ECG Findings (Demo)

Setting Up xLSTMTime: A Comprehensive Guide for Beginners

XLSTM -Time for PTBXL ECG Dataset

1. Download the zip file ***xLSTMTime-main.zip*** from Google Drive and extract it to the desired location
   Google Drive link -
   It includes original code from the author of the xLSTM-Time model along with modification required for the PTBXL dataset
   The original Github repository for xLSTM-Time can be found here:
   https://github.com/muslehal/xLSTMTime

If you want to create a virtual environment then follow step 2 otherwise skip step 2

2. Create Virtual Environment & Installing required packages
   > Open Terminal (you can use vscode terminal or your machine terminal)
   > Locate to ***xLSTMTime-main*** using the **cd** command
   > Run ***python -m venv .venv***
   > Run **source venv/bin/activate** to activate newly created virtual environment
   > Inside the virtual environment Run **pip install -r requirements.txt**
3. Installing required packages without a virtual environment
   > Run **pip install -r requirements.txt** to have the required packages in the global python environment
4. Try to run main.py

Understanding Code Structure and Process Flow

all_six_datasets - This folder has datasets which were used by the author. Datasets are in *.csv* format. To keep it consistent new datasets are all added here inside folders with names defined in *DSETS* in datautils.py. Make sure there is a *.csv* dataset file inside dedicated folders.

data_preprocessing.py - This file reads ECG signal files from *the PTB-XL-Dataset* folder and creates a *.csv* file that can be fed to the model.
Understanding the format in which we can feed the data is extremely crucial. First of all, in the data_preprocessing.ipynp file, we are processing the PTBXL dataset, which is also present in the main folder.
The format of the data is called 'expanded form' which was used in the book (Modern Time Series Forecasting with Python) we followed throughout the course.
Expanded form for PTBXL:
* Each ecg_id has 0 - 999 timesteps because we are using 10s long ECG signal and 100Hz frequency
* we have total 21799 ecg_ids, all the ecg_ids are stacked up with their 999 timesteps
* we have a total of 21799 * 1000 = 21799000 rows and 12 columns for 12 leads in our CSV file

main.py - it has all the crucial parameters to set, which is passed to 'args' to model

1. **Parameters/hyperparameters**
    - --*dset:* dataset name from the list ‘DSETS’ defined in datautils.py
    - --*context_points:* Sequence length (window size) of time-series data fed into the model. If you expect long-range dependencies, larger values might be better. If patterns are shorter, smaller values might be sufficient and faster to train. A good starting point would be to test values equal to your data's primary periodicity.
    - --*target_points:* Forecast horizon - the number of time steps the model predicts into the future. In your Dataset_PTBXL, this corresponds to pred_len. Integer, default: 96. Experimentation: This is determined by your forecasting goal. However, smaller forecast horizons are generally easier to predict.
    - --scaler: Scaling method for input data. String, default: 'standard'. Options could include 'standard' (StandardScaler), 'minmax' (MinMaxScaler), or None (no scaling).
    - --features: Specifies which features to use from the dataset - 'M' (Multivariate) or 'S' (Single/Univariate).
    - --*batch_size:* Number of samples processed in each iteration of training. Larger batch sizes can lead to faster training but might require more memory. Smaller batch sizes can be more stable but might take longer.
    - --*num_workers:* Number of subprocesses to use for data loading. Helps speed up data loading. Integer, default: 1. *Experimentation:* Set to the number of CPU cores.
    - --*patch_len:* Length of each patch. Integer, default: 12.
    - --*stride:* Stride between patches. Integer, default: 12. If stride == patch_len, the patches are non-overlapping. If stride < patch_len, the patches overlap.

      Patching can help the model focus on local patterns and can reduce the computational costs for very long sequences. If you have strong long-range dependencies, patching might hurt performance. If the data is locally smooth, overlapping patches might help.

    - --*revin:* Flag to use RevIN for normalization. Integer, default: 1 (True). RevIN is a technique that can help with the instability of training deep time series models. If your model trains unstably, RevIN can be very helpful.
    - --*n_epochs:* Number of training epochs.
    - --*is_train:* Flag to indicate whether to train the model (1) or run in testing mode (0).
    - --*n_layers*: More layers can capture more complex relationships but can also lead to overfitting and longer training times.
    - --*d_model*: Dimension of the transformer layers (embedding dimension).
    - --*dropout*: Higher dropout can help with overfitting but might slow down training.
    - --*head_dropout*: Head dropout rate. Used to prevent overfitting on heads.

2. **find_lr()** - This function is defined in main.py and sets up the necessary components for training your xLSTM-Time model (data, model, loss function, callbacks) and then uses a learning rate finder to determine a suitable learning rate before starting the full training process, returns suggested learning rate
3. **train_func()** - This function takes the suggested learning rate as an argument and trains the model by setting a few configurations in the function *get_dls(args)* defined in *datautils.py* and a class defined in *src/data/pred_dataset.py*
4. **test_func() -** This functions saves models in *saved_models/−dset (as defined above)* and stores predicted, target values in list named *out,* returns *out*
5. **plot_features_actual_vs_predicted()** - Based on number of feature range defined, it plots actual vs predicted values for different features.

datautils.py - it has a list *DSETS* with names of datasets

It prepares the dataset for training a deep learning model using PyTorch

1. get_dls() - it has a condition for each dataset defined in *DSETS* for checking the current dataset defined in main.py
   Each condition block forwards parameters to the Dataset loading class defined in
   pred_dataset.py . The `get_dls` function loads the dataset and validates it by checking the shape of a sample.

   Key Components

   1. ***Imports**: Uses PyTorch for deep learning, NumPy and Pandas for data handling, and custom modules (`DataLoaders`, `Dataset_PTBXL`) for dataset processing.*
   2. ***Parameters**:*
      - `context_points`*: Past time steps used for prediction.*
      - `target_points`*: Future time steps to predict.*
      - `batch_size` *and* `num_workers`*: For data batch processing and parallelism.*
      - `features`*: Specifies which dataset features to use.*
   3. ***Dataset Loading**:*
      - *The dataset should be in the path specified in* `root_path` *and is loaded from* `patient.csv`*.*
   4. ***Usage**:*
      - *The* `Params` *class sets the configuration values.*
      - *The* `get_dls` *function returns a* `DataLoaders` *object containing the training data.*
      - *After loading, it prints the shape of the first training sample.*
   5. ***Customization:***
      - *Modify* `root_path` *and* `data_path` *for your dataset location.*
      - *Adjust parameters for batch size, features, and time points.*

src/data/pred_dataset.py - This file contains classes for datasets to handle the training, testing, validation split, defining features in the dataset, and size/shape of the dataset.
Dataset_PTBXL and Dataset_longterm classes are added to handle these datasets.

Dataset_PTBXL class:

This class `Dataset_PTBXL` is a custom implementation of a dataset used for time-series forecasting with ECG data. It inherits from `torch.utils.data.Dataset`, and is designed to work with PyTorch for training deep learning models. Here's an explanation of its key components:

Constructor (`__init__` method)

The `__init__` method initializes the dataset, sets various configuration options, and prepares the data for loading. The key arguments are:

- **`root_path`**: The directory where the dataset is stored.
- **`split`**: The dataset split type (`train`, `val`, or `test`).
- **`size`**: Defines the sequence length, label length, and prediction length. Default values are set based on 4-hour intervals.
- **`features`**: Specifies the features to use (`'M'` for multiple features, `'S'` for single feature).
- **`target`**: The target column (usually the column you're trying to predict).
- **`scale`**: Whether to normalize (scale) the data.
- **`timeenc`**: Defines how time encoding works (not implemented in this code snippet).
- **`freq`**: Frequency of time steps (e.g., seconds, minutes).
- **`time_col_name`**: The column name containing timestamps (e.g., `'date'`).
- **`use_time_features`**: Whether to use time-related features for the model.
- **`train_split`** & **`test_split`**: Ratios for splitting the data into training, validation, and test sets.

Reading and Processing the Data (`__read_data__` method)

- **Data Loading**: The dataset is read from a CSV file located at `data_path` within the `root_path`. The data is then processed and split into training, validation, and test sets based on the specified ratios.
- **Scaling**: If `scale=True`, the dataset is normalized using `StandardScaler`. The scaler is fit on the training data and then applied to the entire dataset.

- **Feature Selection**: The `features` argument determines which columns of the data to use. If `'M'`, it takes all columns starting from the 5th column (features). If `'S'`, it only takes the target column.
- **Timestamps**: The `time_col_name` (e.g., `'date'`) is converted to a `datetime` format. Although the time features are not fully implemented in this code, there's an option (`timeenc`) to add them if needed.

Data Slicing

The dataset is split into windows of data:

- **`seq_len`**: The length of the input sequence (how many past time steps are used for prediction).
- **`label_len`**: The length of the label sequence (the length of the true values used for training).
- **`pred_len`**: The length of the prediction sequence (how many future time steps the model predicts).
- Based on the current split (`train`, `val`, `test`), the data is sliced accordingly.

Indexing (`__getitem__` method)

- **Data Retrieval**: For each sample, the `__getitem__` method retrieves the input sequence (`seq_x`) and the output sequence (`seq_y`), using the slicing window defined earlier.
- If `use_time_features` is enabled, time features (such as date/time related data) could be added, but they are currently commented out.
- The method returns the data as a PyTorch tensor via the `_torch` function.

Length of Dataset (`__len__` method)

The `__len__` method returns the number of possible samples in the dataset (i.e., the number of windows that can be created from the data). It ensures that the sequence length and prediction length are respected.

Inverse Transformation (`inverse_transform` method)

The `inverse_transform` method reverses the scaling operation, allowing you to convert the scaled data back to its original form using the fitted scaler.

xlstm1/slstm/cell.py - sLSTM needs GPU for faster processing, in this file it can be defined. There are 2 options CUDA or Vanilla. There is no mention of MacBook's MPS framework.

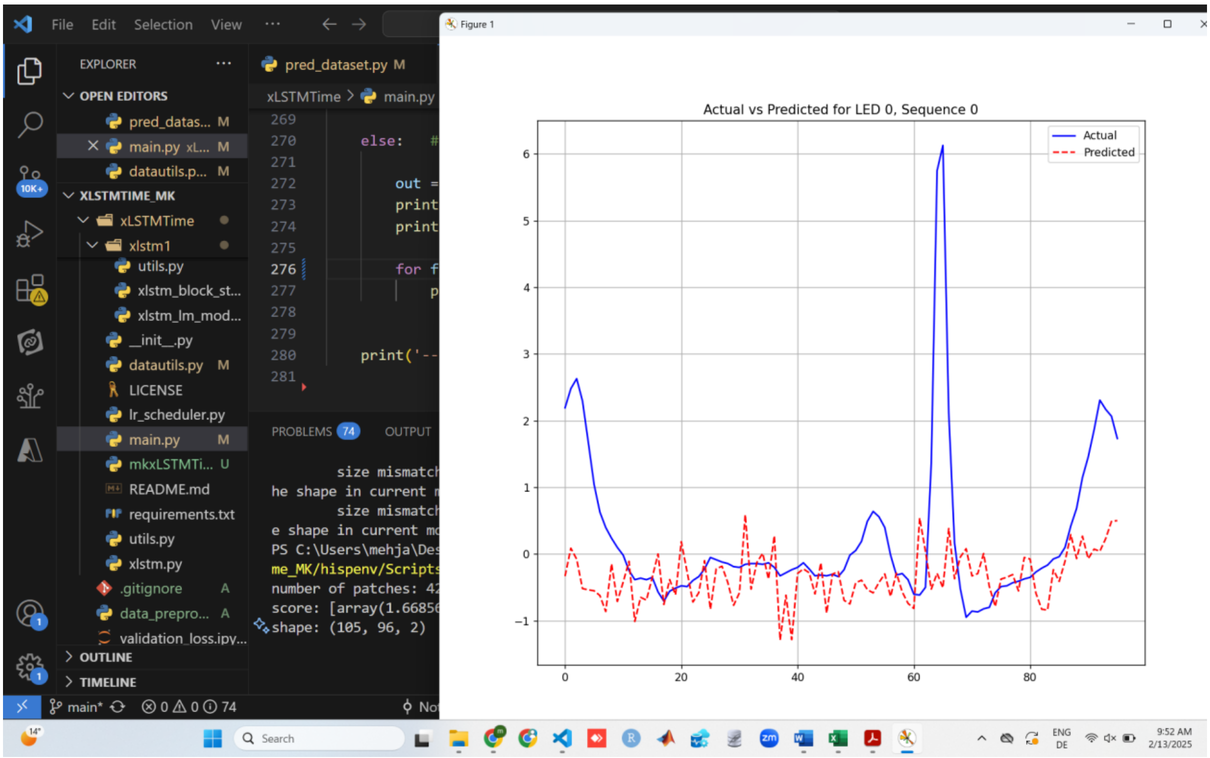
saved_models - Inside this folder, you can find trained models which can be tested by setting –*is_train* parameter in main.py 0.

- **Model # 2 (xLSTMTime_cw512_tw96_patch12_stride12_epochs10_model2)**

ECG data for one patient with 1 Lead and 100 Hz in each second and 10 seconds for one patient therefore 1000 time steps. (no well tuning of parameters)

Context points	512
Target points	96
Batch size	64
lr	1e-3
n_layers	3
d_model	256
dropout	0.2
Patch_len	12
Stride	12

```
 def __init__(self, root_path, split='train', size=None,
                 features='S',
data_path='C:/Users/mehja/Desktop/HIS/wam/HIS_project/xLSTMTime_MK/data/illnes
s-20250129T151711Z-001/illness/patient.csv',
                 target='Lead_11', scale=True, timeenc=0, freq='s',
                 time_col_name='date', use_time_features=False,
                 train_split=0.7, test_split=0.2
                 ):
        # size [seq_len, label_len, pred_len]
        # info
        if size == None:
            self.seq_len = 100
            self.label_len = 12
            self.pred_len = 50
```



➢ **Model # 3 (xLSTMTime_cw100_tw50_patch12_stride12_epochs50_model3)**

ECG data for one patient with 1 Lead and 100 Hz in each second and 10 seconds for one patient therefore 1000 time steps. (Hyperparameters Tuned)

Configuration in main.py

```
args = {
    'context_points': 100,        # Sequence Length
    'target_points': 50,          # Forecast Horizon
    'batch_size': 32,             # Batch Size
    'lr': 1e-3,                   # Learning Rate
    'n_layers': 2,                # Number of Layers
    'd_model': 128,               # Hidden Size
    'dropout': 0.2,               # Dropout Rate
    'patch_len': 12,              # Patch Length
    'stride': 12,                 # Stride Between Patches
}
```

Configuration in pred_dataset.py

```
def __init__(self, root_path, split='test', size=None,
```

```
            features='s',
data_path='C:/Users/mehja/Desktop/HIS/wam/HIS_project/xLSTMTime_MK/data/illness-20250129T151711Z-001/illness/patient.csv',
            target='lead_11', scale=True, timeenc=0, freq='s',
            time_col_name='date', use_time_features=False,
            train_split=0.7, test_split=0.2
            ):
```

```
if size == None:
    self.seq_len = 100  # Sequence Length: number of past time steps (100 is reasonable for 1000 rows)
    self.label_len = 12  # Label Length: number of past time steps to consider for prediction (e.g., 12)
    self.pred_len = 50  # Prediction Length: number of future time steps to forecast (e.g., 50)
```

Why Do We Include 12 Extra Time Steps (label_len)?

Instead of predicting **directly from 100 to 150**, we start seq_y from **time step 88**, not 100.

Reason:

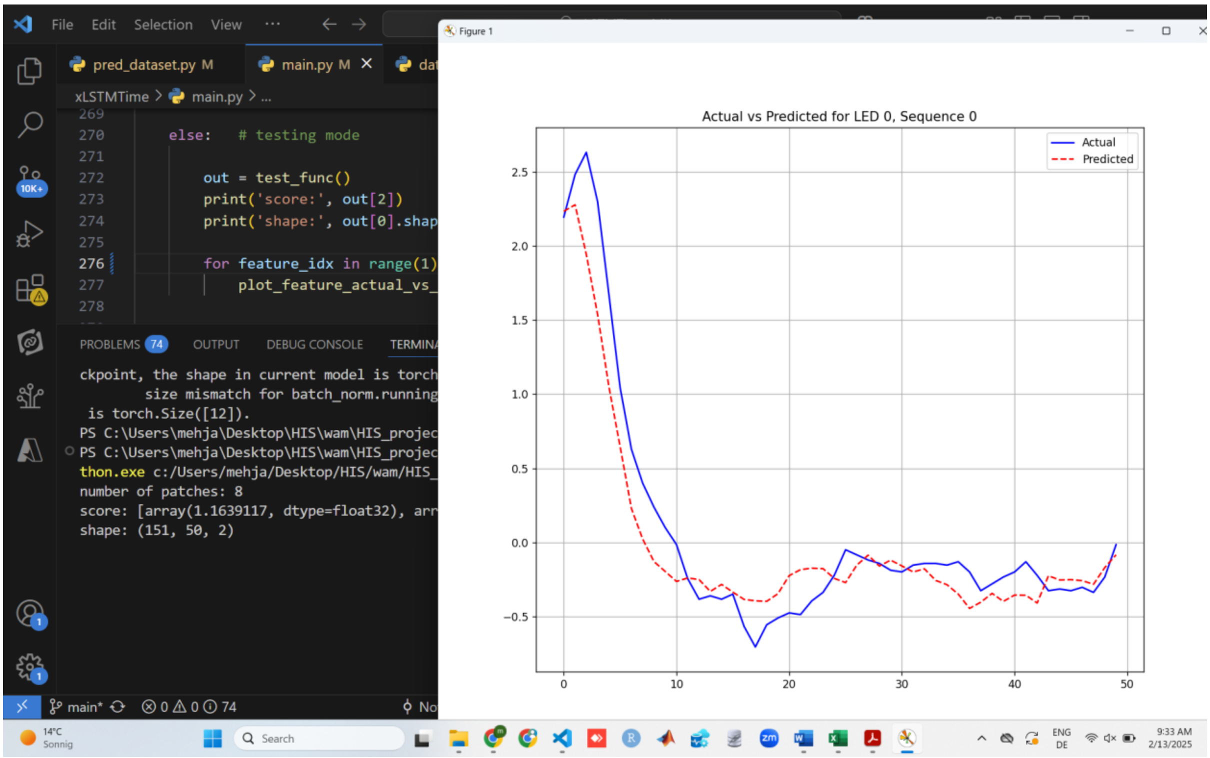
The model needs a transition period to **"warm up"** before making predictions.

- The first **12 time steps (88 to 99)** help the model adjust.
- The next **50 time steps (100 to 149)** are the actual predictions

```sql
Time steps:      0  1  2  ...  88 89 90 ... 99 100 101 102 ... 149
Sequence X:      |-------------------|
Sequence Y:                   |--------------------------------|
Label part:                    |------|
Prediction part:                          |---------------------|
```

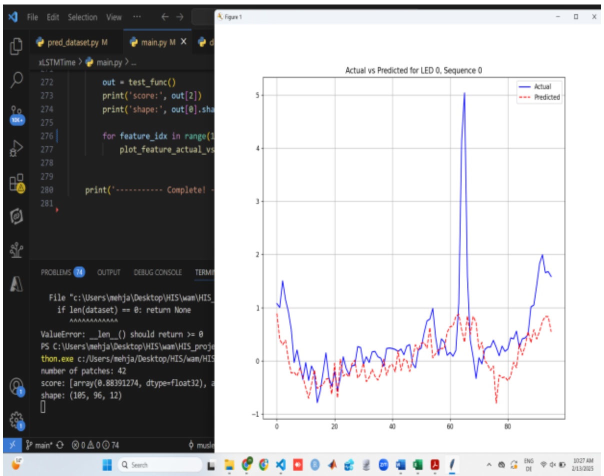
How This Works for All 1000 Rows

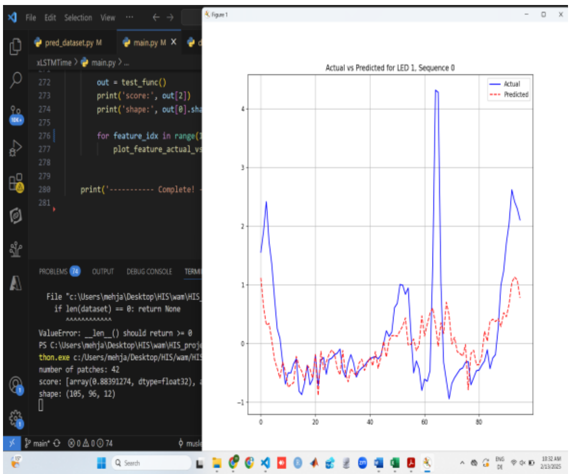
- If you **slide the window** forward (index=1), your **next training set starts at time step 1** instead of 0.
- The model **keeps moving forward until it reaches the end of the 1000 rows.**
- The last possible starting index would be around **850**, since seq_x needs 100 rows.



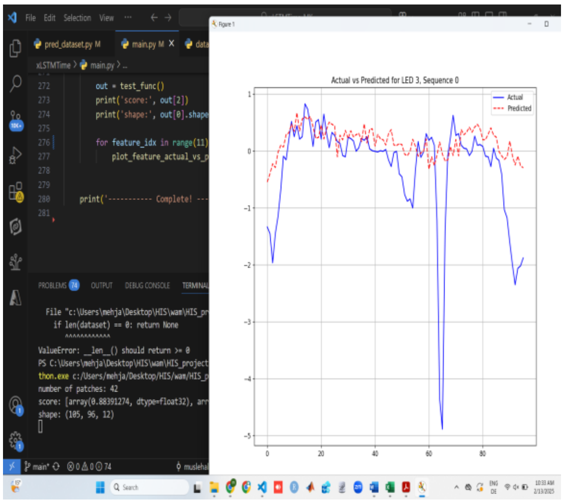
➤ **Model # 1 (xLSTMTime_cw512_tw96_patch12_stride12_epochs50_model1)**

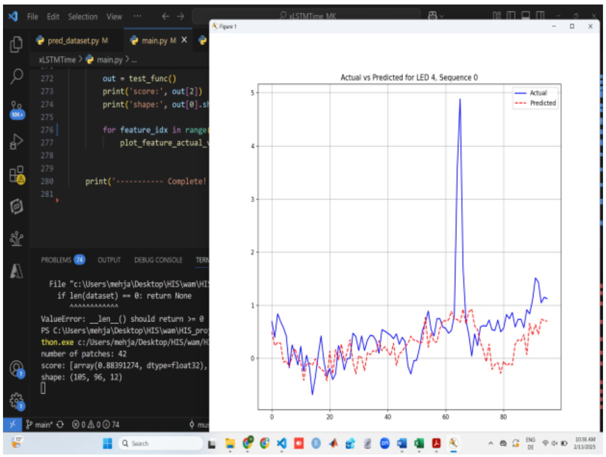
Context points	512
Target points	96
Batch size	32
lr	1e-3
n_layers	3
d_model	128
dropout	0.2
Patch_len	12
Stride	12

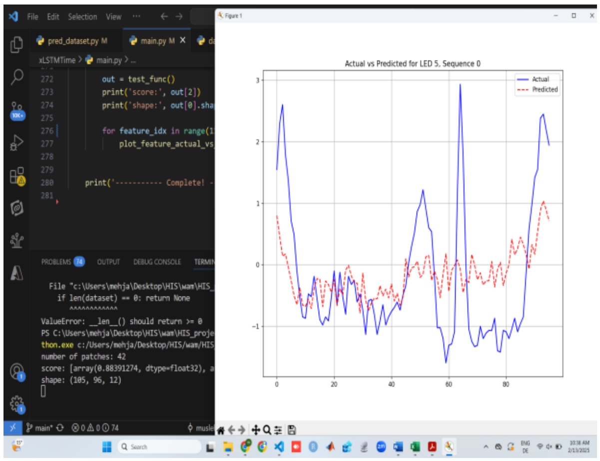


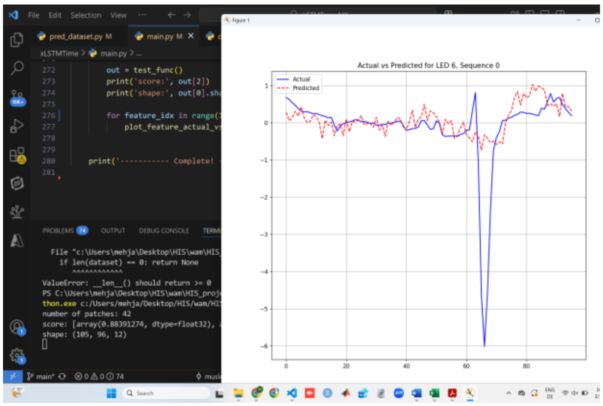


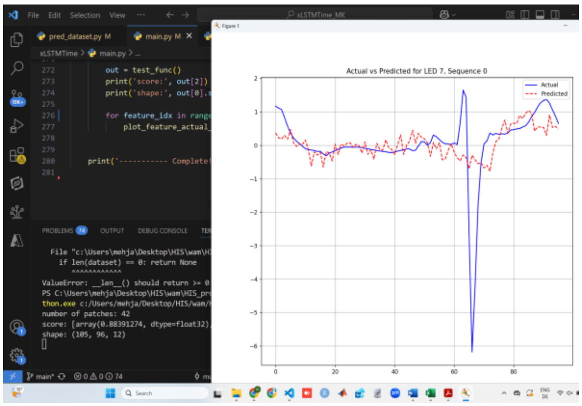

Actual vs Predicted for LED 2, Sequence 0
Actual
Predicted

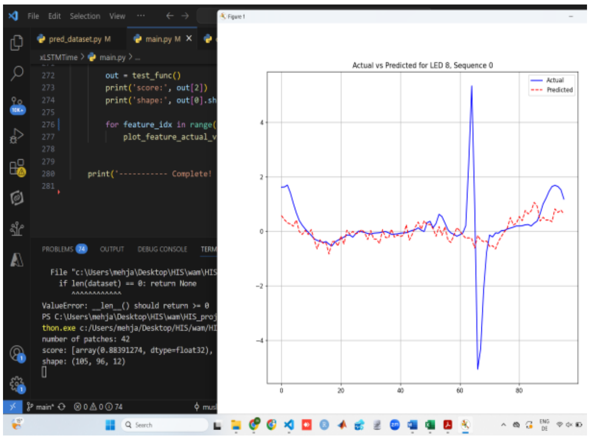

Actual vs Predicted for LED 3, Sequence 0
Actual
Predicted

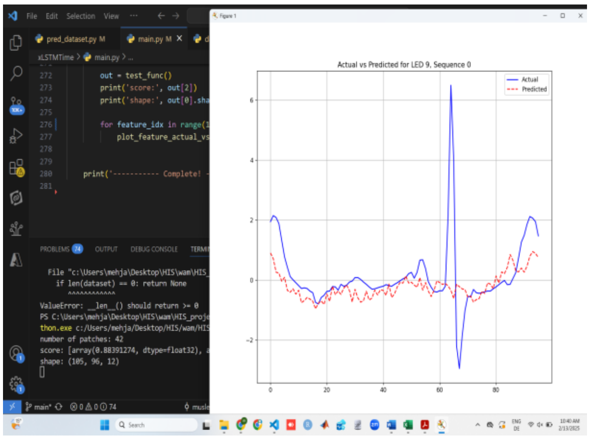

Actual vs Predicted for LED 4, Sequence 0
Actual
Predicted

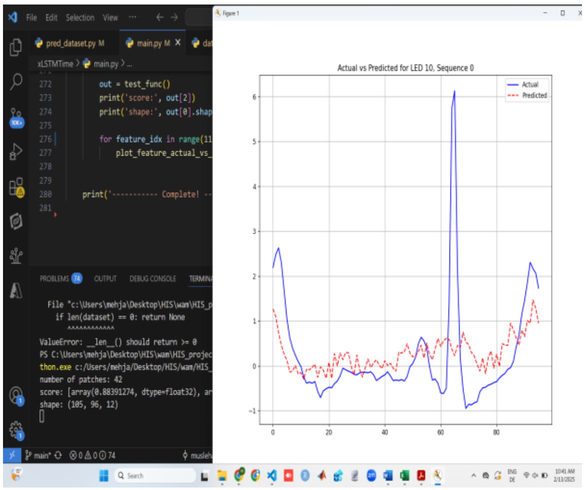

Actual vs Predicted for LED 5, Sequence 0
Actual
Predicted


Actual vs Predicted for LED 6, Sequence 0
Actual
Predicted


Actual vs Predicted for LED 7, Sequence 0
Actual
Predicted







- **Model # 3 (xLSTMTime_cw100_tw50_patch12_stride12_epochs20_model3)**

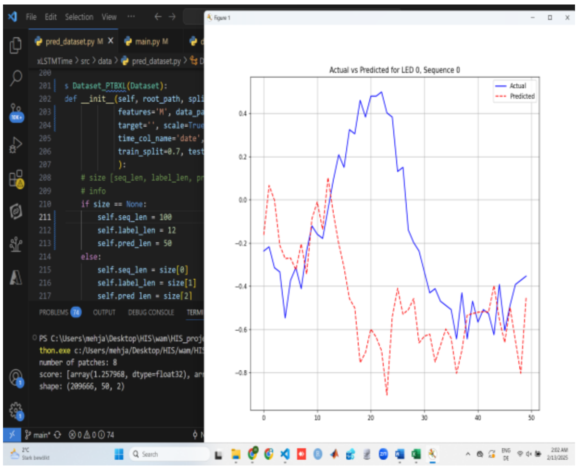
ECG data for one patient with 12 Leads and 100 Hz in each second and 10 seconds for one patient therefore 1000 time steps.

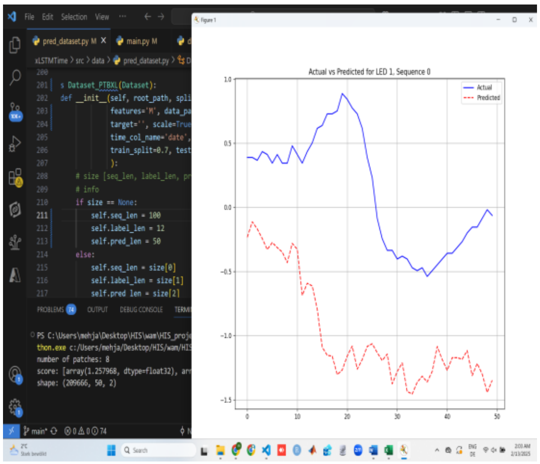
```
args = {
    'context_points': 100,
    'target_points': 50,
    'batch_size': 32,
    'features' : 'M',
    'lr': 1e-3,
    'n_layers': 2,
    'd_model': 128,
    'dropout': 0.2,
    'patch_len': 12,
    'stride': 12
}
```

```
def __init__(self, root_path, split='train', size=None,
```

```
                features='M',
data_path='C:/Users/mehja/Desktop/HIS/wam/HIS_project/xLSTMTime_MK/data/illness-20250129T151711Z-001/illness/patient.csv',
                target='', scale=True, timeenc=0, freq='s',
                time_col_name='date', use_time_features=False,
                train_split=0.7, test_split=0.2
                ):
 if size == None:
            self.seq_len = 100
            self.label_len = 12
            self.pred_len = 50
```

```
 if self.features == 'M' or self.features == 'MS':
          cols_data = df_raw.columns[5:] #takes features from 5th column becuase LED column starts
          df_data = df_raw[cols_data]
```





- **Model # 7 (xLSTMTime_cw512_tw512_patch12_stride12_epochs3_model1)**

context_points - 512

target_points -512

batch_size - 64

scaler - 'standard'

n_layers - 3

drop_out - 0.2

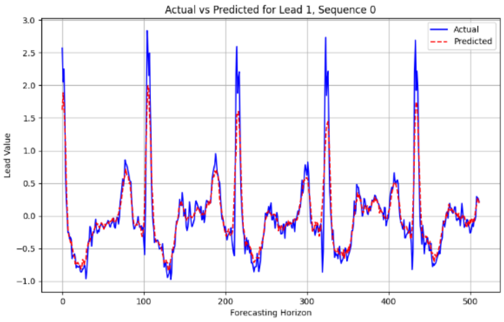
```
class Dataset_PTBXL(Dataset):
    def _init_(self, root_path, split='train', size=None,
                 features='M', data_path='expanded_form.csv',
```

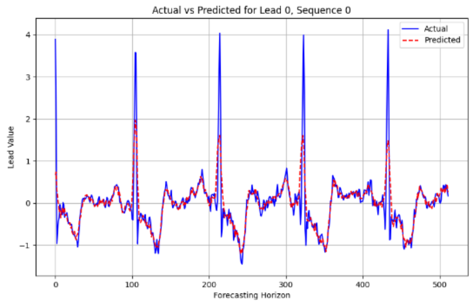
```
                 target='lead_0', scale=True, timeenc=0, freq='t',
                 use_time_features=False):
        # size [seq_len, label_len, pred_len]
        if size is None:
            self.seq_len = 512
            self.label_len = 256
            self.pred_len = 512
        else:
            self.seq_len = size[0]
            self.label_len = size[1]
            self.pred_len = size[2]

        assert split in ['train', 'test', 'val']
        type_map = {'train': 0, 'val': 1, 'test': 2}
        self.set_type = type_map[split]

        self.features = features
        self.target = target
        self.scale = scale
        self.timeenc = timeenc
        self.freq = freq
        self.use_time_features = use_time_features

        self.root_path = root_path
        self.data_path = data_path
        self._read_data_()

    def _read_data_(self):
        self.scaler = StandardScaler()
        df_raw = pd.read_csv(os.path.join(self.root_path, self.data_path))
        # Assuming 80% train, 10% val, 10% test split
        #data_leak consider
        total_ecgs = df_raw['ecg_id'].nunique()
        train_split = int(0.8 * total_ecgs)
        val_split = int(0.9 * total_ecgs)

        if self.set_type == 0:  # train
            df_data = df_raw[df_raw['ecg_id'] < train_split]
        elif self.set_type == 1:  # val
            df_data = df_raw[(df_raw['ecg_id'] >= train_split) &
(df_raw['ecg_id'] < val_split)]
        else:  # test
            df_data = df_raw[df_raw['ecg_id'] >= val_split]

        if self.features == 'M' or self.features == 'MS':
            cols_data = df_data.columns[5:]  #from 0-4 cols has static data
from col 5 all the lead values are taken for prediction.
            self.data = df_data[cols_data].values
```

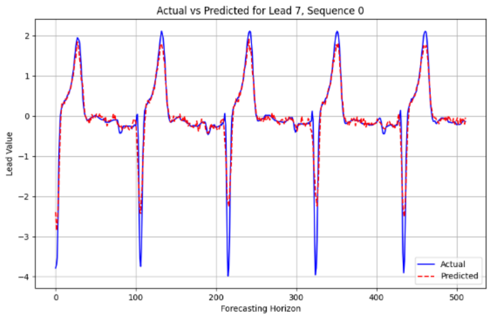
```
        elif self.features == 'S':
            self.data = df_data[[self.target]].values

        if self.scale:
            self.scaler.fit(self.data)
            self.data = self.scaler.transform(self.data)

        # Static features
        self.static_data = df_data[['patient_id', 'age', 'sex']].values

    def _getitem_(self, index):
        s_begin = index * self.seq_len
        s_end = s_begin + self.seq_len
        seq_x = self.data[s_begin:s_end]
        seq_y = self.data[s_begin:s_end]  # Same as seq_x for autoencoder-like
tasks
        static_features = self.static_data[index]

        return _torch(seq_x, seq_y, static_features)

    def _len_(self):
        return len(self.data) // self.seq_len

    def inverse_transform(self, data):
        return self.scaler.inverse_transform(data)
```

```
number of patches: 42
suggested lr: 0.0005214008287999684
number of patches: 42
          epoch     train_loss     valid_loss      valid_mse      valid_mae           time
Better model found at epoch 0 with valid_loss value: 0.3413420148386214.
              0       0.350412       0.341342       0.294470       0.341342          03:11
Better model found at epoch 1 with valid_loss value: 0.3363741537882932.
              1       0.282776       0.336374       0.290464       0.336374          03:18
Better model found at epoch 2 with valid_loss value: 0.3360969767163567.
              2       0.245426       0.336097       0.290136       0.336097          03:19
----------- Complete! -----------
```

```
number of patches: 42
Mean Squared Error : 0.23169969022274017
Mean Absolute Error : 0.18223978579044342
score: [array(0.23169969, dtype=float32), array(0.18223979, dtype=float32)]
shape: (4333, 512, 12)
```



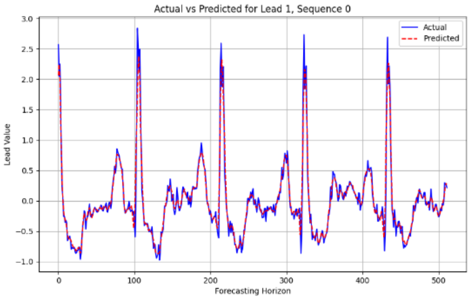


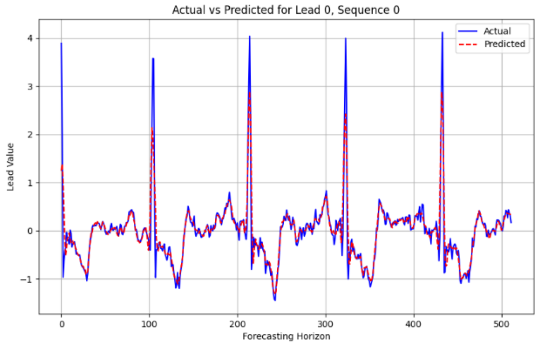


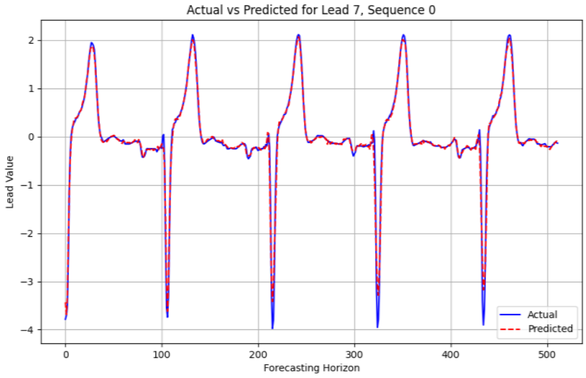
- **Model # 8 (xLSTMTime_cw512_tw512_patch12_stride12_epochs10_model1)**

```
number of patches: 42
suggested lr: 0.00014174741629268049
number of patches: 42
          epoch    train_loss     valid_loss      valid_mse      valid_mae           time
Better model found at epoch 0 with valid_loss value: 0.37884962379918646.
              0      0.461706       0.378850       0.636115       0.378850          03:59
Better model found at epoch 1 with valid_loss value: 0.2536268659037474.
              1      0.303038       0.253627       0.342350       0.253627          03:53
Better model found at epoch 2 with valid_loss value: 0.20673561052172282.
              2      0.215587       0.206736       0.240705       0.206736          04:04
Better model found at epoch 3 with valid_loss value: 0.1759277597300153.
              3      0.180243       0.175928       0.189917       0.175928          03:59
Better model found at epoch 4 with valid_loss value: 0.1567956681012722.
              4      0.158784       0.156796       0.159887       0.156796          04:05
Better model found at epoch 5 with valid_loss value: 0.14053334161972797.
              5      0.144197       0.140533       0.139423       0.140533          04:14
Better model found at epoch 6 with valid_loss value: 0.12856590117809544.
              6      0.134027       0.128566       0.126795       0.128566          04:08
Better model found at epoch 7 with valid_loss value: 0.12243929840306225.
              7      0.127413       0.122439       0.121297       0.122439          04:05
Better model found at epoch 8 with valid_loss value: 0.1199722272431413.
              8      0.123773       0.119972       0.119005       0.119972          03:57
Better model found at epoch 9 with valid_loss value: 0.11947969198502048.
              9      0.122437       0.119480       0.118637       0.119480          04:09
----------- Complete! -----------
```

```
number of patches: 42
Mean Squared Error : 0.11863684654235 84
Mean Absolute Error : 0.11947967112064362
score: [array(0.11863685, dtype=float32), array(0.11947967, dtype=float32)]
shape: (4333, 512, 12)
```







- **Model # 5 (xLSTMTime_cw900_tw100_patch24_stride24_epochs20_model5)**

ECG data for all patients with all Leads and 100 Hz in each second and 10 seconds for one patient therefore 1000 time steps.

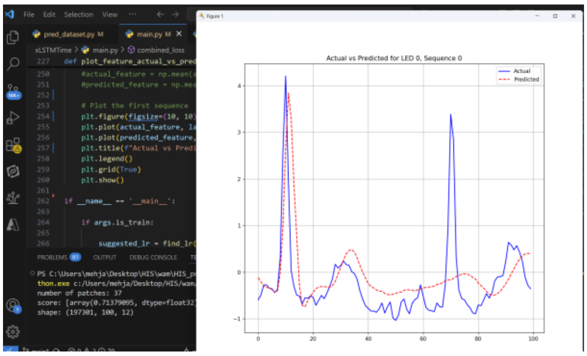
```
args = {
    'context_points': 900,        # Sequence Length (you may adjust based on data)
    'target_points': 100,         # Forecast Horizon
    'batch_size': 32,             # Batch Size, adjust based on memory
    'lr': 1e-3,                   # Learning Rate (lowered for deeper model)
    'n_layers': 4,                # Increase the number of layers to capture more complexity
    'd_model': 256,               # Hidden Size (increase for more capacity)
    'dropout': 0.2,               # Dropout Rate (increased for better regularization)
    'patch_len': 24,              # Patch Length (adjusted for larger data)
    'stride': 24,                 # Stride Between Patches (adjusted for patch length)
    'epochs': 20,                 # number of epochs
}
```

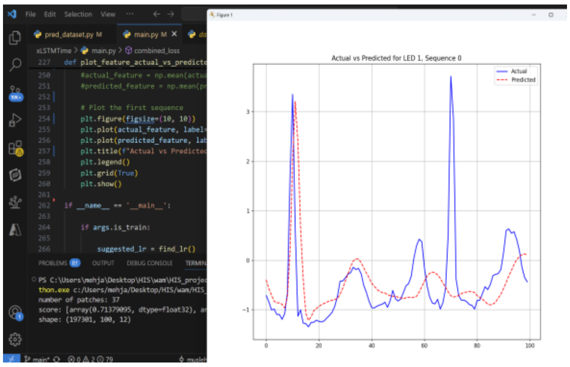
```
class Dataset_PTBXL(Dataset):
   def __init__(self, root_path, split='train', size=None,
             features='M',
data_path='C:/Users/mehja/Desktop/HIS/wam/HIS_project/xLSTMTime_MK/data/illness-20250129T151711Z-001/illness/final_patient_data_Copy.csv',
             target='', scale=True, timeenc=0, freq='s',
             time_col_name='date', use_time_features=False,
             train_split=0.7, test_split=0.2
```

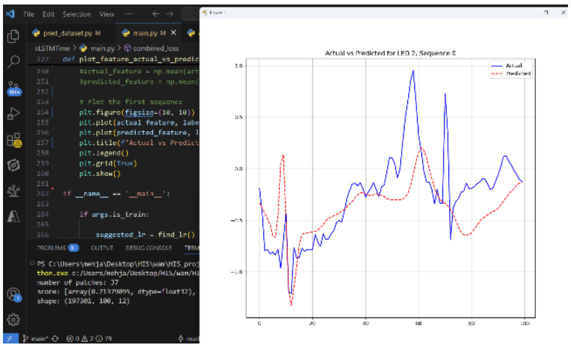
```
        ):
    # size [seq_len, label_len, pred_len]
    # info
    if size == None:
        self.seq_len = 100
        self.label_len = 12
        self.pred_len = 50
    else:
            self.seq_len = size[0]
            self.label_len = size[1]
            self.pred_len = size[2]
        # init
        assert split in ['train', 'test', 'val']
        type_map = {'train': 0, 'val': 1, 'test': 2}
        self.set_type = type_map[split]

        self.features = features
        self.target = target
        self.scale = scale
        self.timeenc = timeenc
        self.freq = freq
        self.time_col_name = time_col_name
        self.use_time_features = use_time_features

        # train test ratio
        self.train_split, self.test_split = train_split, test_split

        self.root_path = root_path
        self.data_path = data_path
        self.__read_data__()

    def __read_data__(self):
        self.scaler = StandardScaler()
        df_raw = pd.read_csv(os.path.join(self.root_path,
                                          self.data_path))

        '''
        df_raw.columns: [time_col_name, ...(other features), target feature]
        '''
        cols = list(df_raw.columns)
        #cols.remove(self.target) if self.target
        #cols.remove(self.time_col_name)
        #df_raw = df_raw[[self.time_col_name] + cols + [self.target]]

        num_train = int(len(df_raw) * self.train_split)
        num_test = int(len(df_raw) * self.test_split)
        num_vali = len(df_raw) - num_train - num_test
```

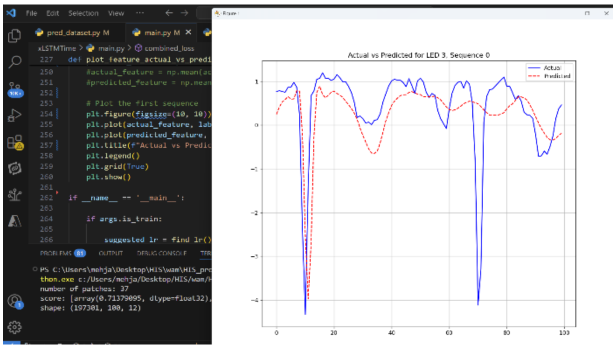
```
        border1s = [0, num_train - self.seq_len, len(df_raw) - num_test -
self.seq_len]
        border2s = [num_train, num_train + num_vali, len(df_raw)]
        border1 = border1s[self.set_type]
        border2 = border2s[self.set_type]

        if self.features == 'M' or self.features == 'MS':
            cols_data = df_raw.columns[5:] #takes features from 5th column
becuase LED column starts
            df_data = df_raw[cols_data]
        elif self.features == 'S':
            df_data = df_raw[[self.target]]

        if self.scale:
            train_data = df_data[border1s[0]:border2s[0]]
            self.scaler.fit(train_data.values)
            data = self.scaler.transform(df_data.values)
        else:
            data = df_data.values

        df_stamp = df_raw[[self.time_col_name]][border1:border2]
        df_stamp[self.time_col_name] =
pd.to_datetime(df_stamp[self.time_col_name])

        self.data_x = data[border1:border2]
        self.data_y = data[border1:border2]
        #self.data_stamp = data_stamp

    def __getitem__(self, index):
        s_begin = index
        s_end = s_begin + self.seq_len
        r_begin = s_end - self.label_len
        r_end = r_begin + self.label_len + self.pred_len

        seq_x = self.data_x[s_begin:s_end]
        seq_y = self.data_y[r_begin:r_end]
        #seq_x_mark = self.data_stamp[s_begin:s_end]
        #seq_y_mark = self.data_stamp[r_begin:r_end]

        if self.use_time_features: return _torch(seq_x, seq_y) #removed
seq_x_mark and seq_y_mark
        else: return _torch(seq_x, seq_y)

    def __len__(self):
        return len(self.data_x) - self.seq_len - self.pred_len + 1

    def inverse_transform(self, data):
        return self.scaler.inverse_transform(data)
```

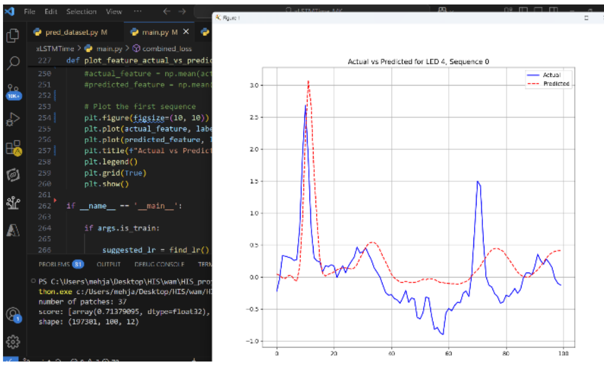
Output

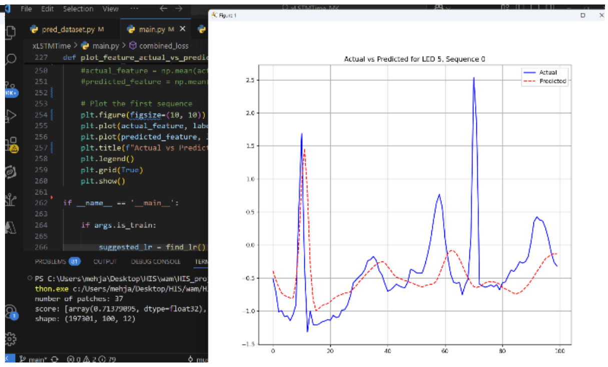

Actual vs Predicted for LED 0, Sequence 0
Actual
Predicted

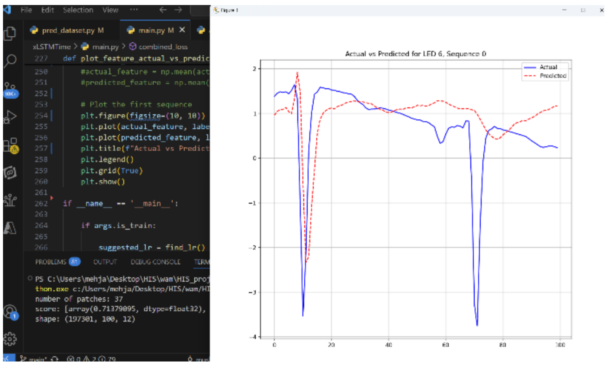

Actual vs Predicted for LED 1, Sequence 0
Actual
Predicted

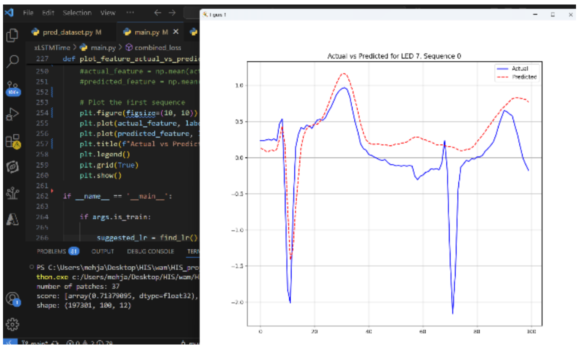

Actual vs Predicted for LED 2, Sequence 0

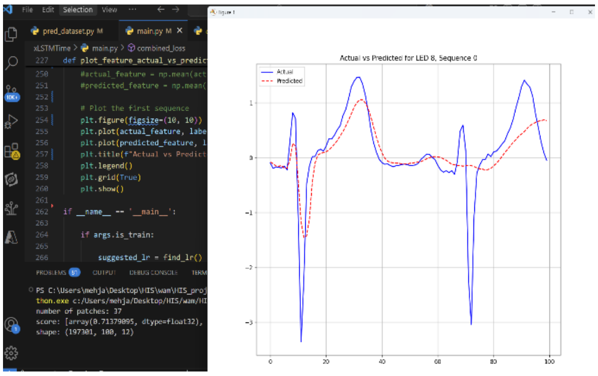

Actual vs Predicted for LED 3, Sequence 0

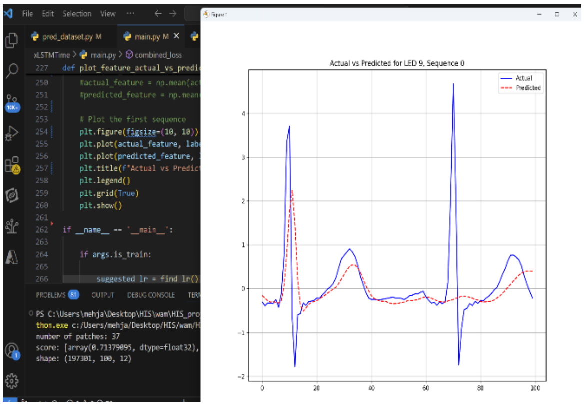

Actual vs Predicted for LED 4, Sequence 0
Actual
Predicted

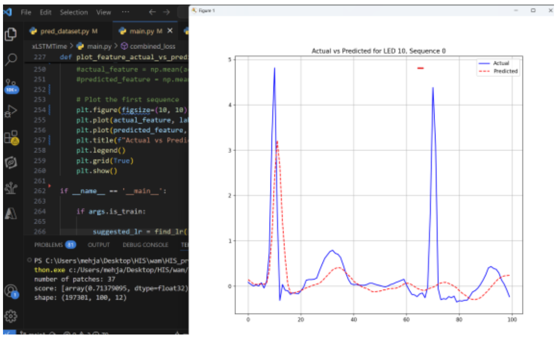

Actual vs Predicted for LED 5, Sequence 0
Actual
Predicted


Actual vs Predicted for LED 6, Sequence 0
Actual
Predicted


Actual vs Predicted for LED 7, Sequence 0



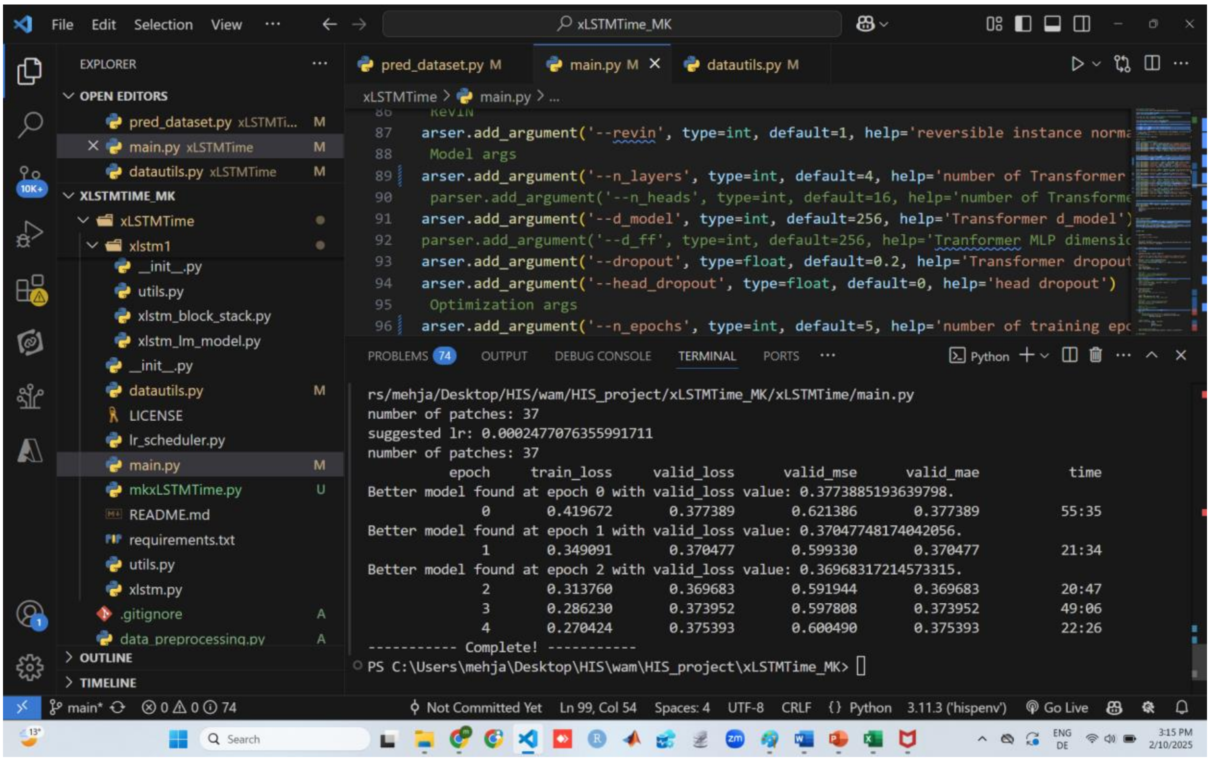




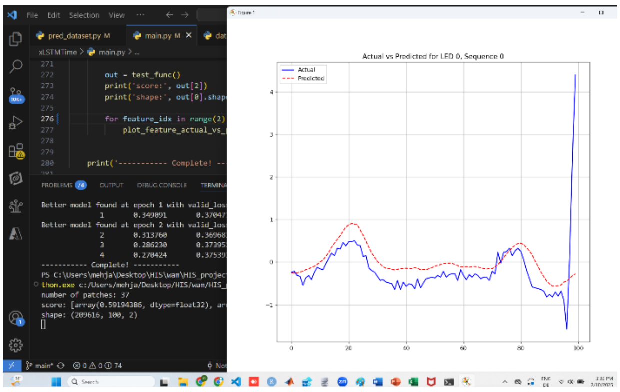
Model # 6:

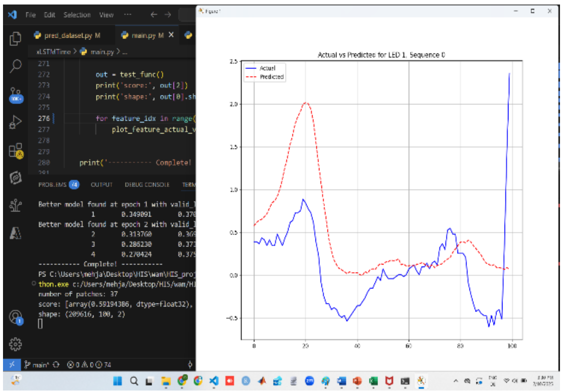
Dataset: MIT-BIH Long-Term ECG Database.
The MIT-BIH Long-Term ECG Database is a collection of 7 long-duration electrocardiogram (ECG) recordings (14 to 22 hours each), with manually reviewed beat annotations.

- All the parameters are exactly the same as for above model.


File Edit Selection View
xLSTMTime_MK
EXPLORER
OPEN EDITORS
pred_dataset.py xLSTMTi... M
main.py xLSTMTime M
datautils.py xLSTMTime M
XLSTMTIME_MK
xLSTMTime
xlstm1
init.py
utils.py
xlstm_block_stack.py
xlstm_lm_model.py
init.py
datautils.py M
LICENSE
lr_scheduler.py
main.py M
mkxLSTMTime.py U
README.md
requirements.txt
utils.py
xlstm.py
.gitignore A
data_preprocessing.py A
OUTLINE
TIMELINE
pred_dataset.py M
main.py M
datautils.py M
xLSTMTime > main.py > ...
arser.add_argument('--revin', type=int, default=1, help='reversible instance norma
Model args
arser.add_argument('--n_layers', type=int, default=4, help='number of Transformer
parser.add_argument('--n_heads', type=int, default=16, help='number of Transforme
arser.add_argument('--d_model', type=int, default=256, help='Transformer d_model')
parser.add_argument('--d_ff', type=int, default=256, help='Tranformer MLP dimensic
arser.add_argument('--dropout', type=float, default=0.2, help='Transformer dropout
arser.add_argument('--head_dropout', type=float, default=0, help='head dropout')
Optimization args
arser.add_argument('--n_epochs', type=int, default=5, help='number of training epc
PROBLEMS 74 OUTPUT DEBUG CONSOLE TERMINAL PORTS
rs/mehja/Desktop/HIS/wam/HIS_project/xLSTMTime_MK/xLSTMTime/main.py
number of patches: 37
suggested lr: 0.0002477076355991711
number of patches: 37
epoch train_loss valid_loss valid_mse valid_mae time
Better model found at epoch 0 with valid_loss value: 0.3773885193639798.
0 0.419672 0.377389 0.621386 0.377389 55:35
Better model found at epoch 1 with valid_loss value: 0.37047748174042056.
1 0.349091 0.370477 0.599330 0.370477 21:34
Better model found at epoch 2 with valid_loss value: 0.36968317214573315.
2 0.313760 0.369683 0.591944 0.369683 20:47
3 0.286230 0.373952 0.597808 0.373952 49:06
4 0.270424 0.375393 0.600490 0.375393 22:26
----------- Complete! -----------
PS C:\Users\mehja\Desktop\HIS\wam\HIS_project\xLSTMTime_MK>
main* Not Committed Yet Ln 99, Col 54 Spaces: 4 UTF-8 CRLF Python 3.11.3 ('hispenv') Go Live
3:15 PM 2/10/2025

Output:


Actual vs Predicted for LED 0, Sequence 0
Actual
Predicted


Actual vs Predicted for LED 1, Sequence 0
Actual
Predicted