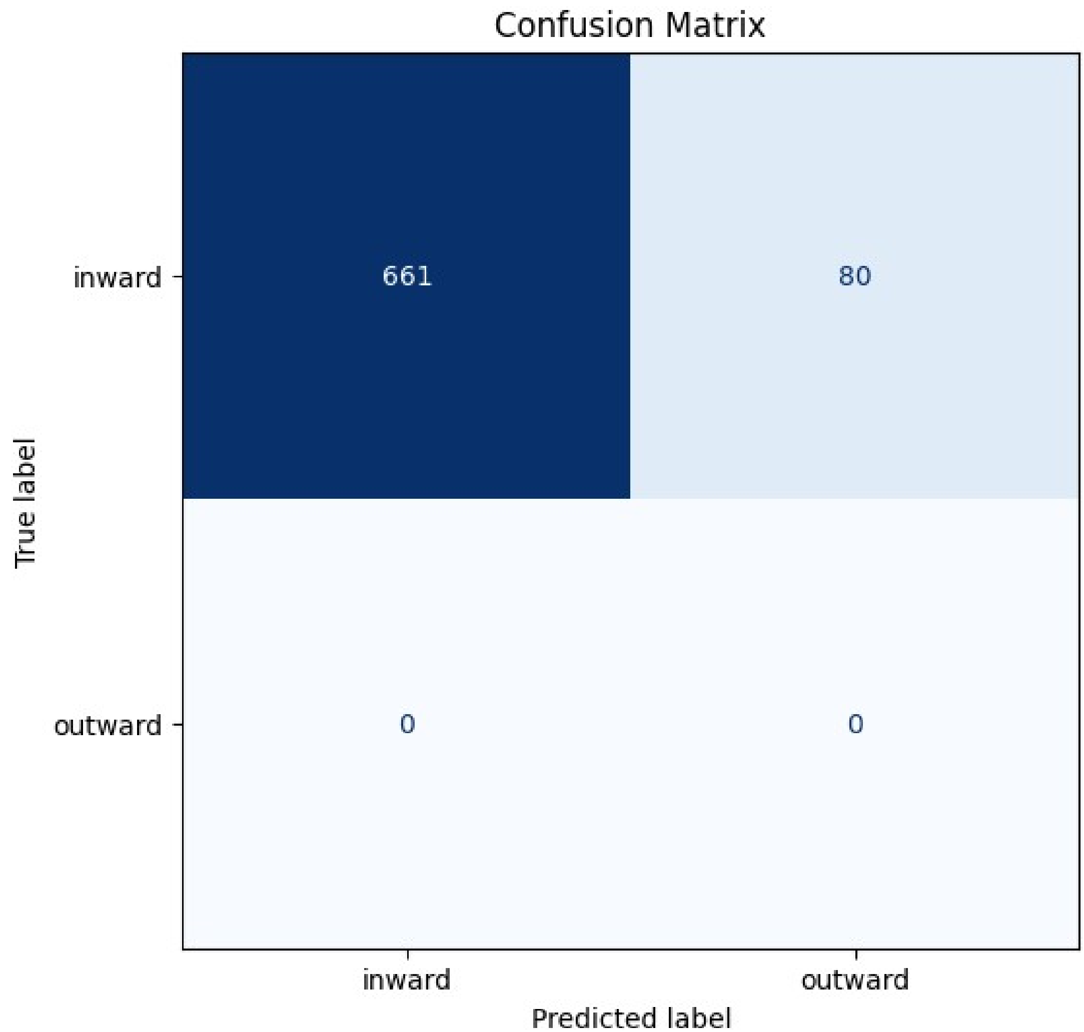



```
Classification Report:
              precision    recall  f1-score   support

      inward       0.99      1.00      1.00       157
     outward       1.00      0.99      0.99       100

    accuracy                           1.00       257
   macro avg       1.00      0.99      1.00       257
weighted avg       1.00      1.00      1.00       257


Accuracy: 0.9961
Weighted F1 Score: 0.9961
```



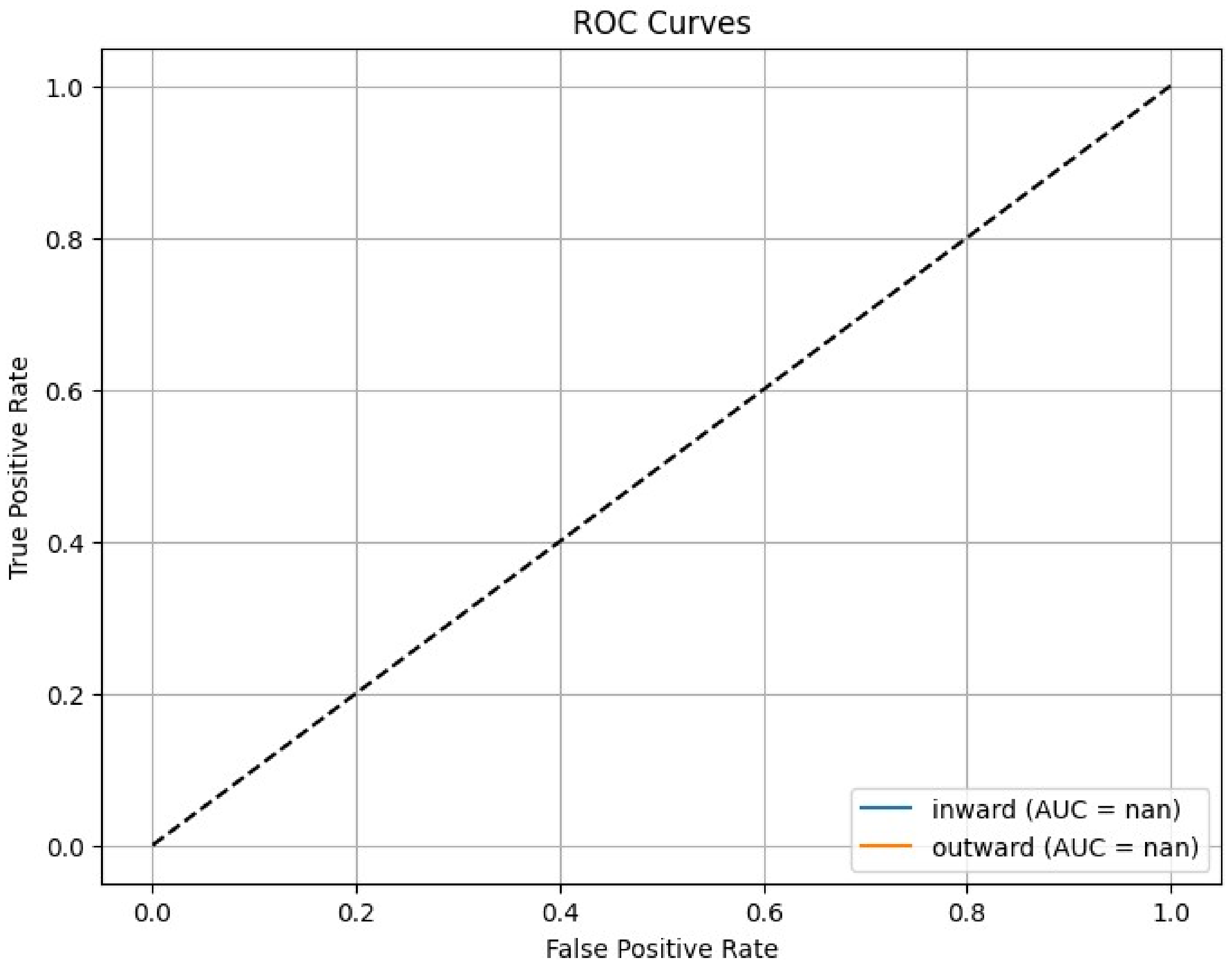
now evaluating on unseen data such as MDMA

```
from utils import evaluate_on_new_csv

evaluate_on_new_csv(
    csv_path="Data/superset/approach_1/cleaned_MDMA.csv",
    model_path="Data/superset/approach_1/model/cnn1d_weights.pth",
    y_labels=y_labels,
    device=device
)


Total Predictions Per Class:
outward: 80 frames
inward: 661 frames
```




```
Classification Report:
              precision    recall  f1-score   support

      inward       1.00      0.89      0.94       741
     outward       0.00      0.00      0.00         0

    accuracy                           0.89       741
   macro avg       0.50      0.45      0.47       741
weighted avg       1.00      0.89      0.94       741


Accuracy: 0.8920
Weighted F1 Score: 0.9429

c:\Users\amirt\OneDrive\Desktop\RA work\Code\god\lib\site-packages\
sklearn\metrics\_classification.py:1706: UndefinedMetricWarning:
Recall is ill-defined and being set to 0.0 in labels with no true
samples. Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, f"{metric.capitalize()} is", 
result.shape[0])
c:\Users\amirt\OneDrive\Desktop\RA work\Code\god\lib\site-packages\
sklearn\metrics\_classification.py:1706: UndefinedMetricWarning: 
Recall is ill-defined and being set to 0.0 in labels with no true 
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", 
result.shape[0])
c:\Users\amirt\OneDrive\Desktop\RA work\Code\god\lib\site-packages\
sklearn\metrics\_classification.py:1706: UndefinedMetricWarning: 
Recall is ill-defined and being set to 0.0 in labels with no true 
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", 
result.shape[0])
c:\Users\amirt\OneDrive\Desktop\RA work\Code\god\lib\site-packages\
sklearn\metrics\_ranking.py:1192: UndefinedMetricWarning: No negative 
samples in y_true, false positive value should be meaningless
  warnings.warn(
c:\Users\amirt\OneDrive\Desktop\RA work\Code\god\lib\site-packages\
sklearn\metrics\_ranking.py:1201: UndefinedMetricWarning: No positive 
samples in y_true, true positive value should be meaningless
  warnings.warn(
```



LIME

```
from utils import CNN1D, explain_prediction_with_lime
from sklearn.preprocessing import LabelEncoder

# Load cleaned MDMA CSV
mdma_df = pd.read_csv("Data/superset/approach_1/cleaned_MDMA.csv",
header=[0, 1])

# Extract features (exclude meta columns)
X = mdma_df.loc[:, mdma_df.columns.get_level_values(0) !=
"meta"].to_numpy(dtype=np.float32)
y = mdma_df[("meta", "bond_type")].values

# Encode labels based on training label space
le = LabelEncoder()
le.fit(y_labels)
y_encoded = le.transform(y)
```

```python
# Prepare feature names from column MultiIndex
feature_names = [
    f"{residue}_{itype}"
    for residue, itype in mdma_df.columns
    if residue != "meta"
]

# Create Dataset
mdma_dataset = FingerprintDataset(X, y_encoded)

# Load trained model
model = CNN1D(input_length=X.shape[1], num_classes=len(y_labels))
model.load_state_dict(torch.load("Data/superset/approach_1/model/cnn1d
_weights.pth", map_location=device))
model.to(device)

explain_index = 100  # which frame to check?

# Run LIME explanation
explain_prediction_with_lime(
    model=model,
    dataset=mdma_dataset,
    index=explain_index,
    y_labels=y_labels,
    feature_names=feature_names,
    device=device

)
```

```
<IPython.core.display.HTML object>
```

SHAP

```python
import pandas as pd

# Load AMP training data
df_train = pd.read_csv("Data/superset/approach_1/cleaned_AMP.csv",
header=[0, 1])

# Extract training feature columns (exclude meta)
train_cols = df_train.loc[:, df_train.columns.get_level_values(0) !=
"meta"].columns
real_feature_names = [f"{a} ({b})" for a, b in train_cols]

# Extract X_train features
X_train = df_train[train_cols].to_numpy(dtype=np.float32)

# Load MDMA data
df_mdma = pd.read_csv("Data/superset/approach_1/cleaned_MDMA.csv",
header=[0, 1])
```

```
# Reindex MDMA to match AMP columns — fill missing columns with 0s
df_mdma_aligned = df_mdma.reindex(columns=train_cols, fill_value=0)
X_mdma = df_mdma_aligned.to_numpy(dtype=np.float32)
```

```
from utils import CNN1D
# Make sure model is defined and matches training
model = CNN1D(input_length=X_train.shape[1],
num_classes=len(y_labels))
model.load_state_dict(torch.load("Data/superset/approach_1/model/cnn1d
_weights.pth", map_location=device))
model.to(device)
model.eval()
```

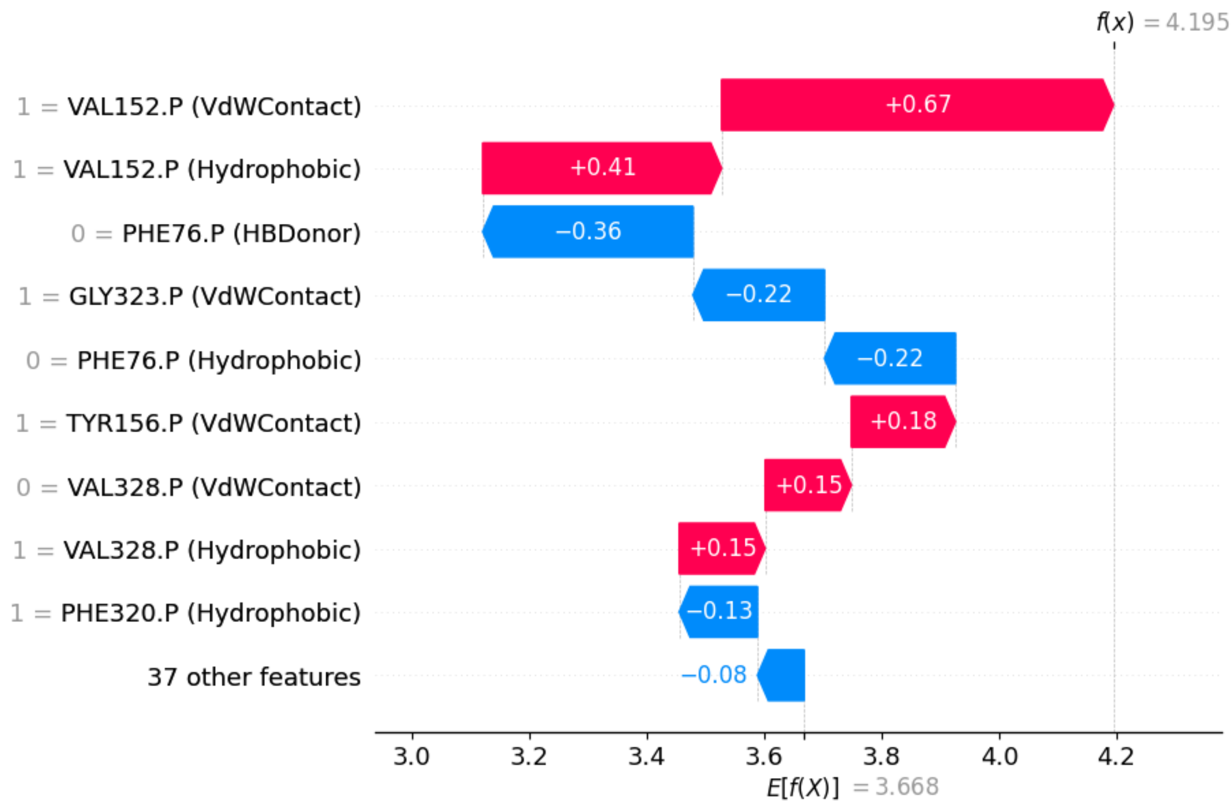
```
CNN1D(
  (net): Sequential(
    (0): Conv1d(1, 16, kernel_size=(3,), stride=(1,), padding=(1,))
    (1): ReLU()
    (2): MaxPool1d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv1d(16, 32, kernel_size=(3,), stride=(1,), padding=(1,))
    (4): ReLU()
    (5): AdaptiveMaxPool1d(output_size=1)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=32, out_features=2, bias=True)
  )
)
```

```
from utils import explain_prediction_shap_deep

explain_prediction_shap_deep(
    model=model,
    X_train=X_train,
    X_mdma=X_mdma,
    real_feature_names=real_feature_names,
    frame_index=50,  # frame name
    device=device
)
```

```
# Show the predicted class for the selected MDMA frame
logits = model(torch.tensor(X_mdma[50][None, None, :],
dtype=torch.float32).to(device))
predicted_class = torch.argmax(logits).item()
print(f"Model prediction for the selected frame:
{y_labels[predicted_class]}")
```

```
c:\Users\amirt\OneDrive\Desktop\RA work\Code\god\lib\site-packages\
shap\explainers\_deep\deep_pytorch.py:255: UserWarning: unrecognized
nn.Module: Flatten
  warnings.warn(f"unrecognized nn.Module: {module_type}")
```

```
c:\Users\amirt\OneDrive\Desktop\RA work\Code\god\lib\site-packages\
shap\explainers\_deep\deep_pytorch.py:255: UserWarning: unrecognized
nn.Module: AdaptiveMaxPool1d
  warnings.warn(f"unrecognized nn.Module: {module_type}")
```




```
Model prediction for the selected frame: inward
```