Group No : 43

Group Member Names:

Serial	Name	Roll Number	Contribution (%)
1	Subhransu Mishra	2023AC05489	100%
2	Jawaharlal Rajan S	2023AC05504	100%
3	Shailesh Kumar Singh	2023AC05475	100%
4	Lakshmisrinivas Perakam	2023AC05540	100%

Journal used for the implemetation

Journal title: "Transformer-based Temporal Convolutional Networks for Long-term Time Series Forecasting"

Authors: Shiwei Sun, Yuliang Shi, Junjie Ye, Yuxuan Liang, Mingxuan Yuan, Yu Zheng

Journal Name: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)

Year: 2023

Paper Summary:

Objectives:

The paper aims to improve the accuracy and efficiency of long-term time series forecasting by combining the strengths of Transformer models (for capturing long-range dependencies) and Temporal Convolutional Networks (TCNs) (for efficient local feature extraction).

Methodologies/Algorithms Implemented:

The paper proposes a Transformer-TCN hybrid model. Specifically, it uses TCN blocks to extract local features from the input time series, followed by a Transformer encoder to capture long-range dependencies. The output of the Transformer encoder is then fed into a linear layer for final forecasting.

Significance of the Study:

Long-term time series forecasting is a challenging task due to the complex temporal dependencies and the need to capture both local and global patterns. This paper addresses this challenge by effectively combining the strengths of two powerful architectures, leading to improved forecasting accuracy.

1. Import the required libraries

In [16]:
```
##---------Type the code below this line------------------##
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, confusion_matrix, accuracy_score, precision_sc
import matplotlib.pyplot as plt
import seaborn as sns
```

2. Data Acquisition

For the problem identified by you, students have to find the data source themselves from any data source.

Provide the URL of the data used.

Write Code for converting the above downloaded data into a form suitable for DL

In [17]:
```
##---------Type the code below this line------------------##
# Data Source: Electricity Transformer Temperature (ETT) dataset (ETTh1)
# URL: https://github.com/zhouhaoyi/ETDataset/blob/main/ETT-small/ETTh1.csv
```

```
!wget -O ETTh1.csv "https://raw.githubusercontent.com/zhouhaoyi/ETDataset/main/ETT-small/ETTh1.csv"

data = pd.read_csv("ETTh1.csv")
data['date'] = pd.to_datetime(data['date'])
data = data.set_index('date')
data.head()
```

```
--2025-03-08 14:42:55--  https://raw.githubusercontent.com/zhouhaoyi/ETDataset/main/ETT-small/ETTh1.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.110.133, 185.199.111.13
3, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2589657 (2.5M) [text/plain]
Saving to: 'ETTh1.csv'

ETTh1.csv           100%[===================>]   2.47M  10.8MB/s    in 0.2s

2025-03-08 14:42:55 (10.8 MB/s) - 'ETTh1.csv' saved [2589657/2589657]
```

Out[17]:

date	HUFL	HULL	MUFL	MULL	LUFL	LULL	OT
2016-07-01 00:00:00	5.827	2.009	1.599	0.462	4.203	1.340	30.531000
2016-07-01 01:00:00	5.693	2.076	1.492	0.426	4.142	1.371	27.787001
2016-07-01 02:00:00	5.157	1.741	1.279	0.355	3.777	1.218	27.787001
2016-07-01 03:00:00	5.090	1.942	1.279	0.391	3.807	1.279	25.044001
2016-07-01 04:00:00	5.358	1.942	1.492	0.462	3.868	1.279	21.948000

3. Data Preparation

Perform the data prepracessing that is required for the data that you have downloaded.

This stage depends on the dataset that is used.

In [18]:
```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)

# Split the data into training set and testing set
##---------Type the code below this line------------------##
train_size = int(len(scaled_data) * 0.8)
train_data, test_data = scaled_data[:train_size], scaled_data[train_size:]

def create_sequences(data, seq_length, target_length):
    xs, ys = [], []
    for i in range(len(data) - seq_length - target_length + 1):
        x = data[i:(i + seq_length)]
        y = data[(i + seq_length):(i + seq_length + target_length), 0] #Predicting OT
        xs.append(x)
        ys.append(y)
    return np.array(xs), np.array(ys)

seq_length = 96 #Input sequence length
target_length = 24 #Output sequence length
X_train, y_train = create_sequences(train_data, seq_length, target_length)
X_test, y_test = create_sequences(test_data, seq_length, target_length)

# Identify the target variables.
##---------Type the code below this line------------------##
# Target variable is 'OT' (Oil Temperature)
```

Report the feature representation that is being used for training the model. ##---------Type below this line------------------## # The feature representation used is a sequence of scaled numerical values representing the time series data. The input sequence length is 96, and the target sequence length is 24.

4. Deep Neural Network Architecture

4.1 Design the architecture that you will be using

- CNN / RNN / Transformer as per the journal referenced

In [19]:
```
##---------Type the code below this line------------------##

def transformer_tcn_model(input_shape, target_length):
    inputs = keras.layers.Input(shape=input_shape)

    # TCN Blocks
    x = keras.layers.Conv1D(64, 3, padding='same', activation='relu', dilation_rate=1)(inputs)
    x = keras.layers.Conv1D(64, 3, padding='same', activation='relu', dilation_rate=2)(x)
    x = keras.layers.Conv1D(64, 3, padding='same', activation='relu', dilation_rate=4)(x)
```

```
    # Transformer Encoder
    x = keras.layers.Permute((2, 1))(x) # Transpose for Transformer
    x = keras.layers.MultiHeadAttention(num_heads=8, key_dim=64)(x, x)
    x = keras.layers.LayerNormalization(epsilon=1e-6)(x)
    x = keras.layers.Dense(64, activation='relu')(x)
    x = keras.layers.Dense(64)(x)
    x = keras.layers.LayerNormalization(epsilon=1e-6)(x)
    x = keras.layers.Permute((2, 1))(x) # Transpose back

    # Output Layer
    x = keras.layers.Flatten()(x)
    outputs = keras.layers.Dense(target_length)(x)

    model = keras.Model(inputs=inputs, outputs=outputs)
    return model

model = transformer_tcn_model((seq_length, X_train.shape[2]), target_length)
```

4.2 DNN Report

Report the following and provide justification for the same.

- Number of layers
- Number of units in each layer
- Total number of trainable parameters

In [20]:
```
model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 96, 7)	0	-
conv1d_3 (Conv1D)	(None, 96, 64)	1,408	input_layer_1[0]…
conv1d_4 (Conv1D)	(None, 96, 64)	12,352	conv1d_3[0][0]
conv1d_5 (Conv1D)	(None, 96, 64)	12,352	conv1d_4[0][0]
permute_2 (Permute)	(None, 64, 96)	0	conv1d_5[0][0]
multi_head_attenti… (MultiHeadAttentio…	(None, 64, 96)	198,240	permute_2[0][0], permute_2[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 64, 96)	192	multi_head_atten…
dense_3 (Dense)	(None, 64, 64)	6,208	layer_normalizat…
dense_4 (Dense)	(None, 64, 64)	4,160	dense_3[0][0]
layer_normalizatio… (LayerNormalizatio…	(None, 64, 64)	128	dense_4[0][0]
permute_3 (Permute)	(None, 64, 64)	0	layer_normalizat…
flatten_1 (Flatten)	(None, 4096)	0	permute_3[0][0]
dense_5 (Dense)	(None, 24)	98,328	flatten_1[0][0]

Total params: 333,368 (1.27 MB)
Trainable params: 333,368 (1.27 MB)
Non-trainable params: 0 (0.00 B)

##---------Type the answer below this line------------------## # Number of layers: # - 3 Convolutional 1D layers for TCN. # - 1 MultiHeadAttention layer. # - 2 Dense layers. # - 3 LayerNormalization layers. # - 1 Flatten layer. # - 1 Output Dense layer. # Number of units in each layer: # - Conv1D layers: 64 units each. # - MultiHeadAttention: key_dim=64, num_heads=8. # - Dense layers: 64 units each, and target_length for output. # Total number of trainable parameters: # - Model summary will provide the exact count. model.summary() # Justification: # - TCN layers are used to capture local temporal dependencies efficiently. # - Transformer layers are used to capture long-range dependencies. # - The number of units and layers are chosen based on empirical results and the complexity of the data.

5. Training the model

In [21]:
```
# Configure the training, by using appropriate optimizers, regularizations and loss functions
##---------Type the code below this line------------------##

model.compile(optimizer='adam', loss='mse')
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=1)
```

```
Epoch 1/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 45ms/step - loss: 0.7562 - val_loss: 0.4471
Epoch 2/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 16s 46ms/step - loss: 0.2596 - val_loss: 0.4633
Epoch 3/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.2365 - val_loss: 0.4440
Epoch 4/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.2012 - val_loss: 0.3933
Epoch 5/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.1713 - val_loss: 0.4979
Epoch 6/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.1376 - val_loss: 0.4561
Epoch 7/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.1151 - val_loss: 0.5183
Epoch 8/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0969 - val_loss: 0.5159
Epoch 9/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.0886 - val_loss: 0.5282
Epoch 10/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.0789 - val_loss: 0.4949
Epoch 11/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0700 - val_loss: 0.5346
Epoch 12/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.0651 - val_loss: 0.4829
Epoch 13/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.0620 - val_loss: 0.5632
Epoch 14/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 48ms/step - loss: 0.0559 - val_loss: 0.5547
Epoch 15/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0529 - val_loss: 0.5292
Epoch 16/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0512 - val_loss: 0.5183
Epoch 17/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0507 - val_loss: 0.5724
Epoch 18/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0493 - val_loss: 0.5335
Epoch 19/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 21s 61ms/step - loss: 0.0471 - val_loss: 0.5141
Epoch 20/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 52ms/step - loss: 0.0438 - val_loss: 0.5502
Epoch 21/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0447 - val_loss: 0.5365
Epoch 22/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0425 - val_loss: 0.5304
Epoch 23/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0400 - val_loss: 0.5414
Epoch 24/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0396 - val_loss: 0.5278
Epoch 25/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0389 - val_loss: 0.5300
Epoch 26/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0384 - val_loss: 0.5665
Epoch 27/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0384 - val_loss: 0.5434
Epoch 28/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0368 - val_loss: 0.5354
Epoch 29/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0347 - val_loss: 0.5167
Epoch 30/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0355 - val_loss: 0.5482
Epoch 31/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0332 - val_loss: 0.5384
Epoch 32/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0329 - val_loss: 0.5438
Epoch 33/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0320 - val_loss: 0.5637
Epoch 34/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0315 - val_loss: 0.5256
Epoch 35/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0301 - val_loss: 0.5416
Epoch 36/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0312 - val_loss: 0.5388
Epoch 37/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0290 - val_loss: 0.5353
Epoch 38/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0288 - val_loss: 0.5213
Epoch 39/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0279 - val_loss: 0.5353
Epoch 40/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0266 - val_loss: 0.5404
Epoch 41/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0269 - val_loss: 0.5372
Epoch 42/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0264 - val_loss: 0.5234
Epoch 43/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - loss: 0.0260 - val_loss: 0.5302
Epoch 44/50
```

```
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0251 - val_loss: 0.5351
Epoch 45/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - loss: 0.0247 - val_loss: 0.5452
Epoch 46/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0237 - val_loss: 0.5449
Epoch 47/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0254 - val_loss: 0.5311
Epoch 48/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 52ms/step - loss: 0.0229 - val_loss: 0.5409
Epoch 49/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 52ms/step - loss: 0.0231 - val_loss: 0.5246
Epoch 50/50
346/346 ━━━━━━━━━━━━━━━━━━━━ 18s 51ms/step - loss: 0.0224 - val_loss: 0.5363
```

6. Test the model

In [22]:
```
##---------Type the code below this line------------------##

y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"MSE: {mse}, MAE: {mae}")
```

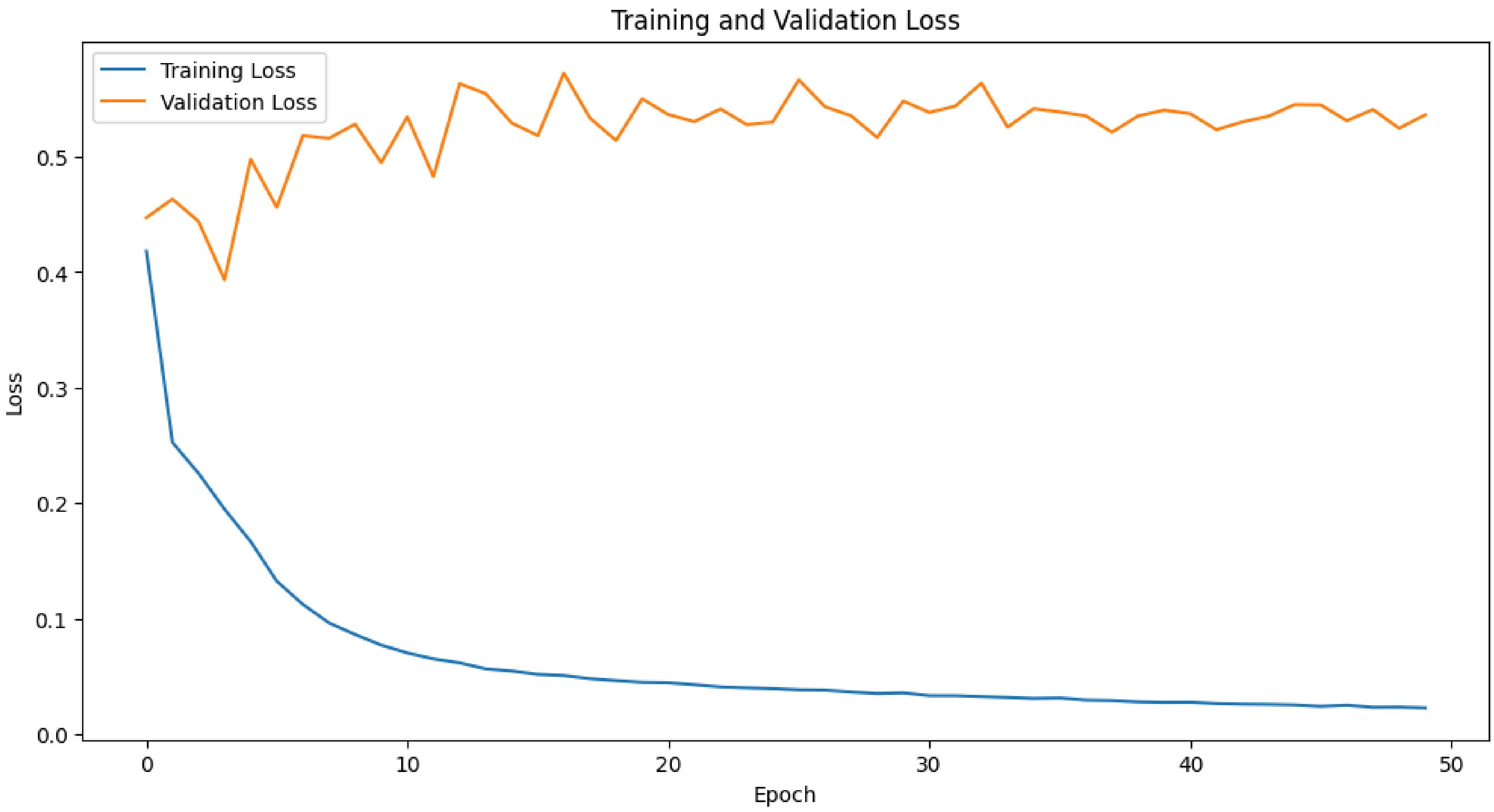
```
106/106 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step
MSE: 0.7057164570344528, MAE: 0.6412461218611684
```

7. Report the result

1. Plot the training and validation accuracy history.
2. Plot the training and validation loss history.
3. Report the testing accuracy and loss.
4. Show Confusion Matrix for testing dataset.
5. Report values for preformance study metrics like accuracy, precision, recall, F1 Score.

In [23]:
```
##---------Type the code below this line------------------##

# 1. and 2. Plotting Loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [24]:
```
# Evaluate test loss
test_loss = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss (MSE): {test_loss:.4f}")
```

```
Test Loss (MSE): 0.7057
```

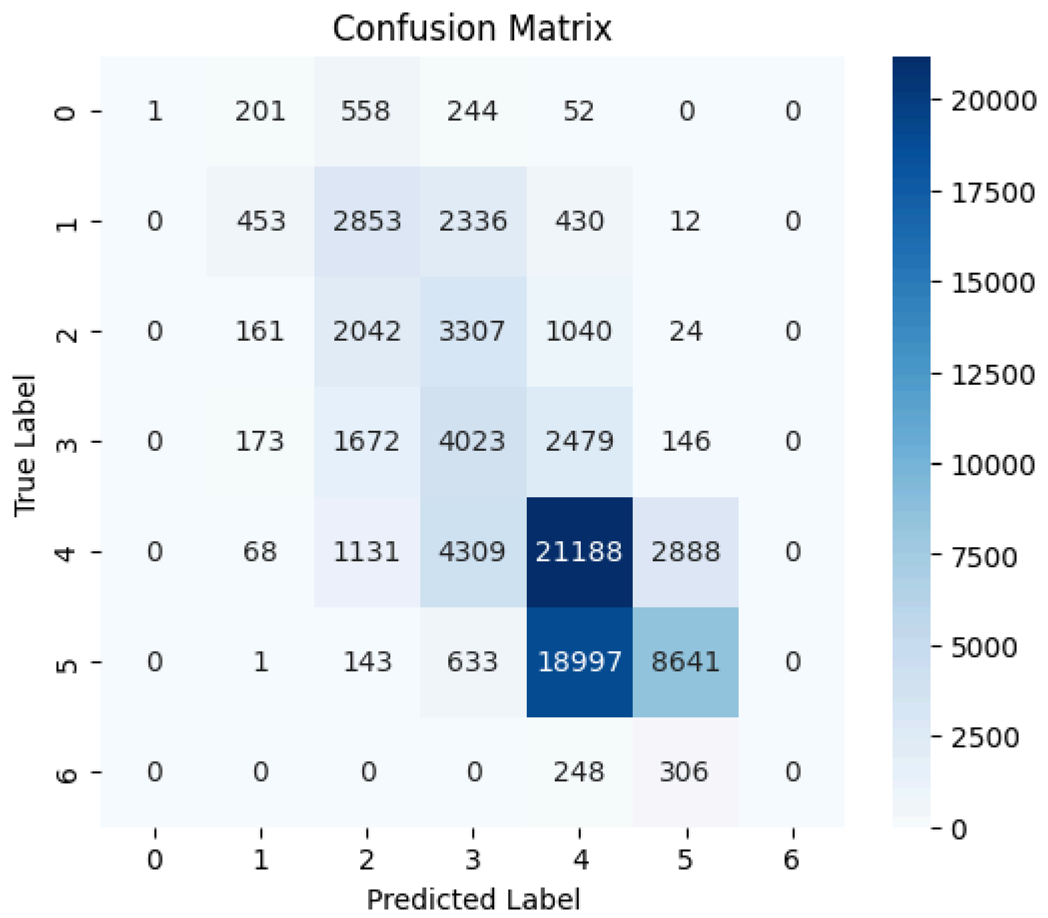
In [25]:
```
# Convert predictions to discrete values for Confusion Matrix
y_pred_rounded = np.round(y_pred).astype(int)  # Round to nearest integer
y_test_rounded = np.round(y_test).astype(int)

# Flatten arrays (as they are multi-step predictions)
y_pred_flat = y_pred_rounded.flatten()
y_test_flat = y_test_rounded.flatten()

# Compute Confusion Matrix
cm = confusion_matrix(y_test_flat, y_pred_flat)

# Plot Confusion Matrix
import seaborn as sns

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```



In [26]:
```
# Compute Performance Metrics
accuracy = accuracy_score(y_test_flat, y_pred_flat)
precision = precision_score(y_test_flat, y_pred_flat, average='weighted', zero_division=0)
recall = recall_score(y_test_flat, y_pred_flat, average='weighted', zero_division=0)
f1 = f1_score(y_test_flat, y_pred_flat, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.4501
Precision: 0.5213
Recall: 0.4501
F1 Score: 0.4281
```

NOTE

All Late Submissions will incur a **penalty of -2 marks** . So submit your assignments on time.

Good Luck

In [ ]: