

FINAL_gru_discrete_DAiA

February 2, 2023

1 Model Building

1.1 Imports

```
[1]: # Imports
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import *
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.losses import *
from tensorflow.keras.metrics import MeanSquaredError
from tensorflow.keras.optimizers import Adam
import tensorflow as tf
import tensorflow.keras as K

from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score
```

Set Graphic Resolution

```
[2]: plt.rcParams['figure.dpi'] = 300
plt.rcParams['savefig.dpi'] = 300
```

Read In the different Datasets

```
[53]: df = pd.read_excel('DiscreteMotion_Data_Horizontalsetup.xlsx')
```

Set the dataframes' time index to a 45000 entity long datetime64 range separated by 10 ms

```
[4]: time_index = np.arange('1970-01-01T00:00:00.000', '1970-01-01T00:07:30.
                           ↵000', dtype='datetime64[10ms]')
df['time_index'] = time_index
df.set_index('time_index', inplace=True)
```

```
[5]: df.describe()
```

```
[5]:          Pow_100      Pow_200      Pow_300      Pow_400      Pow_500  \
count  45000.000000  45000.000000  45000.000000  45000.000000  45000.000000
mean   0.843142     1.249023     1.604354     1.951205     2.206332
std    0.253721     0.579047     0.920542     1.298846     1.663406
min   0.000000     0.000000     0.000000     0.000000     0.000000
25%   0.834000     1.332000     0.531000     0.294000     0.264000
50%   0.900000     1.458000     2.040000     2.625000     3.204000
75%   0.960000     1.575000     2.210000     2.896000     3.553000
max   1.316000     2.360000     3.528000     5.100000     6.380000

          Pow_600      Pow_700      Pow_800      Pow_900      Pow_1000  \
count  45000.000000  45000.000000  45000.000000  45000.000000  45000.000000
mean   2.493221     2.730921     2.848026     3.118614     3.295054
std    2.058099     2.446147     2.785256     3.201309     3.574365
min   0.000000     0.000000     0.000000     0.000000     0.000000
25%   0.265500     0.264000     0.244000     0.258000     0.264000
50%   3.738000     2.873000     1.220000     0.828000     0.735000
75%   4.242000     4.950000     5.655000     6.498750     7.310000
max   7.608000     9.099999     9.641000    10.642000    11.375999

          Pow_1100     Pow_1200     Pow_1300     Pow_1400     Pow_1500  \
count  45000.000000  45000.000000  45000.000000  45000.000000  45000.000000
mean   3.453644     3.577217     3.732571     3.902852     3.908951
std    3.955490     4.300227     4.628946     5.014324     5.244952
min   0.000000     0.000000     0.000000     0.000000     0.000000
25%   0.264000     0.238000     0.234000     0.256000     0.243000
50%   0.606000     0.516000     0.458000     0.417000     0.486000
75%   8.103001     8.639999     8.649000     8.613000     7.670001
max   12.558001    13.631000    14.899000    16.224001    17.237999

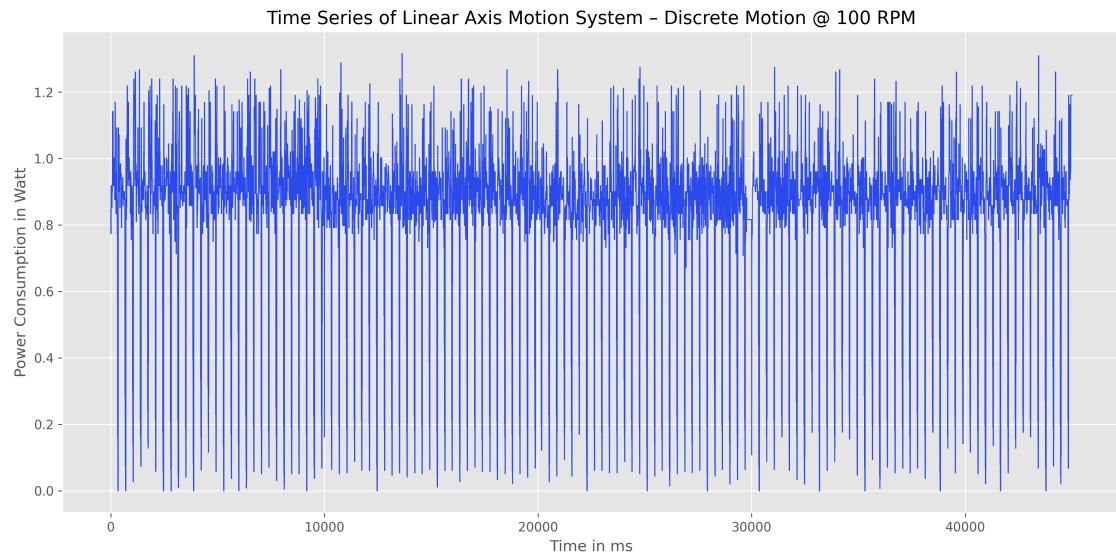
          Pow_1600     Pow_1700     Pow_1800     Pow_1900     Pow_2000
count  45000.000000  45000.000000  45000.000000  45000.000000  45000.000000
mean   4.052579     4.156900     4.182729     4.218840     4.217175
std    5.600147     5.836061     5.949107     6.142617     6.150047
min   0.000000     0.000000     0.000000     0.000000     0.000000
25%   0.255000     0.273000     0.256000     0.255000     0.270000
50%   0.434500     0.426000     0.405000     0.417000     0.447000
75%   7.525000     7.524000     7.560000     7.232000     7.175000
max   19.139999    20.235001    22.019999    23.808001    23.487999
```

```
[58]: fig = plt.figure(figsize=(15, 7))
plt.style.use("ggplot")
plt.plot(df.Pow_100, color="#2a4ae9", linewidth=0.77)
# Set the plot title and axis labels
```

```

plt.title('Time Series of Linear Axis Motion System - Discrete Motion @ 100 RPM')
plt.xlabel('Time in ms')
plt.ylabel('Power Consumption in Watt')
plt.show()

```



1.2 Defining Functions

Split the data according into Train Data (80%) , Validation Data (10%) and Test Data (10%)

```
[6]: def split_data(df):
    train = df[:36000]
    val = df[36000:40500]
    test = df[40500:]
    return train, val, test
```

Normalize the Data in a range between 0 and 1

```
[7]: def scale_data(train, val, test):
    scaler = MinMaxScaler(feature_range=(0,1))
    train_scaled = scaler.fit_transform(train)
    val_scaled = scaler.transform(val)
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, val_scaled, test_scaled
```

function to create 3D arrays with certain window sizes for X_input and Y_output (**Window-Size Approach**)

```
[8]: def to_supervised(sequence, window_size, forecast_horizon):
    X, y = [], []
    for i in range(len(sequence) - window_size - forecast_horizon + 1):
        X.append(sequence[i:i+window_size])
        y.append(sequence[i+window_size:i+window_size+forecast_horizon])
    return np.array(X), np.array(y)
```

1.3 Data Preparation

Split the dataframe into Train-, Val- and Test Data

```
[9]: train, val, test = split_data(df)
```

Scale the Data between 0 and 1 using the MinMaxScaler

```
[10]: scaler, train, val, test = scale_data(train, val, test)
```

Prepare the input for supervised learning (using a **window-size of 40** and a **forecast-horizon of 20**)

```
[11]: X_train, y_train = to_supervised(train, window_size=40, forecast_horizon=20)
X_val, y_val = to_supervised(val, window_size=40, forecast_horizon=20)
X_test, y_test = to_supervised(test, window_size=40, forecast_horizon=20)
```

Check the shape of the Training Data

```
[12]: X_train.shape
```

```
[12]: (35941, 40, 20)
```

1.4 Build GRU Model

1.4.1 Model Summary

```
[13]: model = Sequential()
model.add(GRU(128, input_shape=(40, 20), activation='relu'))
model.add(RepeatVector(20))
model.add(GRU(64, return_sequences=True))
model.add(TimeDistributed(Dense(32)))
model.add(Dense(20))
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 128)	57600
repeat_vector (RepeatVector)	(None, 20, 128)	0

```
)  
  
gru_1 (GRU)           (None, 20, 64)      37248  
  
time_distributed (TimeDistr (None, 20, 32)      2080  
ibuted)  
  
dense_1 (Dense)       (None, 20, 20)       660  
  
=====  
Total params: 97,588  
Trainable params: 97,588  
Non-trainable params: 0
```

```
[14]: tf.keras.utils.plot_model(model, show_shapes=True, show_dtype=True, dpi=300)
```

```
[14]:
```

gru_input	input:	[None, 40, 20]
InputLayer		
float32	output:	[None, 40, 20]

↓

gru	input:	(None, 40, 20)
GRU		
float32	output:	(None, 128)

↓

repeat_vector	input:	(None, 128)
RepeatVector		
float32	output:	(None, 20, 128)

↓

gru_1	input:	(None, 20, 128)
GRU		
float32	output:	(None, 20, 64)

↓

time_distributed(dense)	input:	(None, 20, 64)
TimeDistributed(Dense)		
float32	output:	(None, 20, 32)

↓

dense_1	input:	(None, 20, 32)
Dense		
float32	output:	(None, 20, 20) ⁶

1.4.2 Compile the GRU Model

```
[15]: model.compile(loss='mae', optimizer=Adam(learning_rate=0.001), metrics=['mse'])
```

Create Callback and Fit the Training Dataset

```
[16]: stop_early_gru = K.callbacks.EarlyStopping(monitor='val_loss', mode='min',  
    verbose=1, patience=5)  
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_val,  
    y_val), callbacks=[stop_early_gru])
```

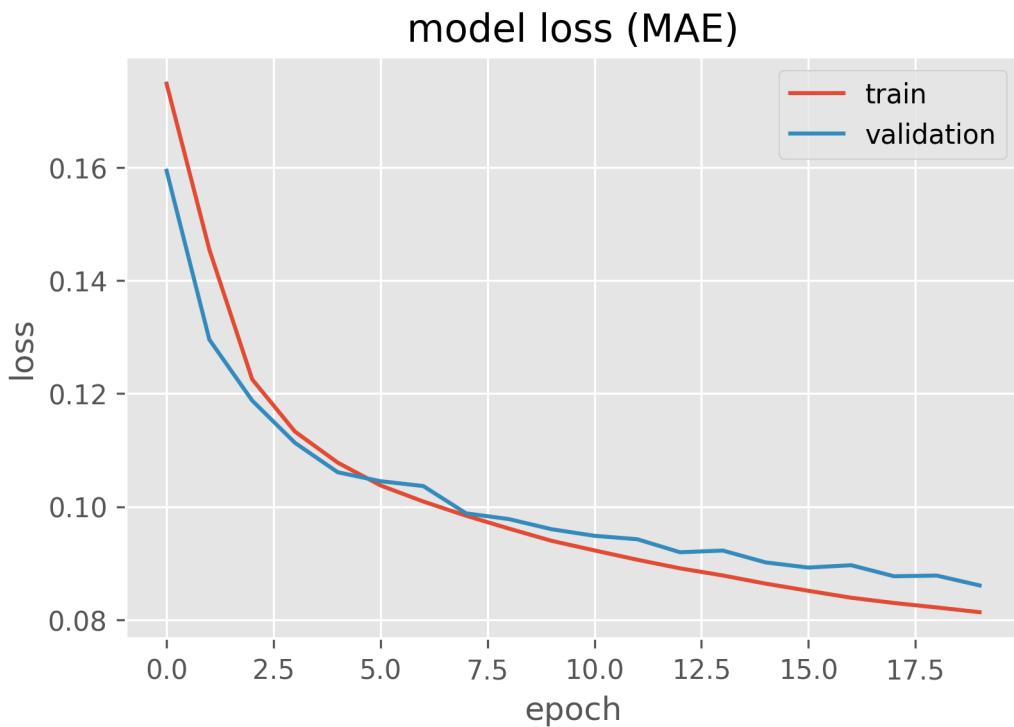
```
Epoch 1/20  
1124/1124 [=====] - 66s 55ms/step - loss: 0.1748 - mse:  
0.0753 - val_loss: 0.1595 - val_mse: 0.0601  
Epoch 2/20  
1124/1124 [=====] - 62s 55ms/step - loss: 0.1455 - mse:  
0.0495 - val_loss: 0.1296 - val_mse: 0.0372  
Epoch 3/20  
1124/1124 [=====] - 61s 55ms/step - loss: 0.1225 - mse:  
0.0342 - val_loss: 0.1188 - val_mse: 0.0312  
Epoch 4/20  
1124/1124 [=====] - 61s 54ms/step - loss: 0.1133 - mse:  
0.0308 - val_loss: 0.1113 - val_mse: 0.0294  
Epoch 5/20  
1124/1124 [=====] - 63s 56ms/step - loss: 0.1078 - mse:  
0.0290 - val_loss: 0.1061 - val_mse: 0.0276  
Epoch 6/20  
1124/1124 [=====] - 61s 54ms/step - loss: 0.1038 - mse:  
0.0277 - val_loss: 0.1045 - val_mse: 0.0268  
Epoch 7/20  
1124/1124 [=====] - 62s 55ms/step - loss: 0.1009 - mse:  
0.0269 - val_loss: 0.1037 - val_mse: 0.0269  
Epoch 8/20  
1124/1124 [=====] - 63s 56ms/step - loss: 0.0984 - mse:  
0.0262 - val_loss: 0.0988 - val_mse: 0.0258  
Epoch 9/20  
1124/1124 [=====] - 63s 56ms/step - loss: 0.0961 - mse:  
0.0255 - val_loss: 0.0978 - val_mse: 0.0253  
Epoch 10/20  
1124/1124 [=====] - 63s 56ms/step - loss: 0.0940 - mse:  
0.0249 - val_loss: 0.0960 - val_mse: 0.0248  
Epoch 11/20  
1124/1124 [=====] - 62s 56ms/step - loss: 0.0923 - mse:  
0.0244 - val_loss: 0.0949 - val_mse: 0.0245  
Epoch 12/20
```

```
1124/1124 [=====] - 61s 54ms/step - loss: 0.0906 - mse:  
0.0238 - val_loss: 0.0942 - val_mse: 0.0244  
Epoch 13/20  
1124/1124 [=====] - 61s 54ms/step - loss: 0.0891 - mse:  
0.0233 - val_loss: 0.0919 - val_mse: 0.0236  
Epoch 14/20  
1124/1124 [=====] - 60s 53ms/step - loss: 0.0878 - mse:  
0.0227 - val_loss: 0.0922 - val_mse: 0.0237  
Epoch 15/20  
1124/1124 [=====] - 60s 53ms/step - loss: 0.0864 - mse:  
0.0223 - val_loss: 0.0901 - val_mse: 0.0230  
Epoch 16/20  
1124/1124 [=====] - 60s 54ms/step - loss: 0.0851 - mse:  
0.0219 - val_loss: 0.0892 - val_mse: 0.0229  
Epoch 17/20  
1124/1124 [=====] - 61s 55ms/step - loss: 0.0839 - mse:  
0.0216 - val_loss: 0.0897 - val_mse: 0.0231  
Epoch 18/20  
1124/1124 [=====] - 60s 53ms/step - loss: 0.0830 - mse:  
0.0213 - val_loss: 0.0877 - val_mse: 0.0227  
Epoch 19/20  
1124/1124 [=====] - 61s 55ms/step - loss: 0.0822 - mse:  
0.0210 - val_loss: 0.0878 - val_mse: 0.0228  
Epoch 20/20  
1124/1124 [=====] - 59s 53ms/step - loss: 0.0813 - mse:  
0.0208 - val_loss: 0.0861 - val_mse: 0.0225
```

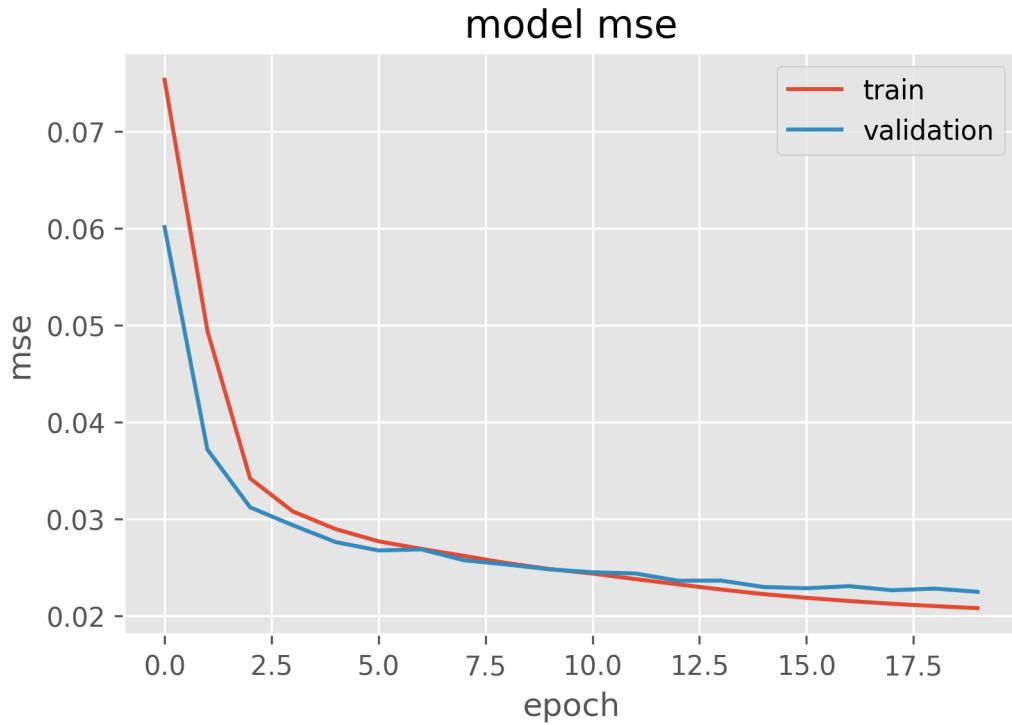
1.4.3 Model History

Summarize and plot the history of the loss and val_loss

```
[19]: plt.plot(history.history['loss'])  
plt.plot(history.history['val_loss'])  
plt.style.use("ggplot")  
plt.title('model loss (MAE)')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train', 'validation'], loc='upper right')  
plt.show()
```



```
[18]: plt.plot(history.history['mse'])
plt.plot(history.history['val_mse'])
plt.style.use("ggplot")
plt.title('model mse')
plt.ylabel('mse')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



Evaluate Model // Show the MAE and MSE metrics

```
[20]: model.evaluate(X_test, y_test)
```

```
139/139 [=====] - 4s 25ms/step - loss: 0.0896 - mse: 0.0237
```

```
[20]: [0.08956620842218399, 0.023685818538069725]
```

1.4.4 Predicting GRU

Predict GRU Model

```
[21]: yhat = model.predict(X_test)
```

```
139/139 [=====] - 5s 26ms/step
```

Saving the shape of y_test

```
[22]: test_shape = y_test.shape
```

Print RMSE Plots from t+1 to t+20

```
[23]: from math import sqrt
# evaluate forecasts against expected values
```

```

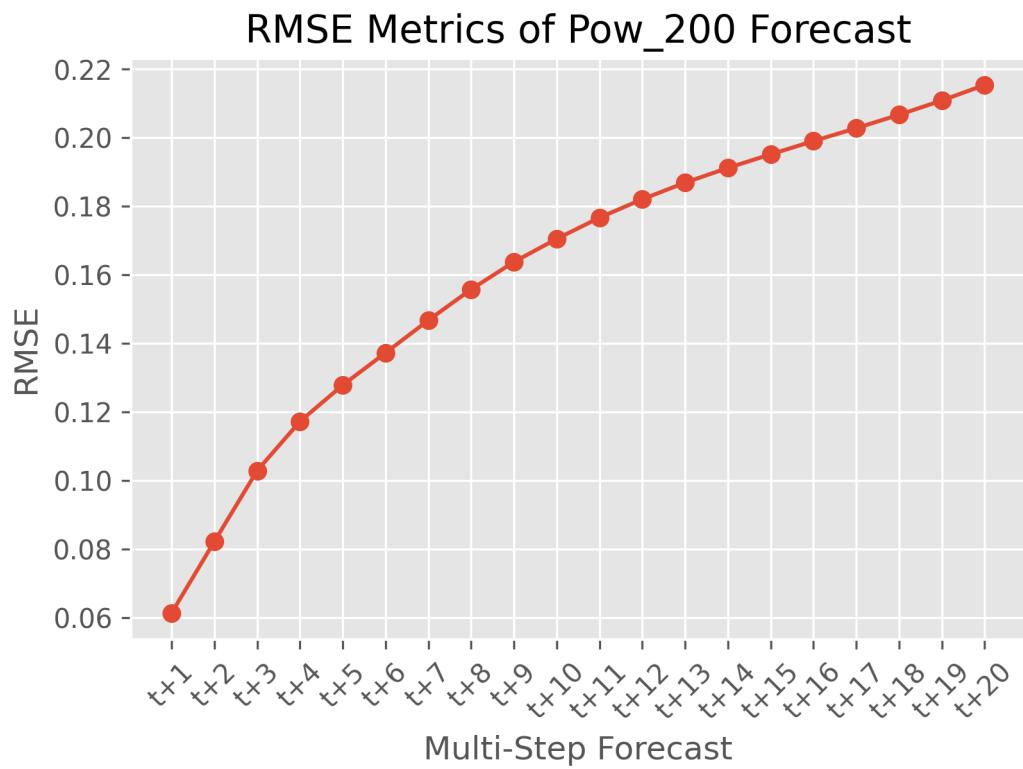
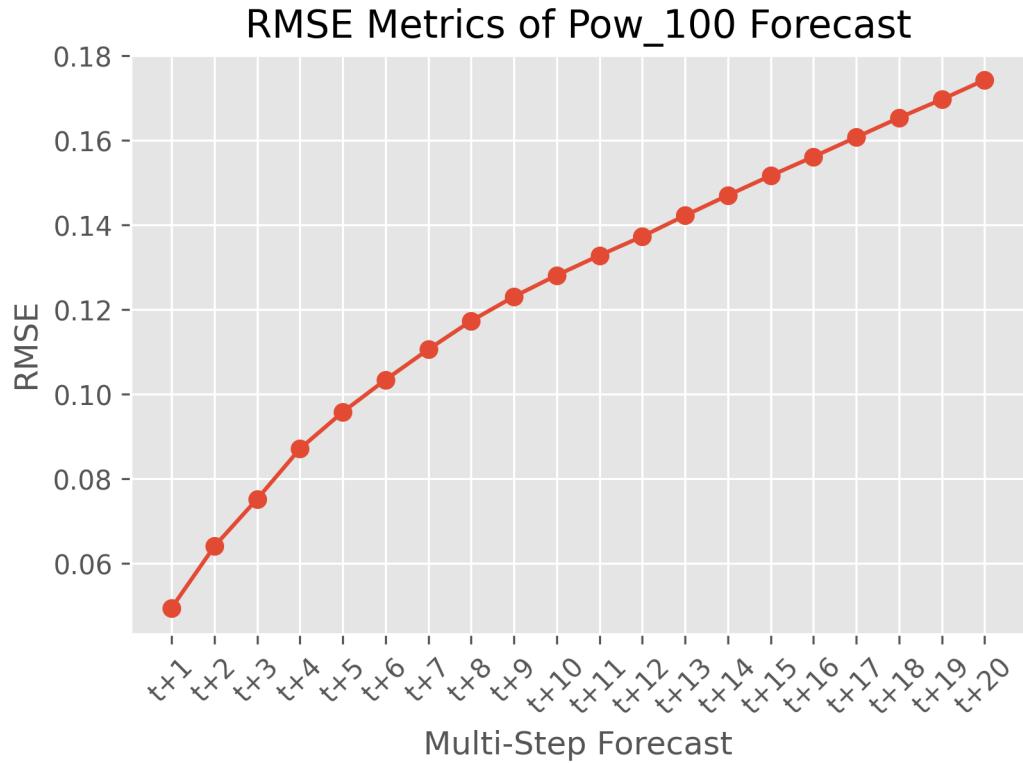
def evaluate_forecasts(actual, predicted):
    scores_pow = results = np.zeros((20, 20))
    scores = []
    # calculate an RMSE score for each prediction period
    for i in range(actual.shape[1]):
        for t in range(actual.shape[2]):
            # calculate mse
            scores_pow[i, t] = sqrt(mean_squared_error(actual[:, t, i], predicted[:, t, i]))
    # calculate overall RMSE
    s = 0
    for row in range(actual.shape[0]):
        for col in range(actual.shape[1]):
            s += (actual[row, col, 1] - predicted[row, col, 1])**2
        score = sqrt(s / (actual.shape[0] * actual.shape[1]))
    return np.array(score), np.array(scores_pow)

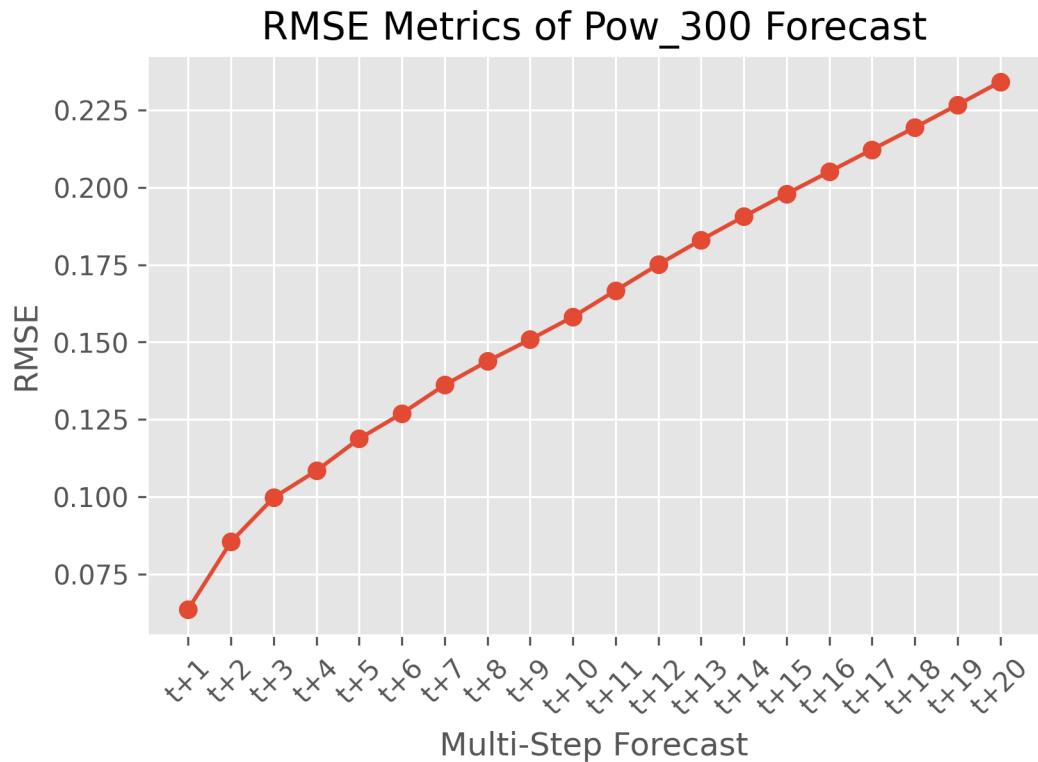
```

[24]: def summarize_scores(name, score, scores):
 s_scores = ', '.join(['%.3f' % s for s in scores])
 print('%s: [%.3f] %s' % (name, score, s_scores))

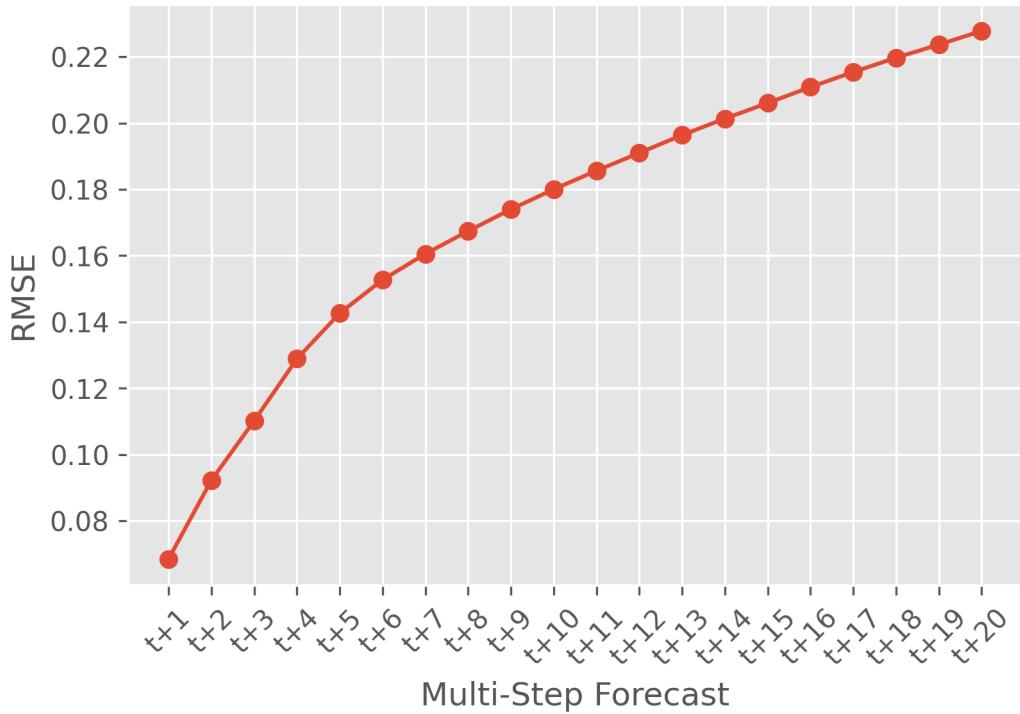
[25]: overall_RMSE, all_RMSEs = evaluate_forecasts(y_test, yhat)

[26]: forecast = []
 for i in range(1,21):
 title = f't+{i}'
 forecast.append(title)
 for i in range(20):
 plt.style.use("ggplot")
 plt.plot(forecast, all_RMSEs[i], marker='o', label='gru')
 plt.xticks(rotation=45)
 plt.title('RMSE Metrics of Pow_ ' +str(100*(i+1))+ ' Forecast')
 plt.xlabel('Multi-Step Forecast')
 plt.ylabel('RMSE')
 plt.show()

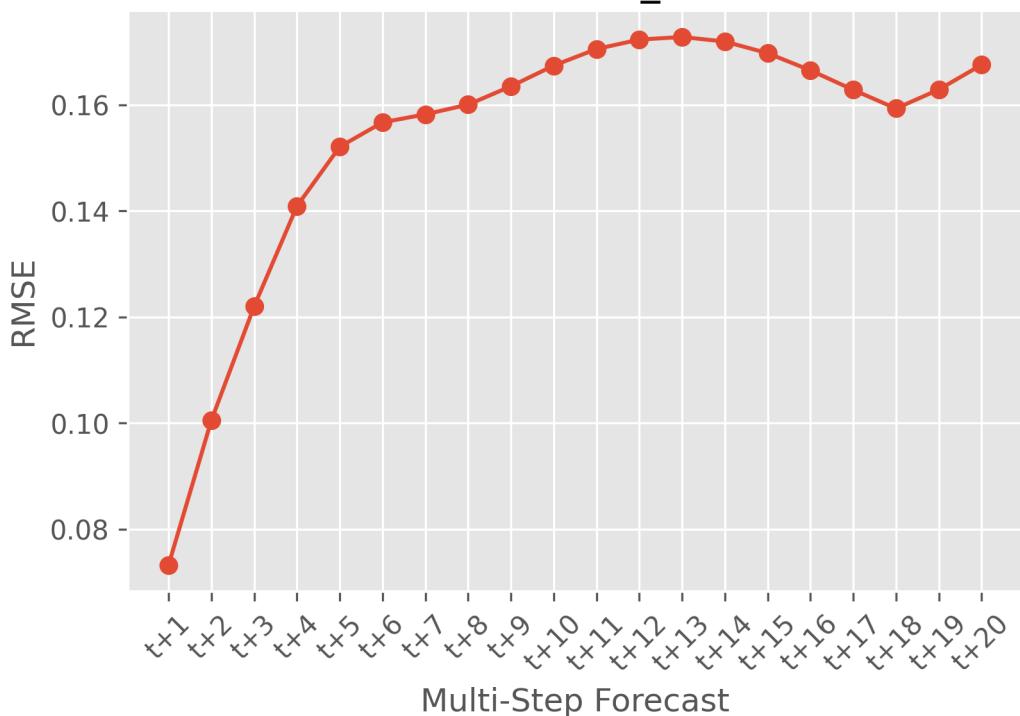




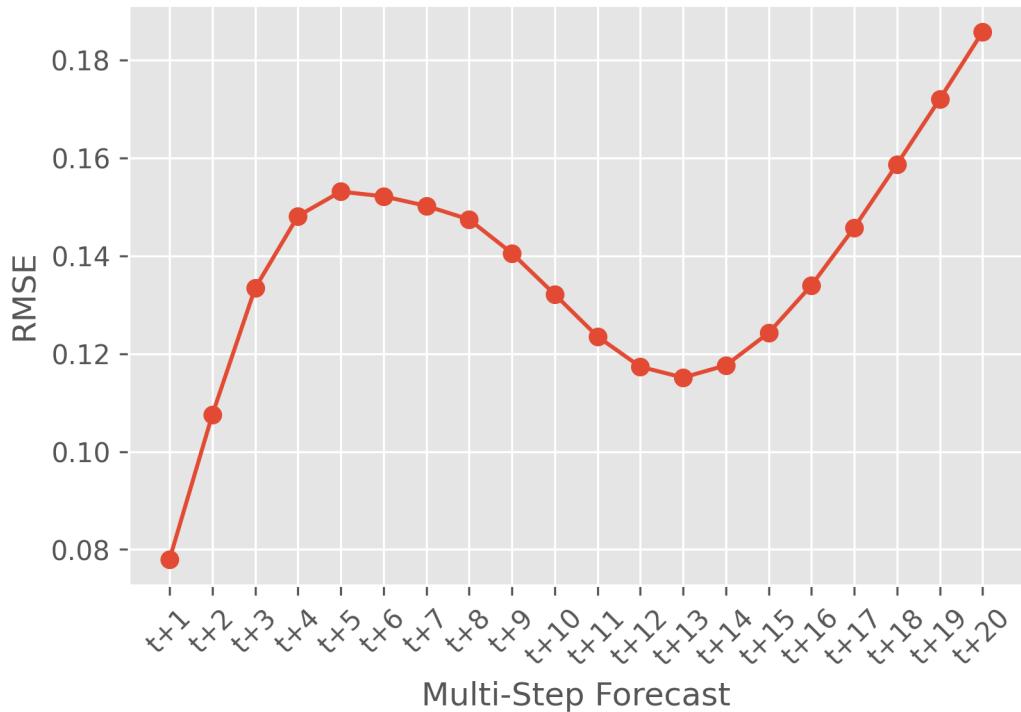
RMSE Metrics of Pow_400 Forecast



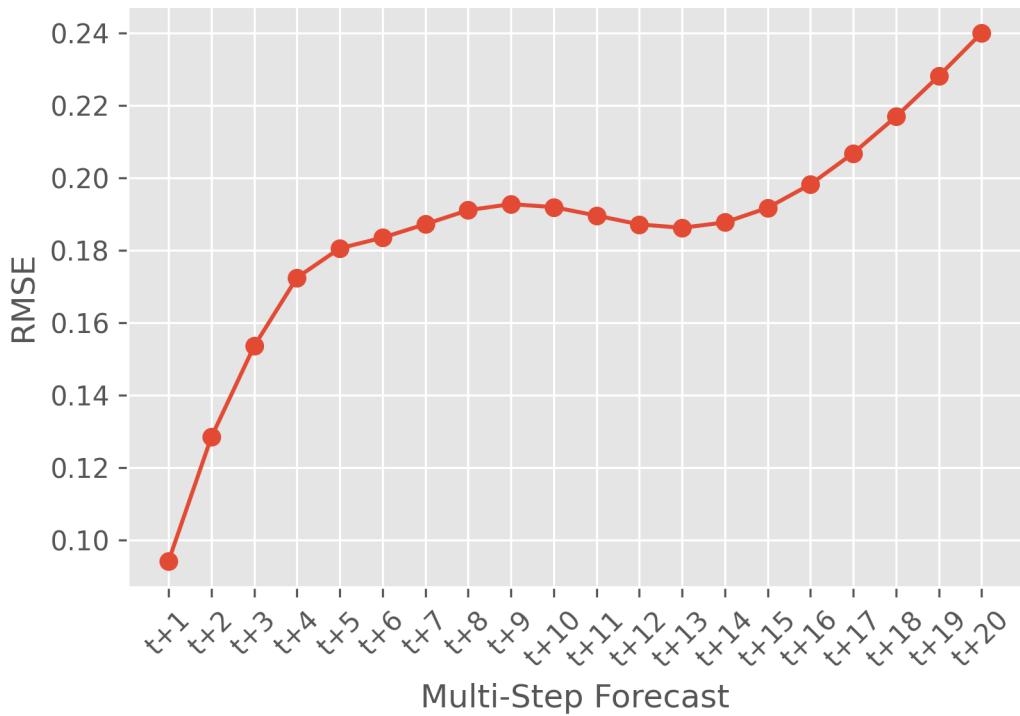
RMSE Metrics of Pow_500 Forecast



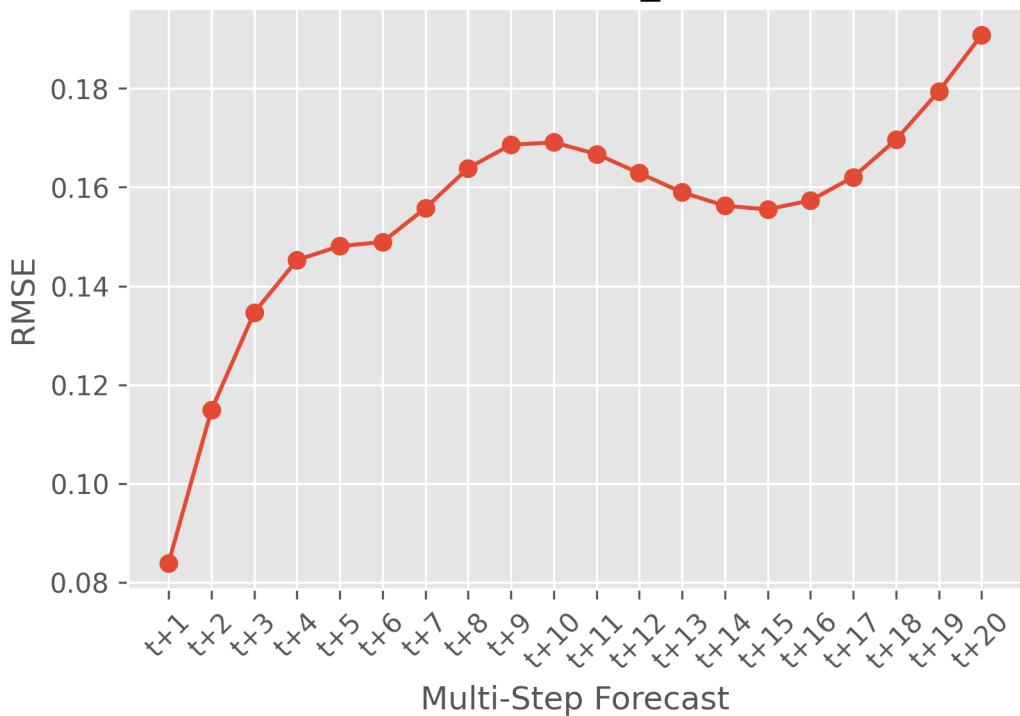
RMSE Metrics of Pow_600 Forecast



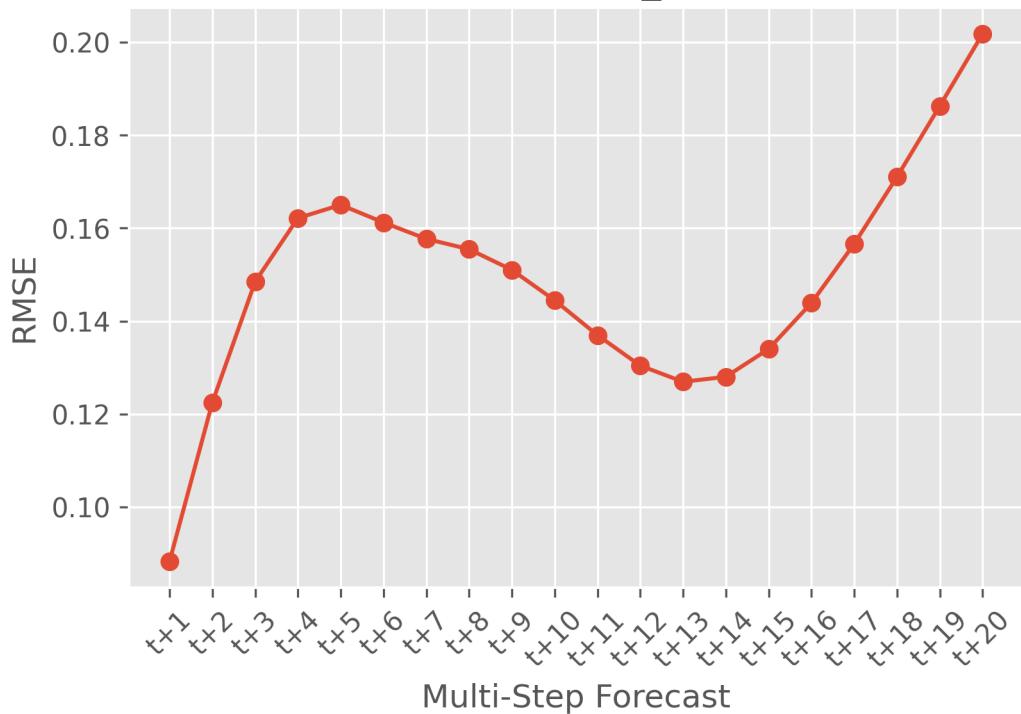
RMSE Metrics of Pow_700 Forecast



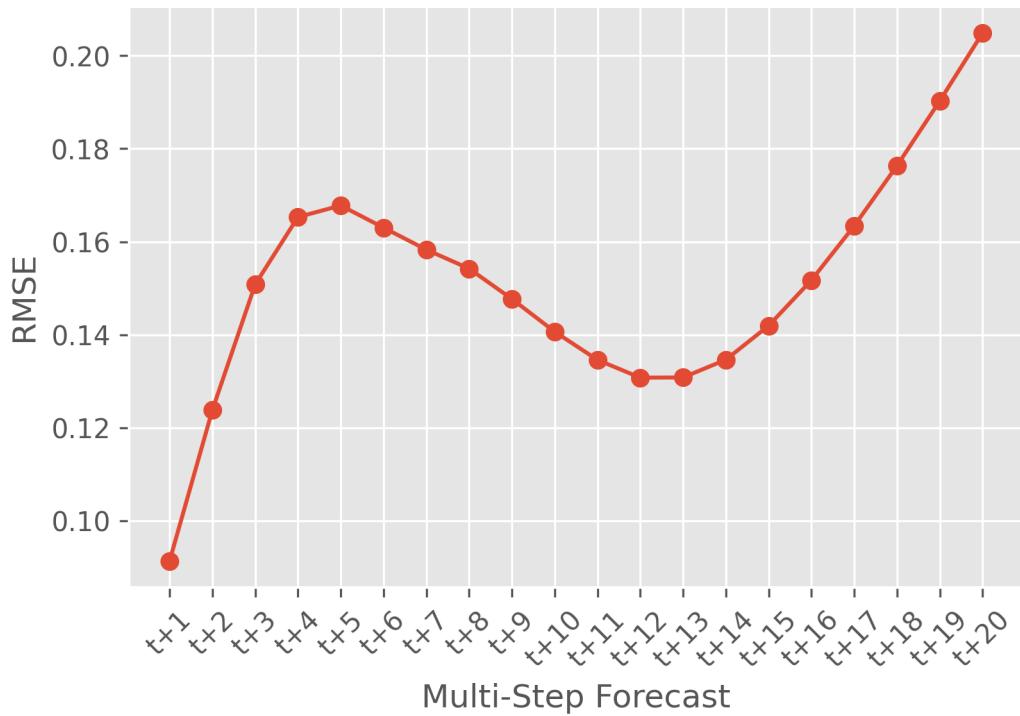
RMSE Metrics of Pow_800 Forecast



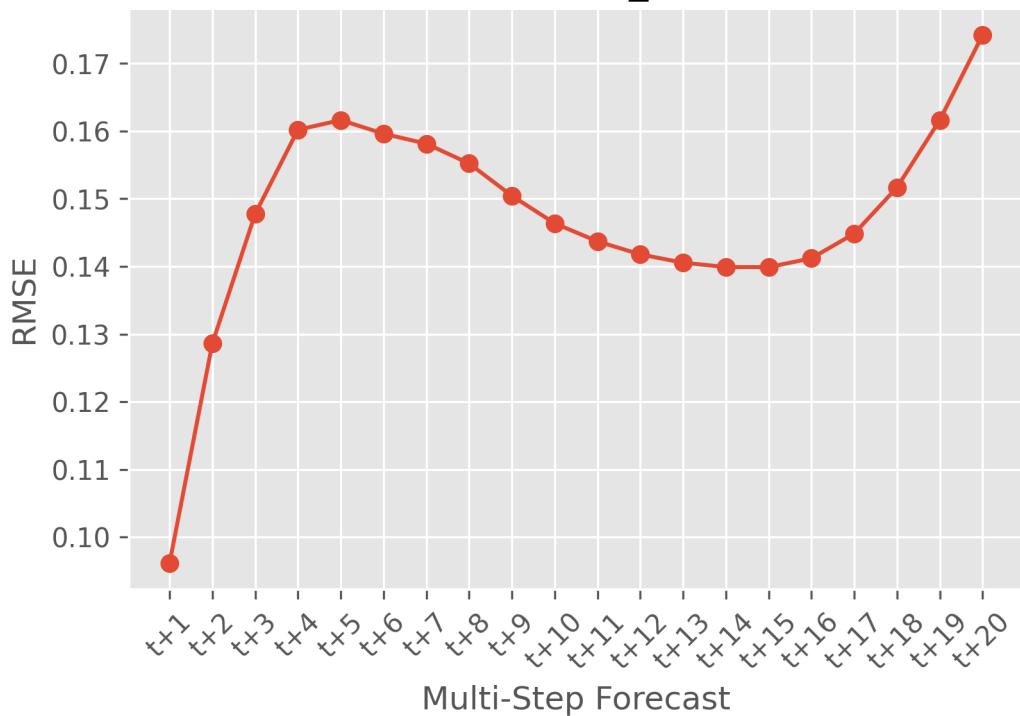
RMSE Metrics of Pow_900 Forecast

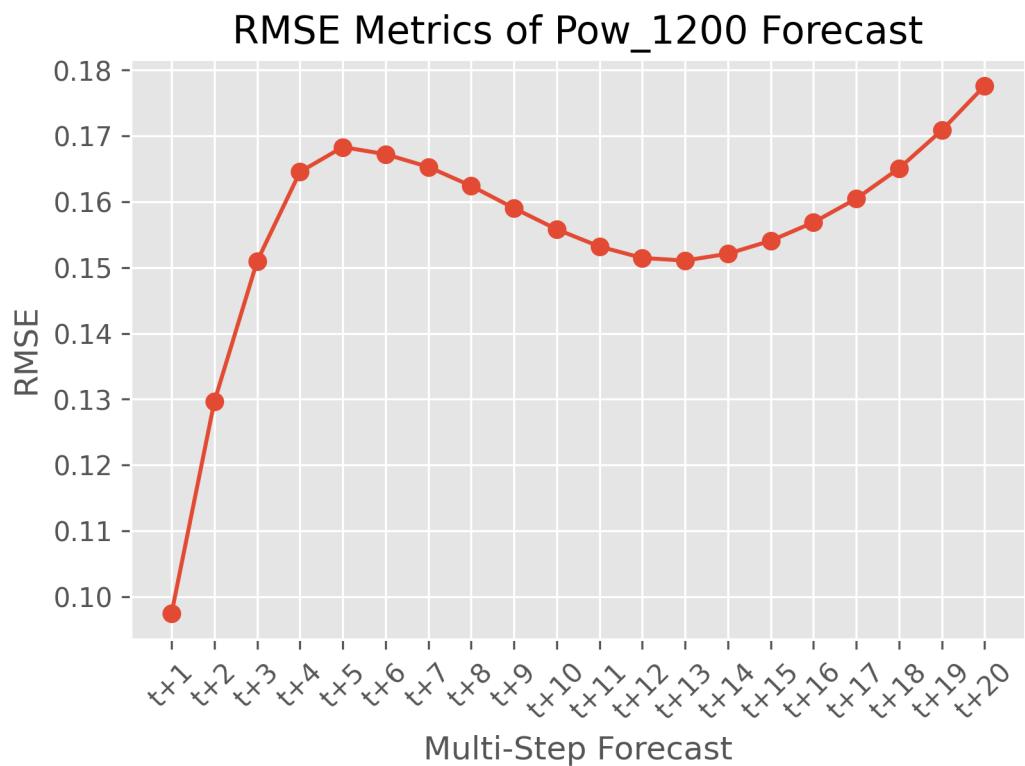


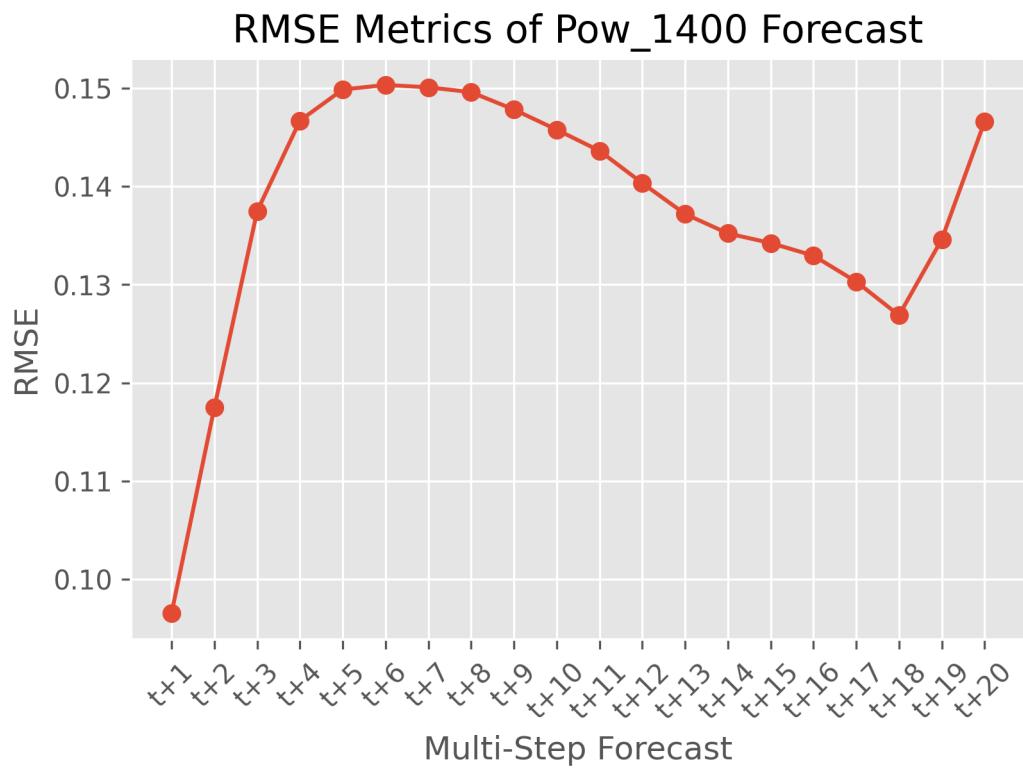
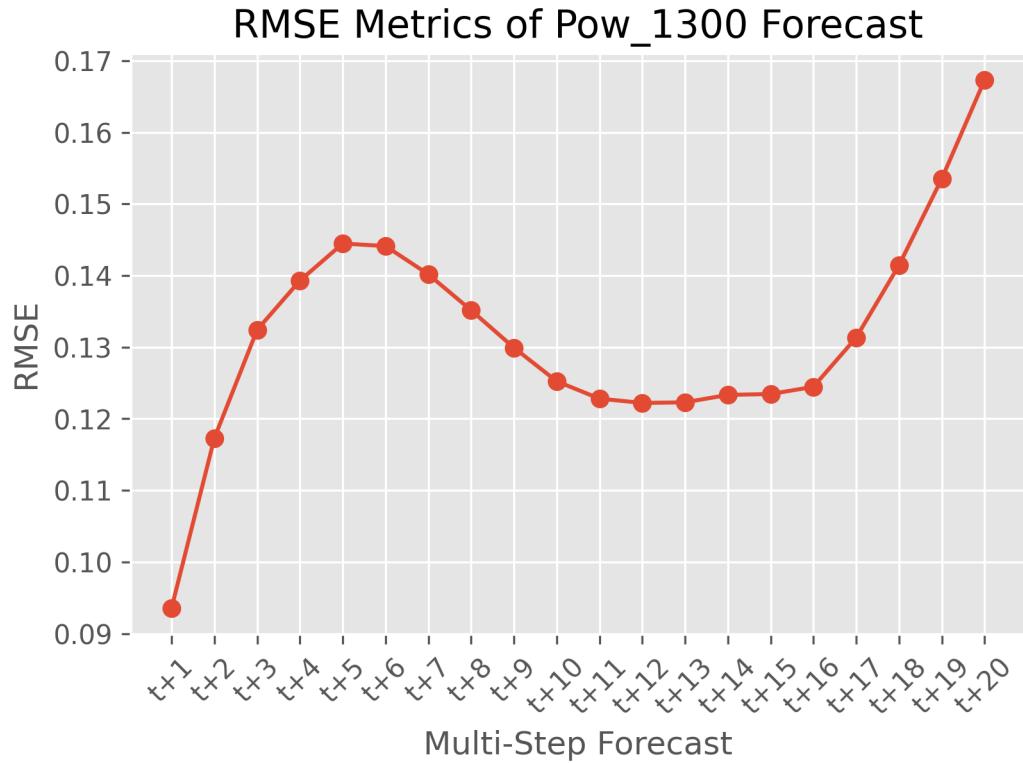
RMSE Metrics of Pow_1000 Forecast

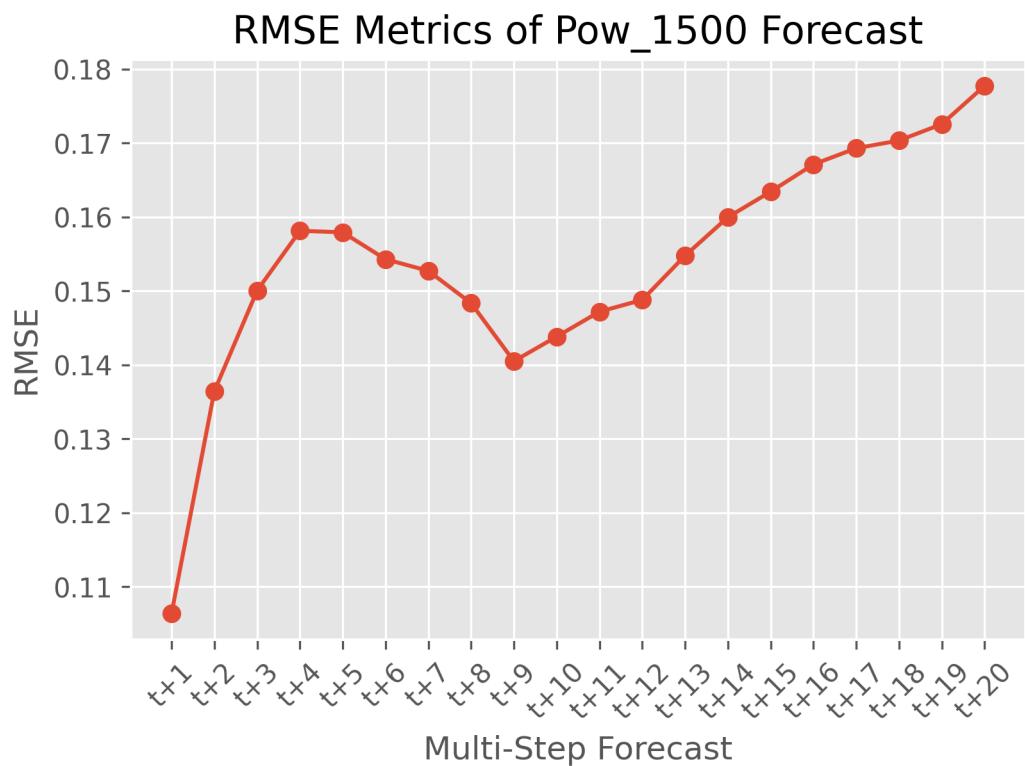


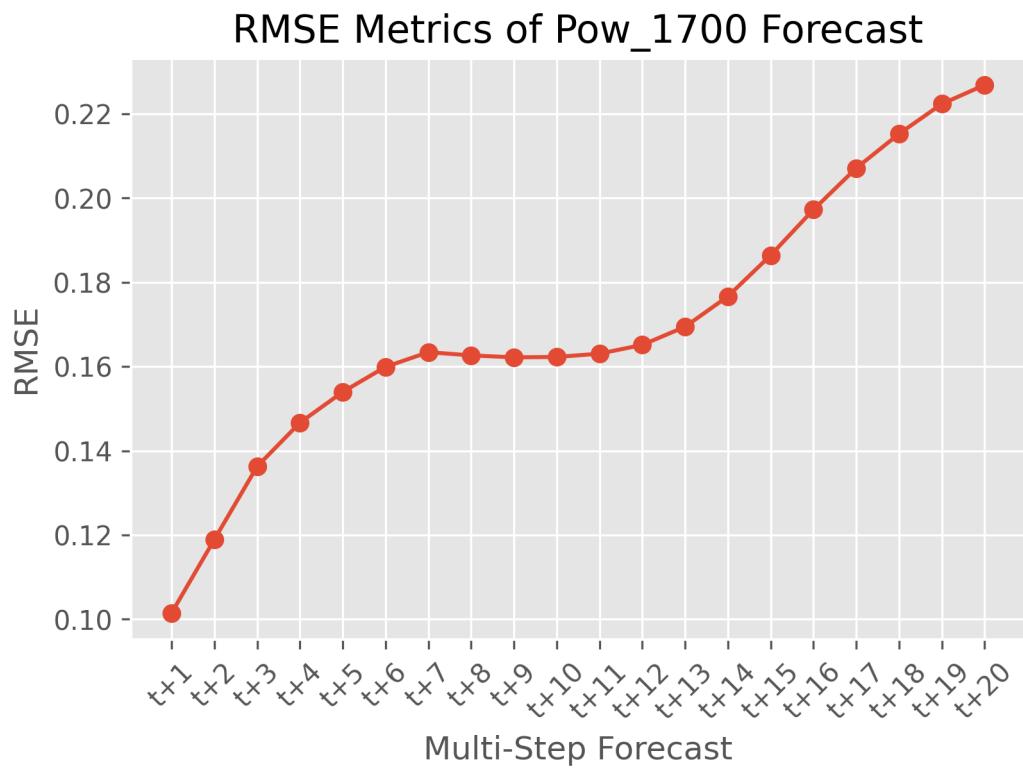
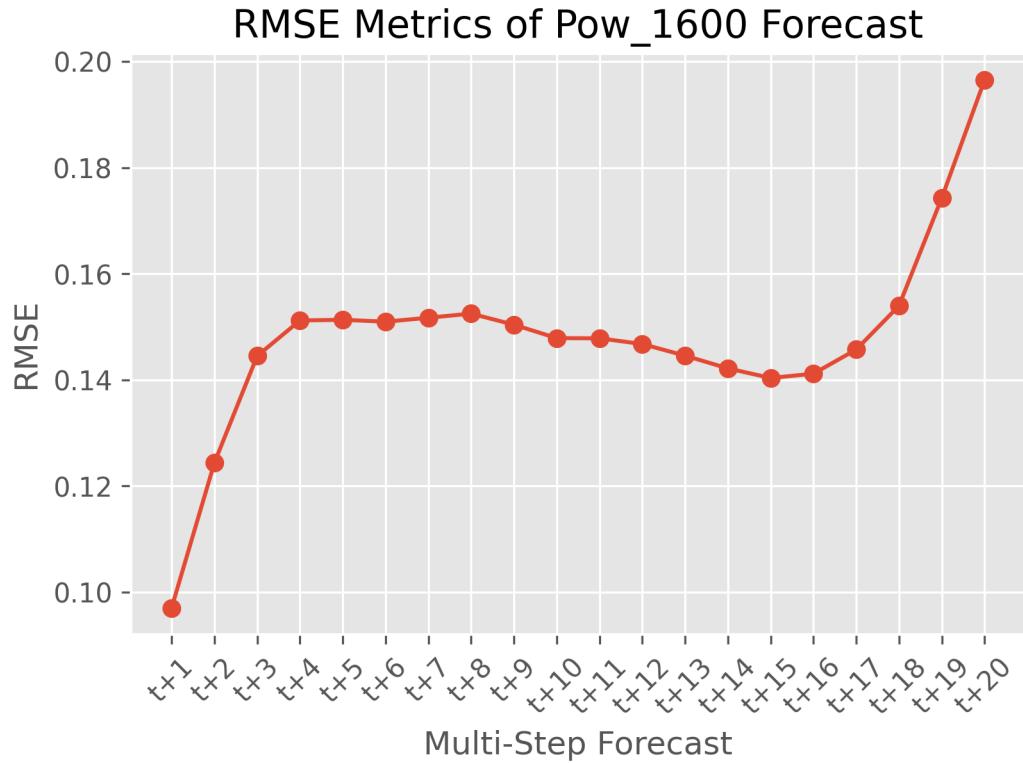
RMSE Metrics of Pow_1100 Forecast



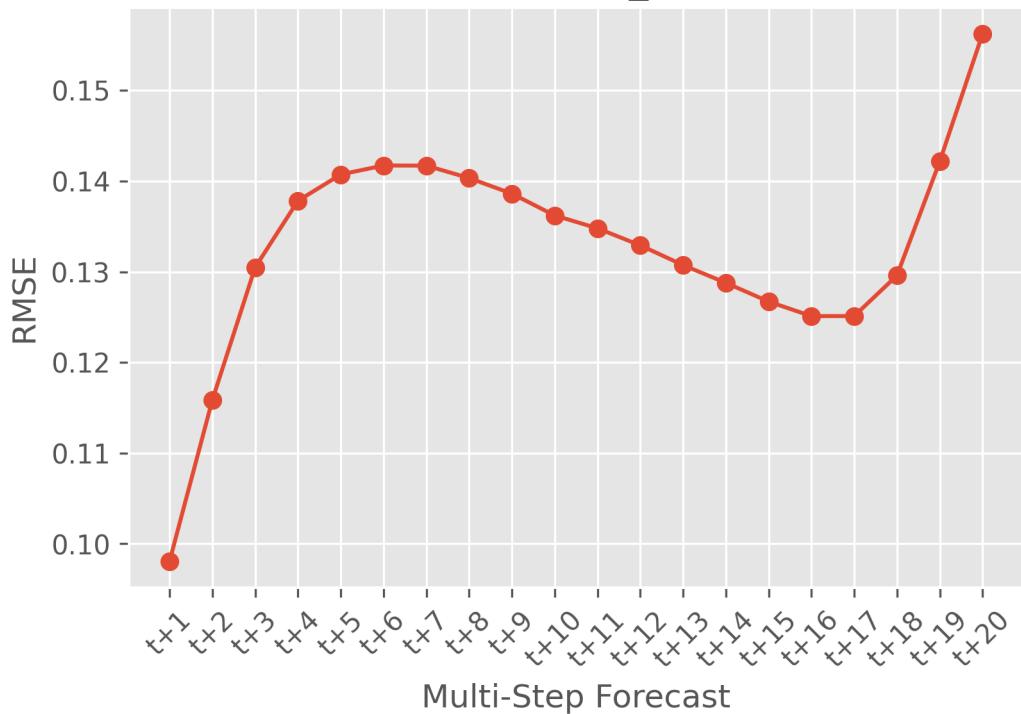




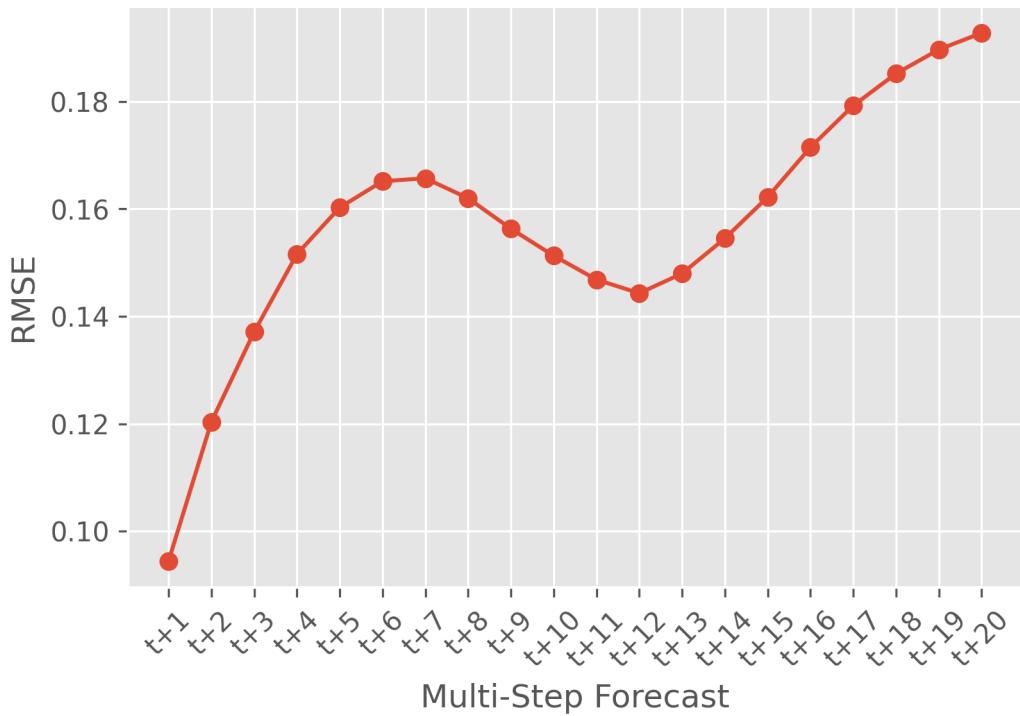




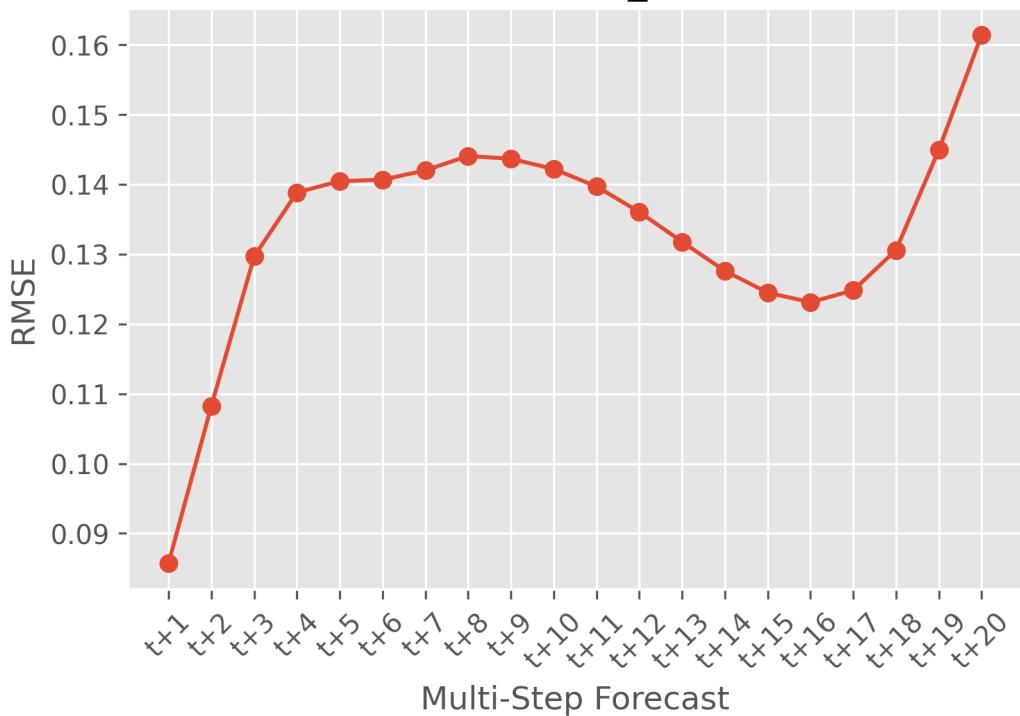
RMSE Metrics of Pow_1800 Forecast



RMSE Metrics of Pow_1900 Forecast



RMSE Metrics of Pow_2000 Forecast



Backing up the data

```
[27]: yhat2 = yhat  
y_test2 = y_test
```

New Unseen Predictions

```
[28]: last_predictions = yhat[-2:,:,:]  
# Generate new predictions  
forecast_predictions = model.predict(last_predictions)  
forecast_predictions2 = model.predict(forecast_predictions)  
forecast_predictions3 = model.predict(forecast_predictions2)
```

```
1/1 [=====] - 1s 569ms/step  
1/1 [=====] - 0s 22ms/step  
1/1 [=====] - 0s 20ms/step
```

Rescale

```
[29]: def inverse_scale_data(yhat, y_test):  
    yhat_reshaped = yhat.reshape(-1,yhat.shape[-1])  
    y_test_reshaped = y_test.reshape(-1,y_test.shape[-1])  
  
    yhat_inverse = scaler.inverse_transform(yhat_reshaped)  
    y_test_inverse = scaler.inverse_transform(y_test_reshaped)  
    return yhat_inverse, y_test_inverse
```

```
[30]: re_yhat, re_y_test = inverse_scale_data(yhat2, y_test2)
```

1.4.5 Calculating the additional Predictions of 600ms (AddOn – Recursive Method)

```
[31]: y_test_3 = y_test  
y_test_4 = y_test  
y_test_5 = y_test
```

```
[32]: re_forecast_predictions, re_y_test3 = inverse_scale_data(forecast_predictions,  
    ↪y_test_3)  
re_forecast_predictions2, re_y_test4 =  
    ↪inverse_scale_data(forecast_predictions2, y_test_4)  
re_forecast_predictions3, re_y_test5 =  
    ↪inverse_scale_data(forecast_predictions3, y_test_5)
```

```
[33]: # reshape from 2D back to 3D (2D was necessary for the inverse scaling)  
re_yhat2 = re_yhat.reshape(test_shape)  
re_y_test2 = re_y_test.reshape(test_shape)
```

```
[34]: re_forecast_predictions = re_forecast_predictions.reshape(forecast_predictions.
    ↪shape)
re_forecast_predictions2 = re_forecast_predictions2.
    ↪reshape(forecast_predictions2.shape)
re_forecast_predictions3 = re_forecast_predictions3.
    ↪reshape(forecast_predictions2.shape)
```

2 Model Evaluation

2.1 Defining Functions

Build the dataframe for the additional 600ms predictions (Recursive Method)

```
[35]: f_p0 = pd.DataFrame(re_yhat2[-1, -1, :])
f_p1 = pd.DataFrame(re_forecast_predictions[-1])
f_p2 = pd.DataFrame(re_forecast_predictions2[-1])
f_p3 = pd.DataFrame(re_forecast_predictions3[-1])
f_p = pd.concat([f_p0.transpose(), f_p1, f_p2, f_p3])

time_index2 = np.arange(4457, 4518, dtype='float32')
f_p['time_index2'] = time_index2
f_p.set_index('time_index2', inplace=True)
```

Function for printing out the 20 RMP's Predictions vs. its Acutals

```
[36]: from sklearn.metrics import r2_score
def print_act_and_pred_tables(yhat,ytest,forecast_horizon,
    ↪start_graph,end_graph):
    pow_preds = []
    pow_actuals = []
    for i in range(20):
        pow_preds.append(yhat[:, forecast_horizon-1, i]) # 1 refers to the forecast
    ↪horizon --> t+1; shape of yhat: [(length), (n_outputs), (n_features)]
        pow_actuals.append(ytest[:, 0, i])

    data = {}
    for i in range(20):
        pow_pred_label = f"Pow{100*(i+1)} Predictions"
        pow_act_label = f"Pow{100*(i+1)} Actuals"
        data[pow_pred_label] = pow_preds[i]
        data[pow_act_label] = pow_actuals[i]
    df_new = pd.DataFrame(data=data)
    #Plot
    for i in range(20):
        fig = plt.figure(figsize=(15, 7))
        plt.style.use("ggplot")
        # Select the actuals and predictions columns
```

```

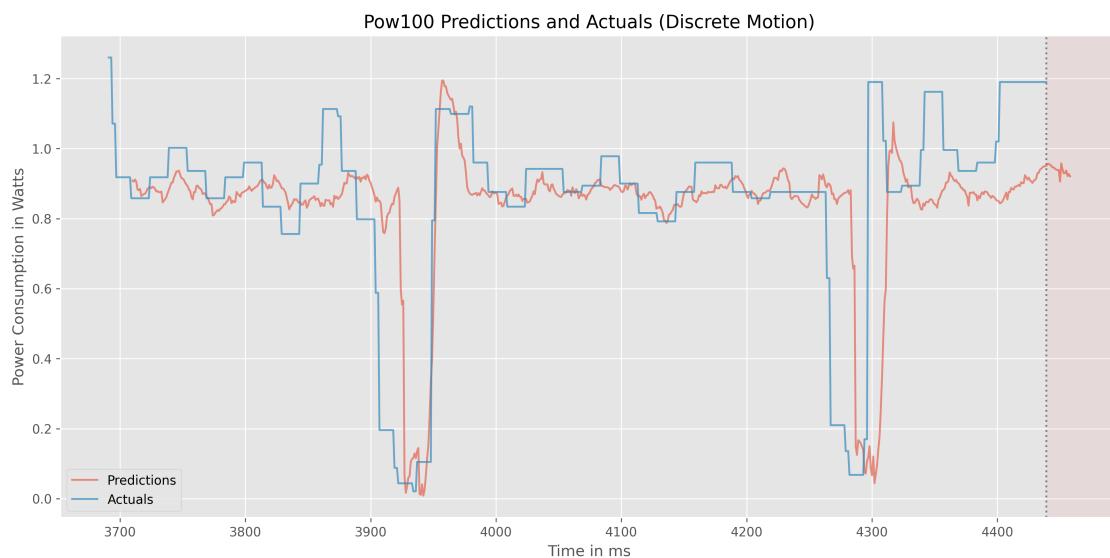
actuals = df_new[f"Pow{100*(i+1)} Actuals"][start_graph:end_graph]
predictions = df_new[f"Pow{100*(i+1)} Predictions"][start_graph:end_graph]
# shift the t+20 prediction 20 (relating to the wished forecast horizon) to the right
p_temp = predictions.to_frame()
p_temp['time_index_shift'] = predictions.index+forecast_horizon-1
p_temp.set_index('time_index_shift', inplace=True)
# Plot the data
plt.plot(p_temp, alpha=0.6)
plt.plot(actuals, alpha=0.7)
# Set the plot title and axis labels
plt.title(f'Pow{100*(i+1)} Predictions and Actuals (Discrete Motion)')
plt.xlabel('Time in ms')
plt.ylabel('Power Consumption in Watts')
# Draw horizontal Lines for better comparison
plt.axvline(x=4439, linestyle=":", color="grey")
plt.axvline(x=4477, linestyle="--", linewidth = 75, color="red", alpha=0.05)
# Set the legend
plt.legend(['Predictions', 'Actuals'])
# Show the plot
plt.show()

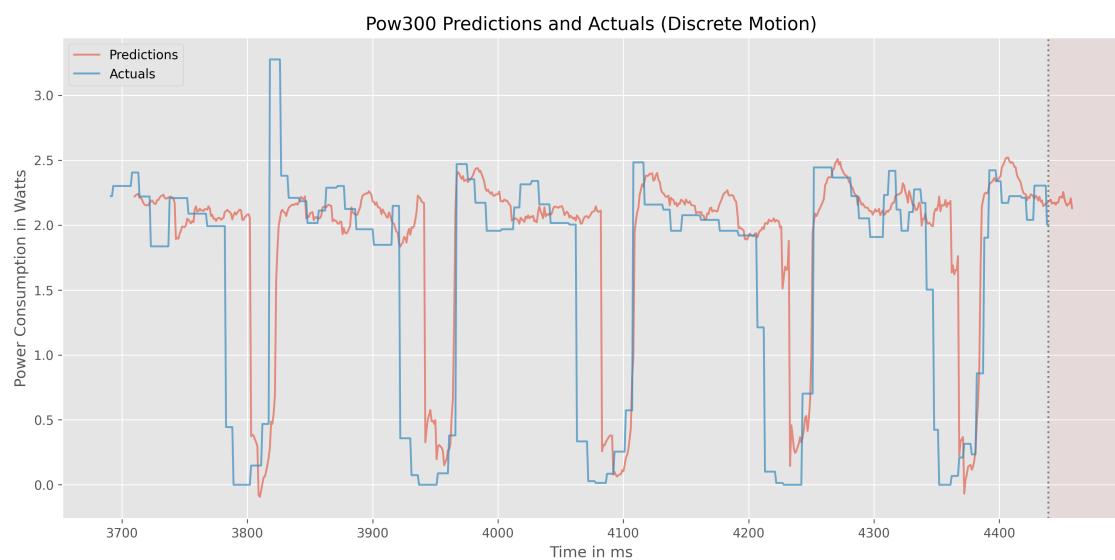
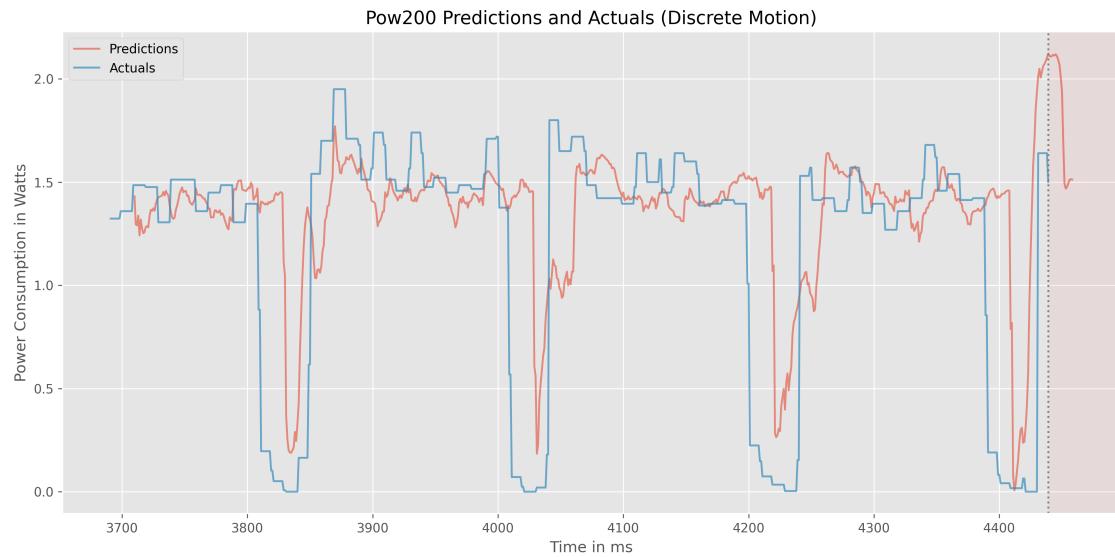
```

2.2 Print Actuals and Predictions

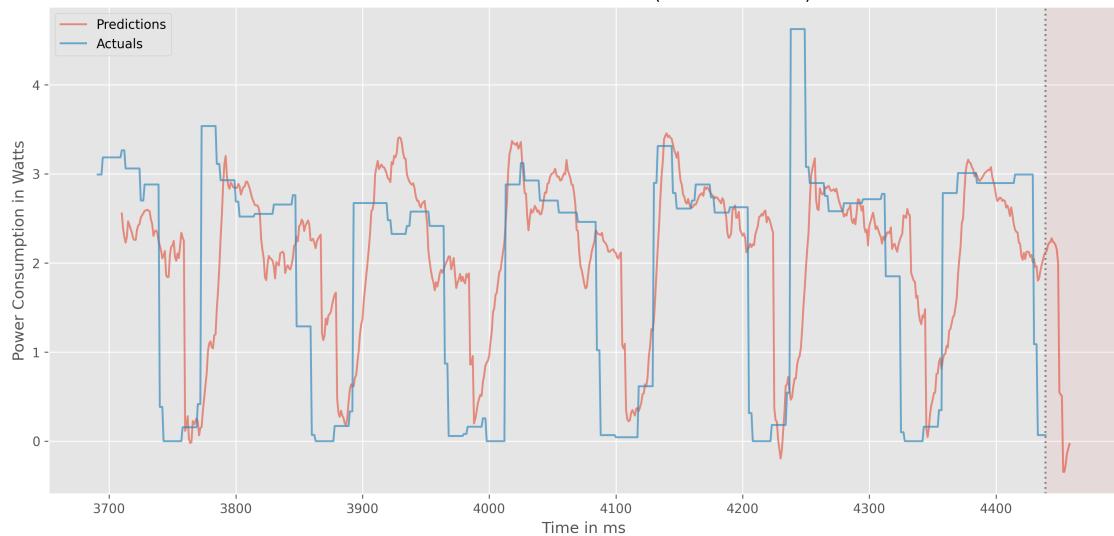
2.2.1 With Forecast Horizon of t+20 [Range: last 750 Timestamps]

[37]: print_act_and_pred_tables(re_yhat2,re_y_test2,20, -750,-1)

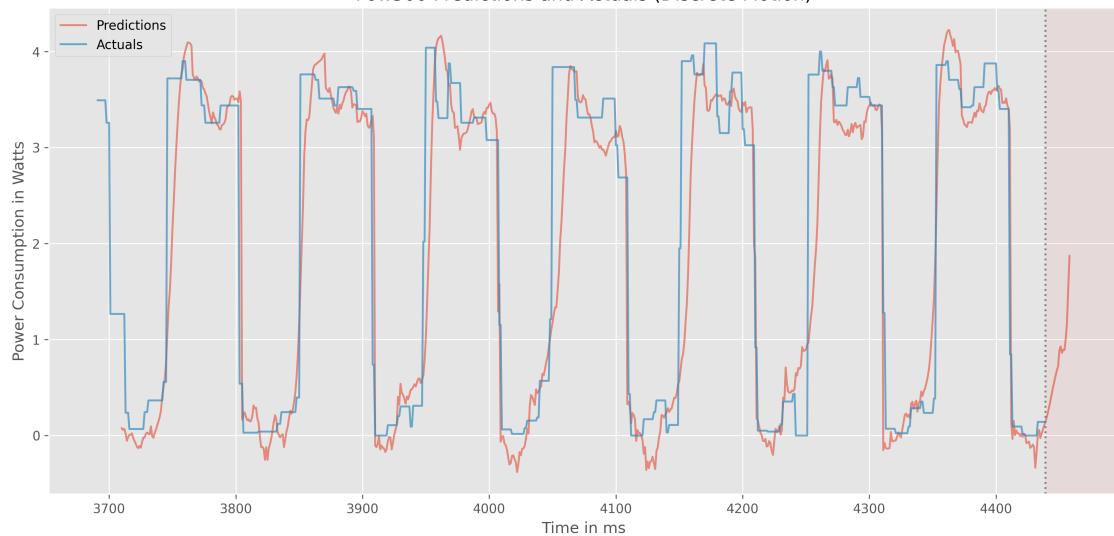




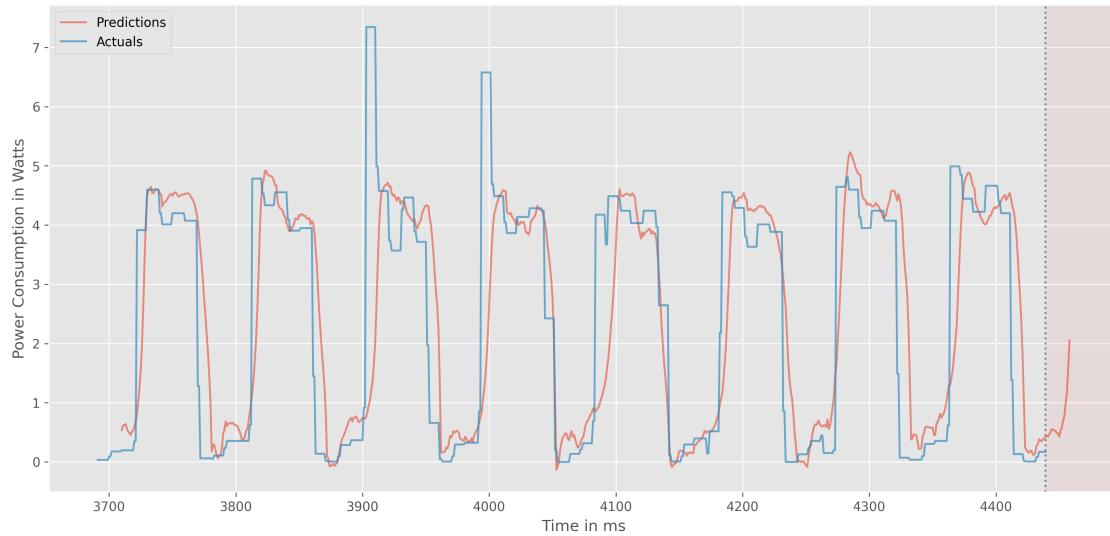
Pow400 Predictions and Actuals (Discrete Motion)



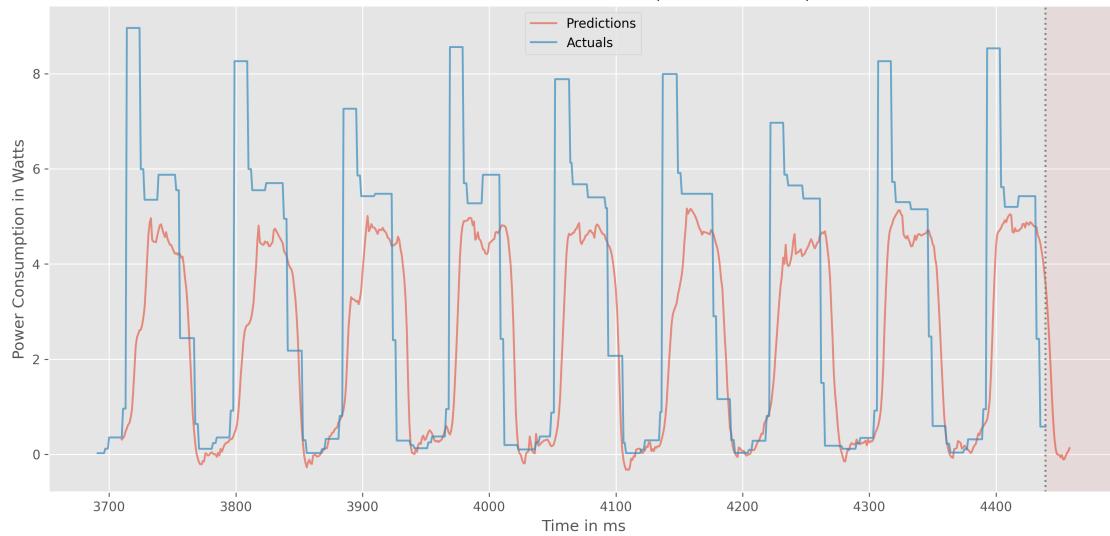
Pow500 Predictions and Actuals (Discrete Motion)



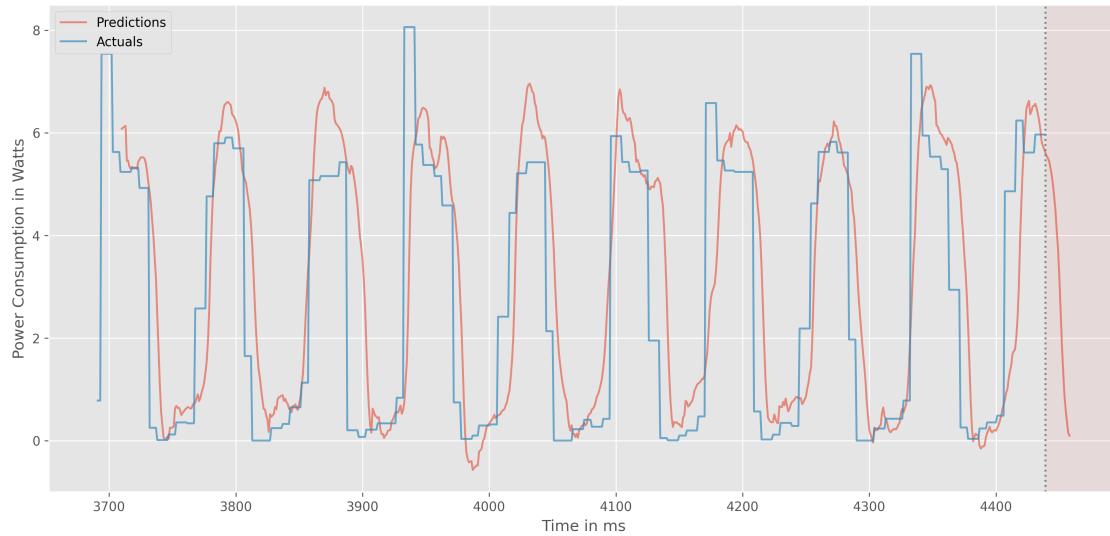
Pow600 Predictions and Actuals (Discrete Motion)



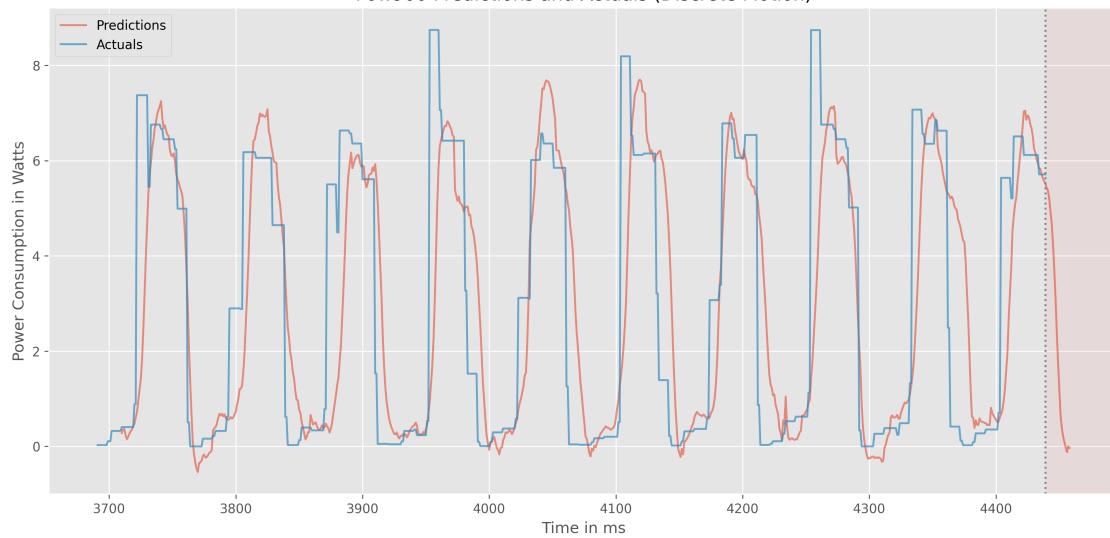
Pow700 Predictions and Actuals (Discrete Motion)



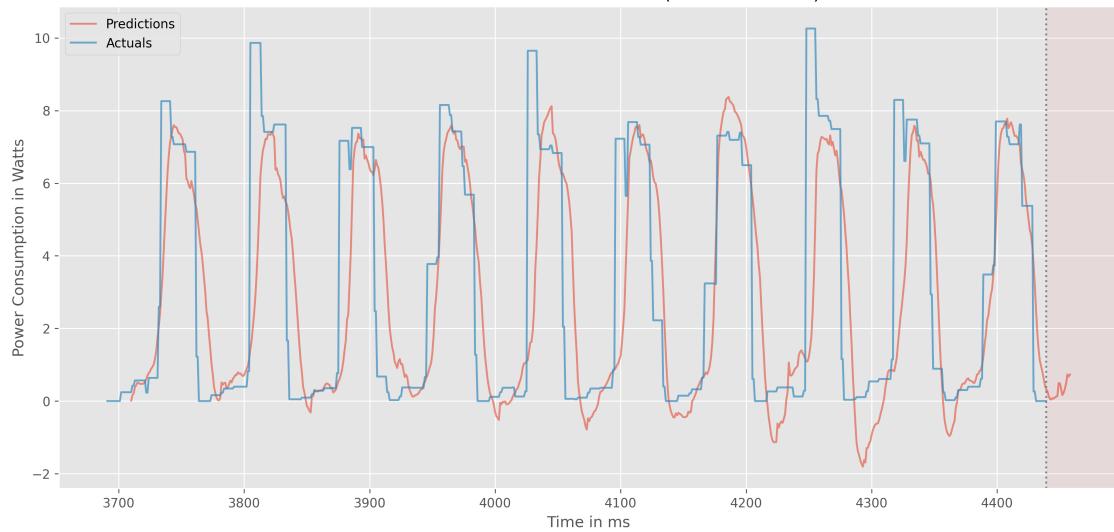
Pow800 Predictions and Actuals (Discrete Motion)



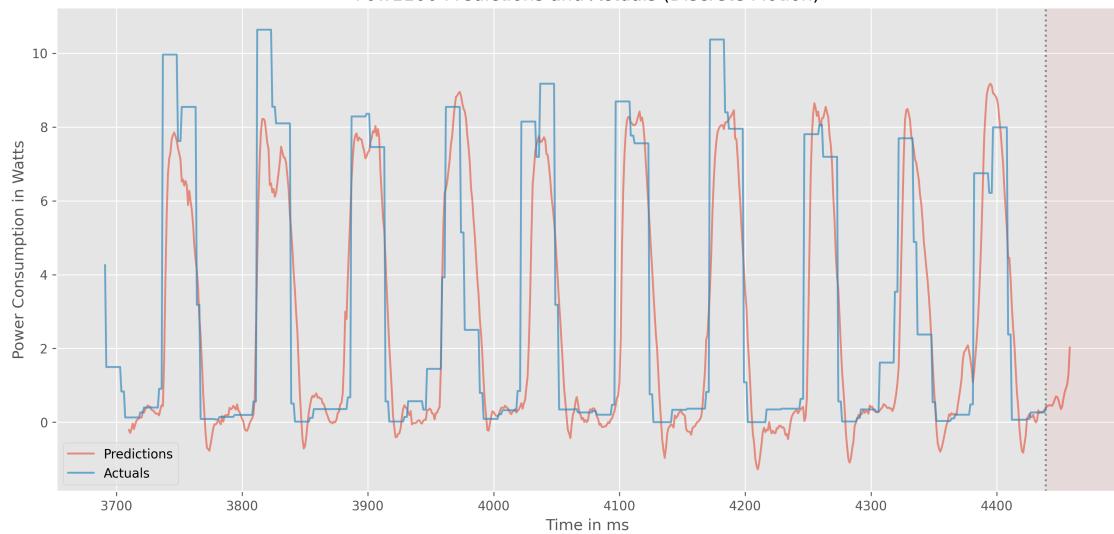
Pow900 Predictions and Actuals (Discrete Motion)

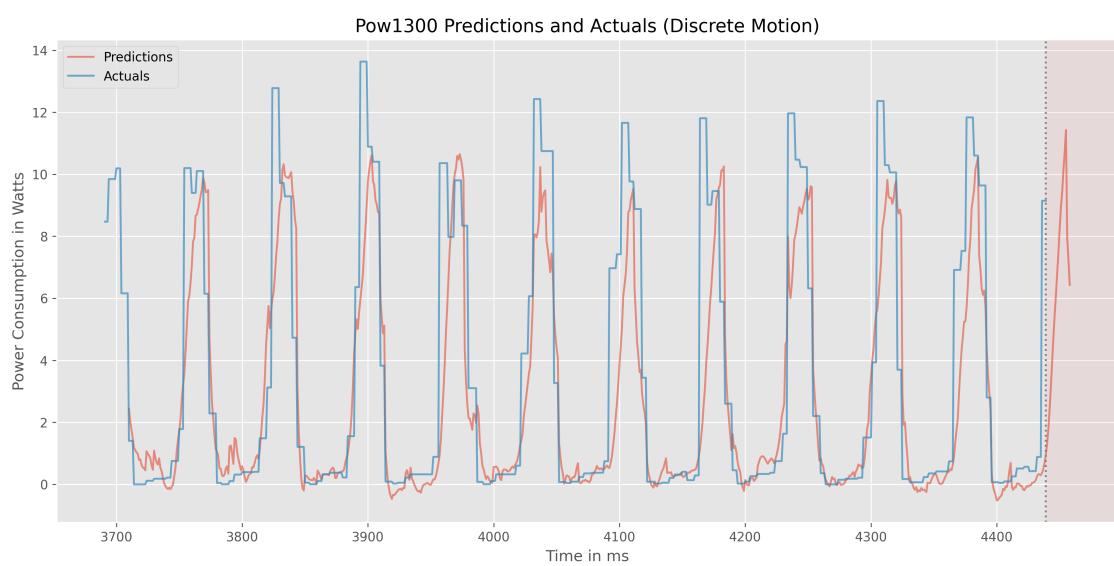
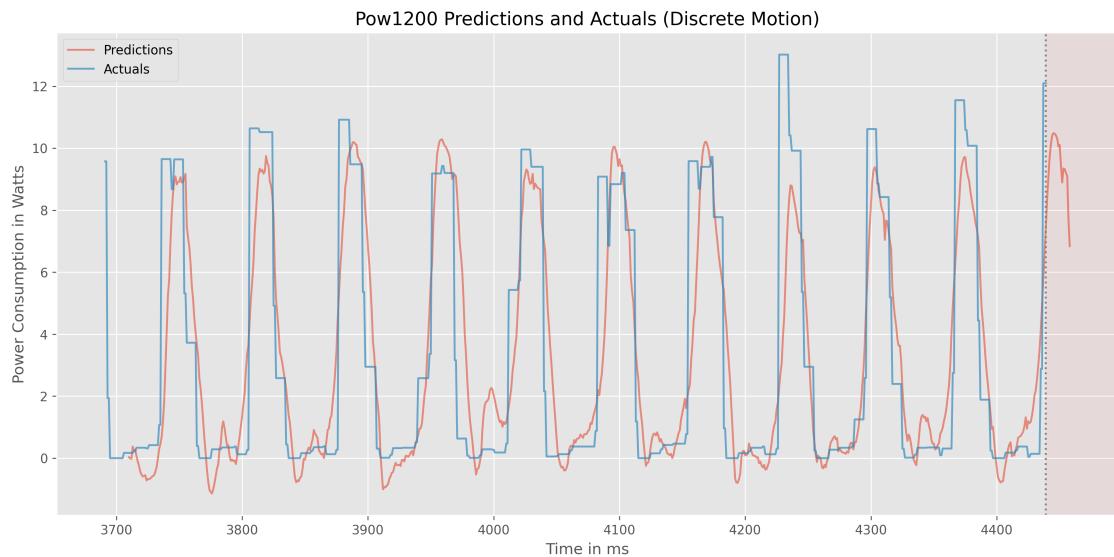


Pow1000 Predictions and Actuals (Discrete Motion)

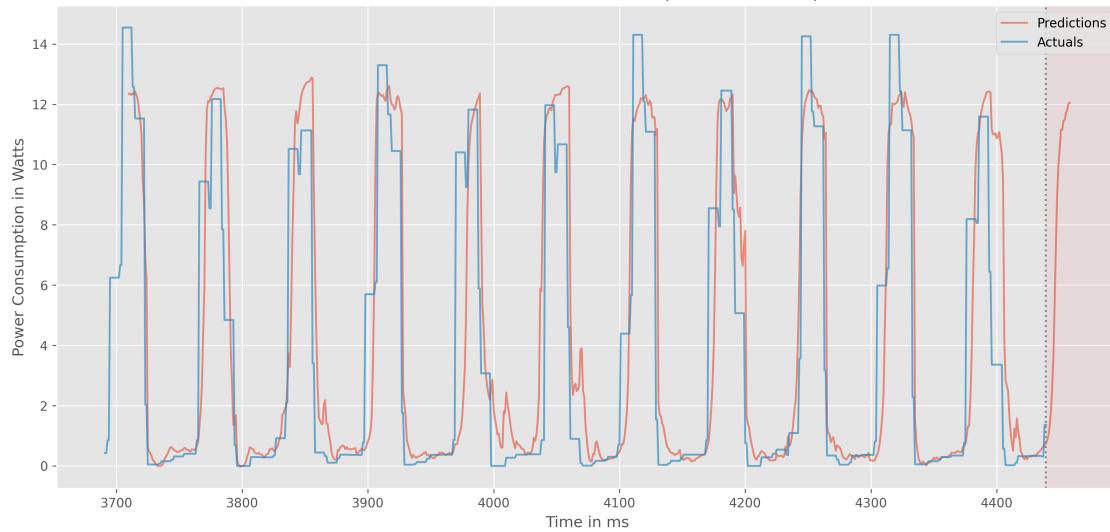


Pow1100 Predictions and Actuals (Discrete Motion)

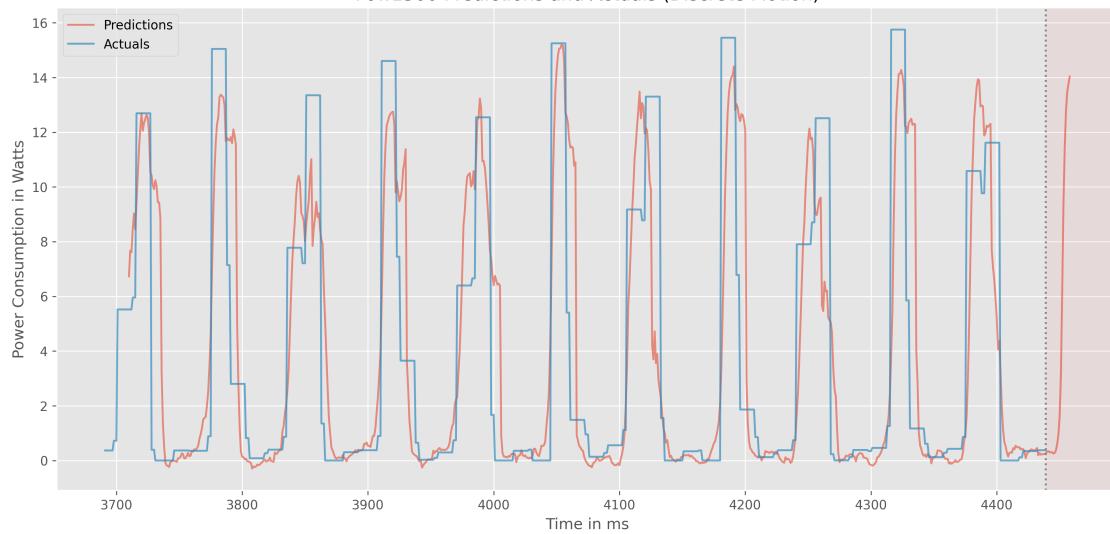




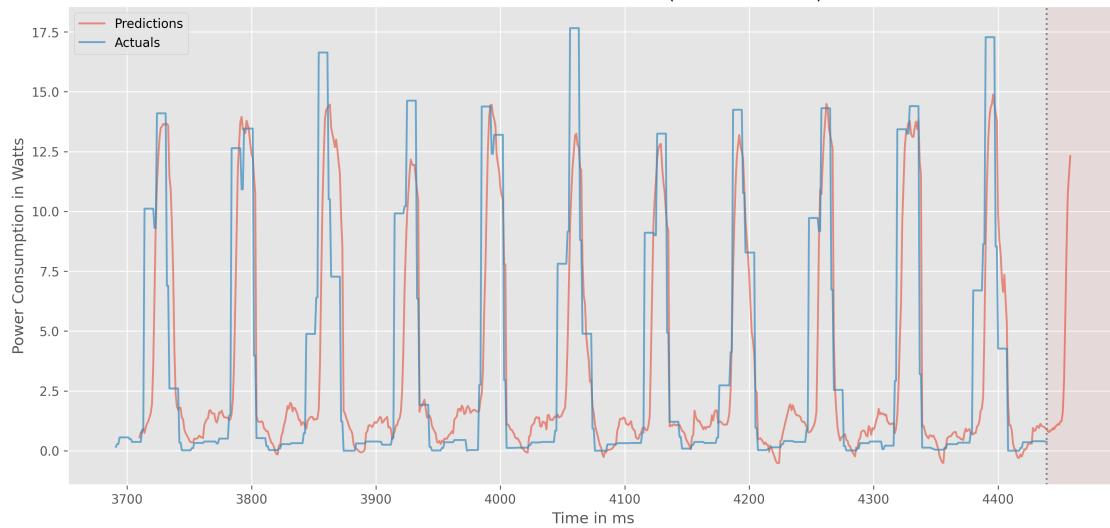
Pow1400 Predictions and Actuals (Discrete Motion)



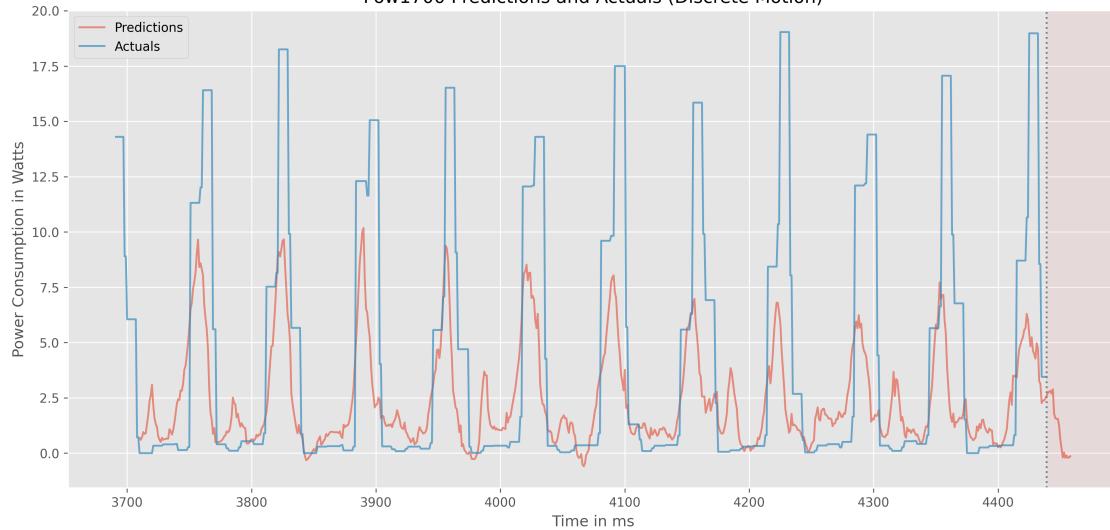
Pow1500 Predictions and Actuals (Discrete Motion)

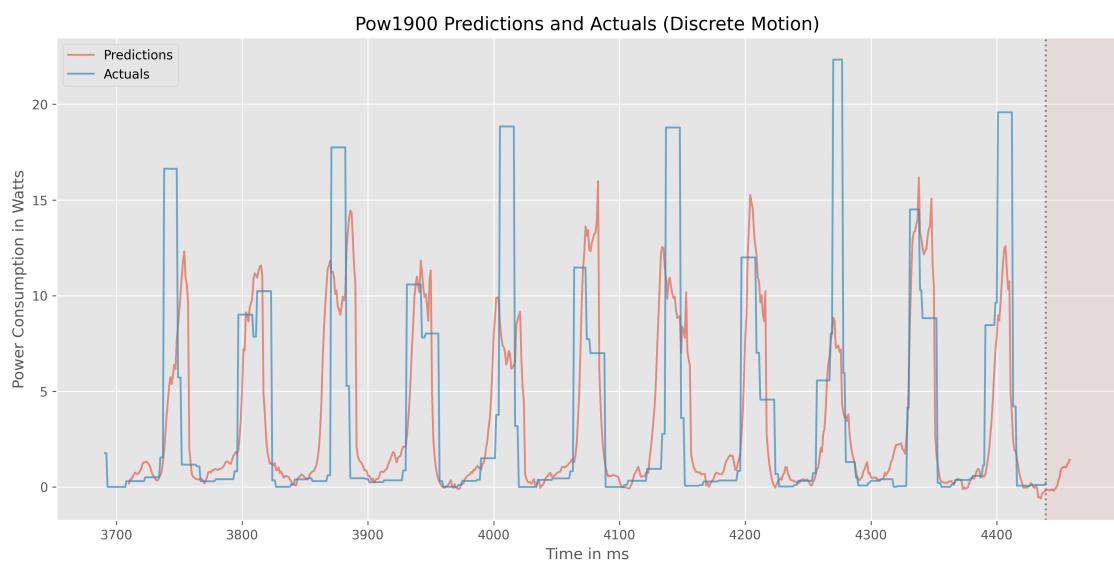
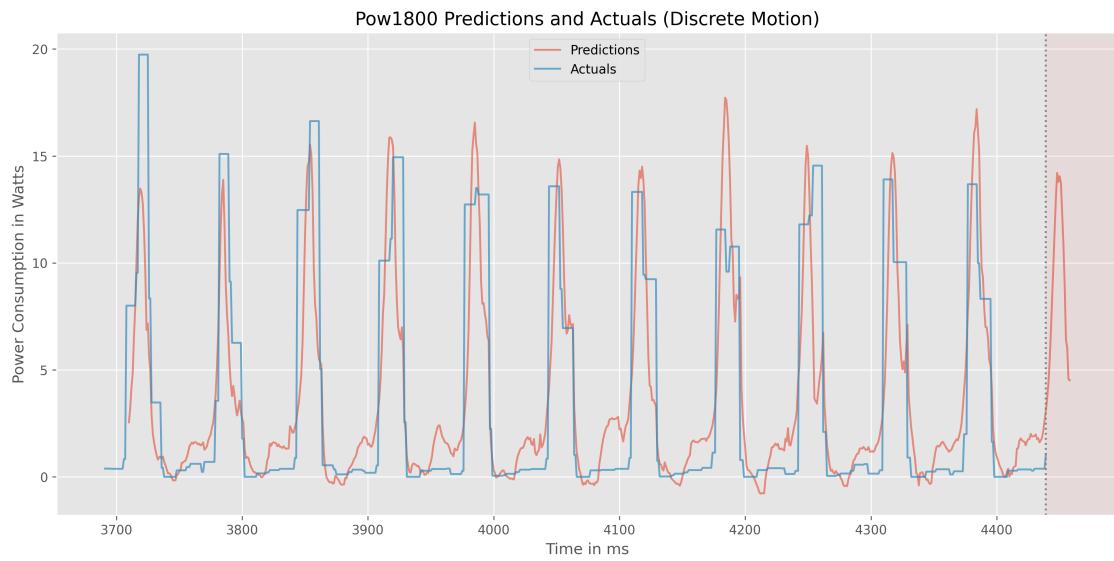


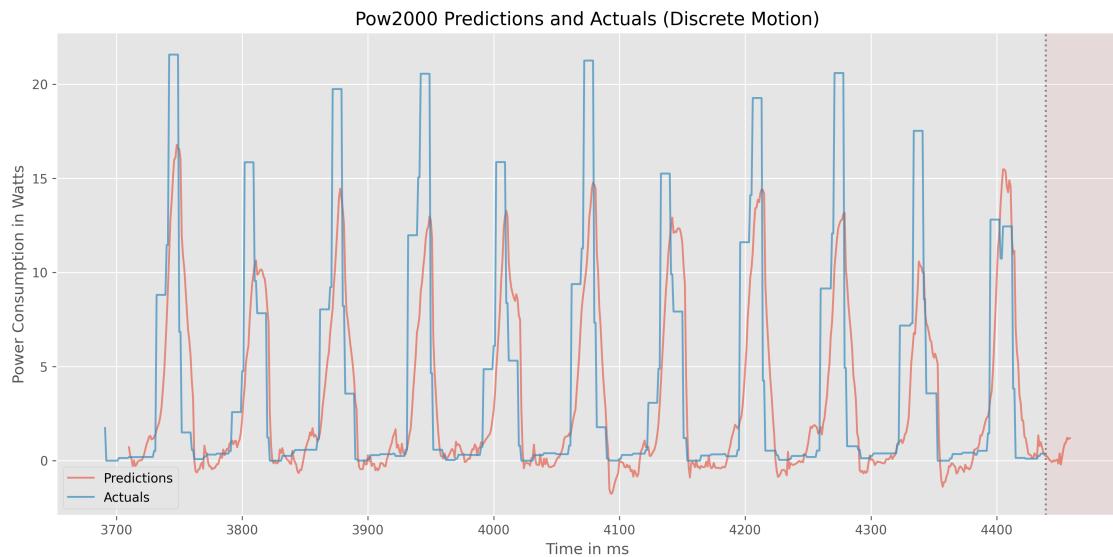
Pow1600 Predictions and Actuals (Discrete Motion)



Pow1700 Predictions and Actuals (Discrete Motion)

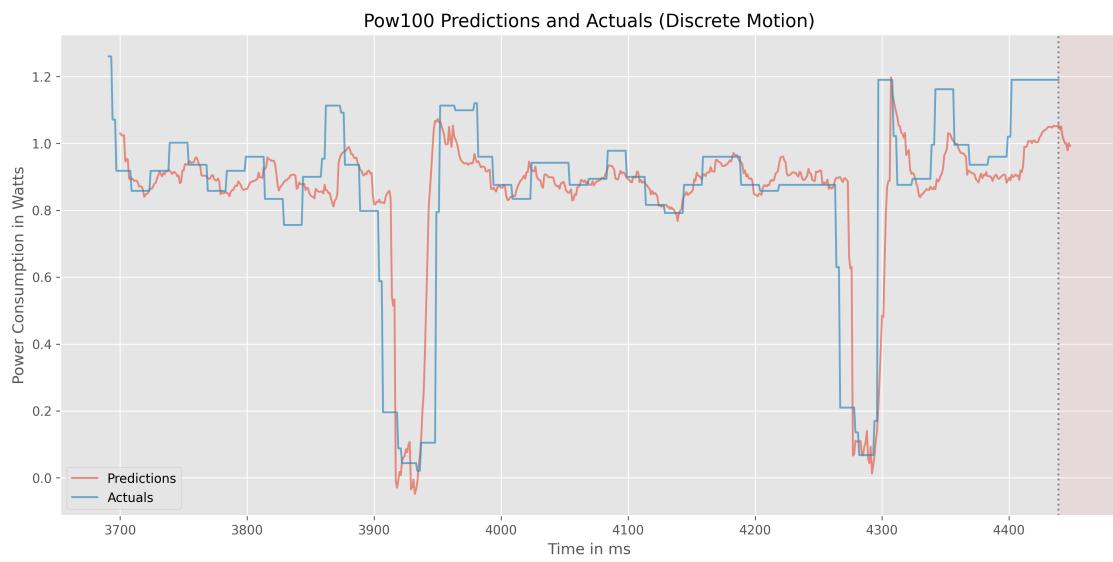


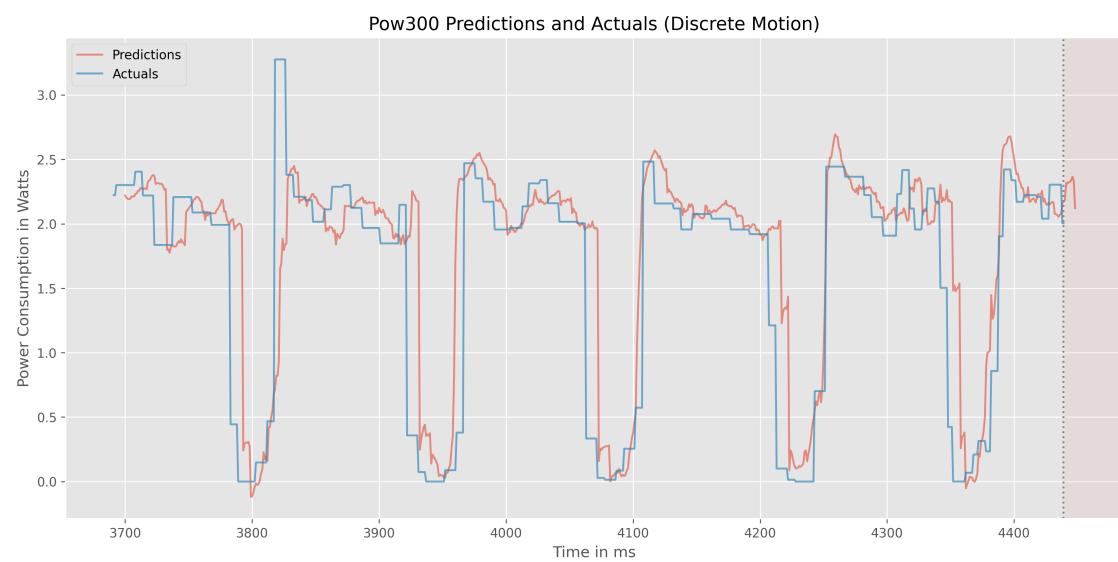
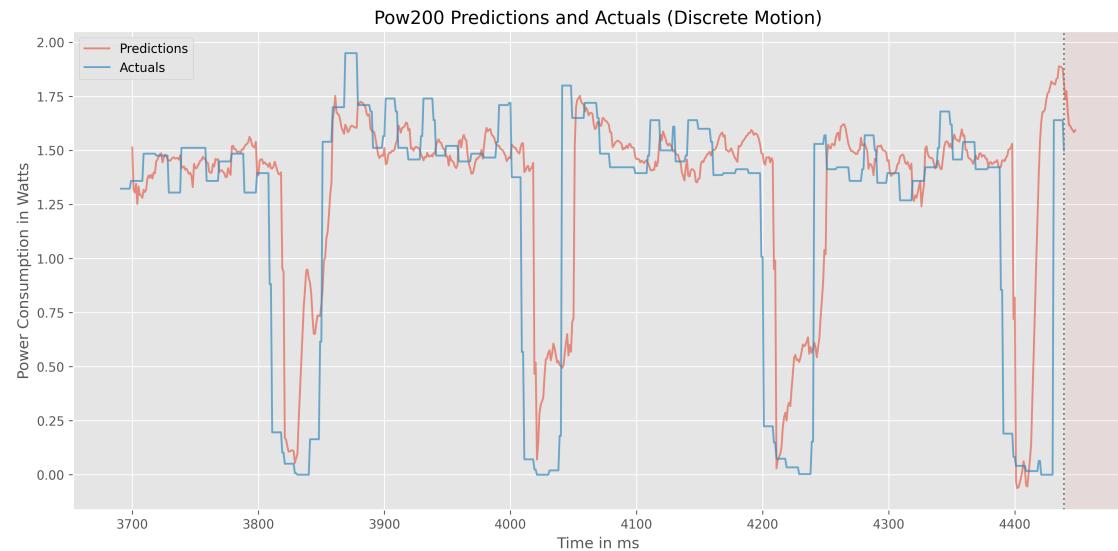




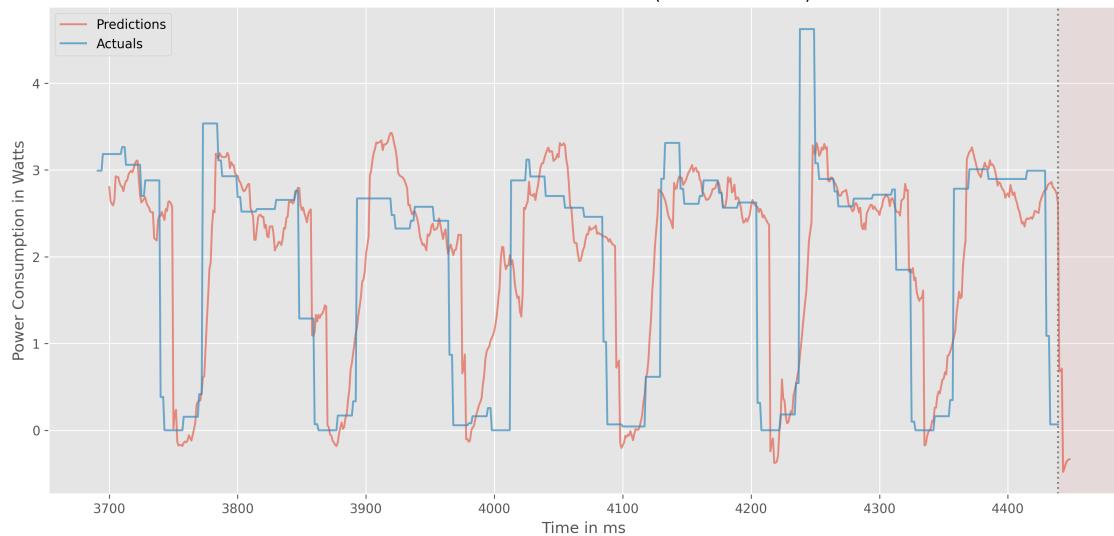
2.2.2 With Forecast Horizon of $t+10$ [Range: last 750 Timestamps]

```
[38]: print_act_and_pred_tables(re_yhat2,re_y_test2,10, -750,-1)
```

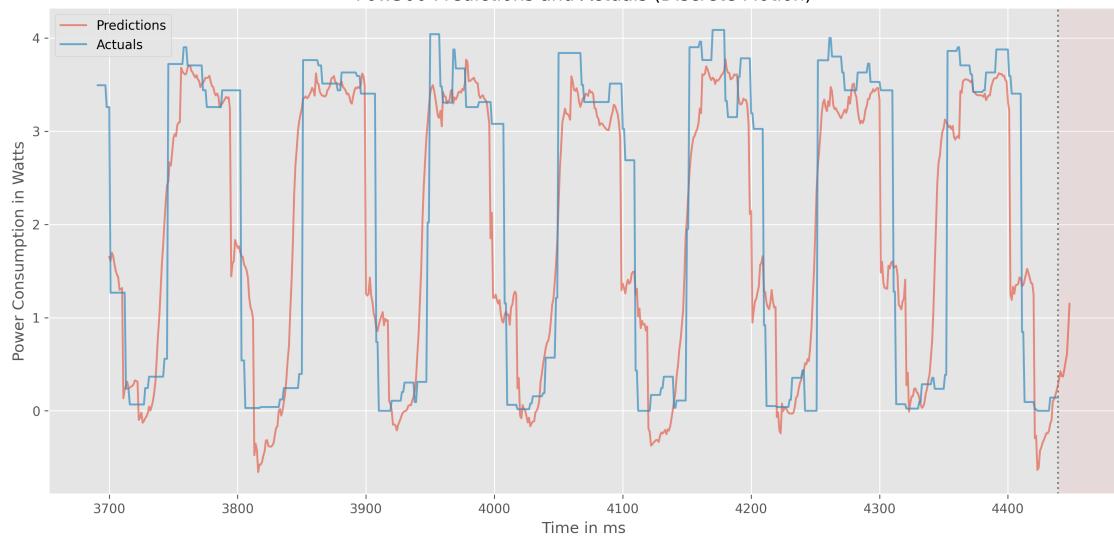




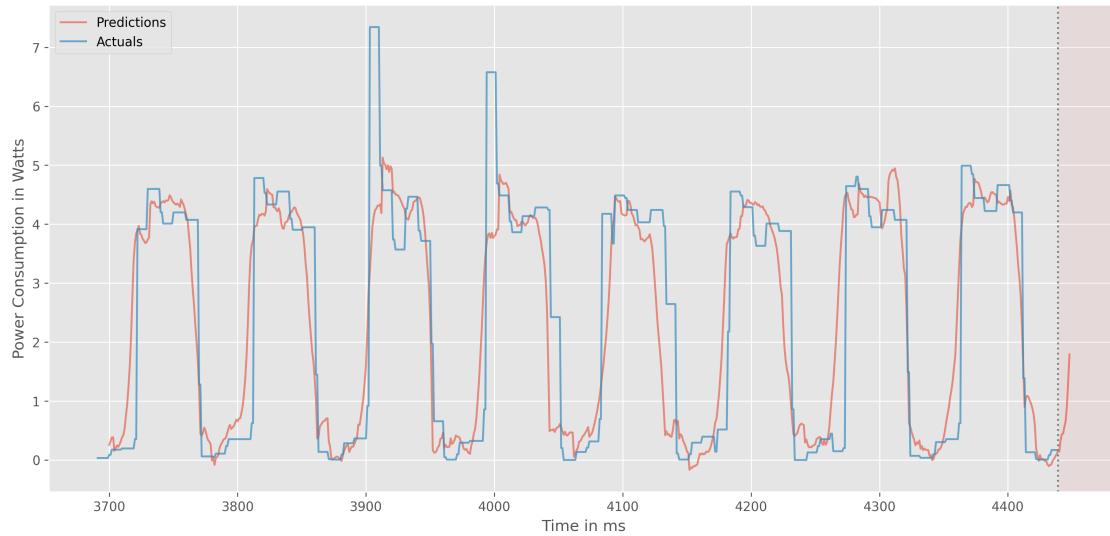
Pow400 Predictions and Actuals (Discrete Motion)



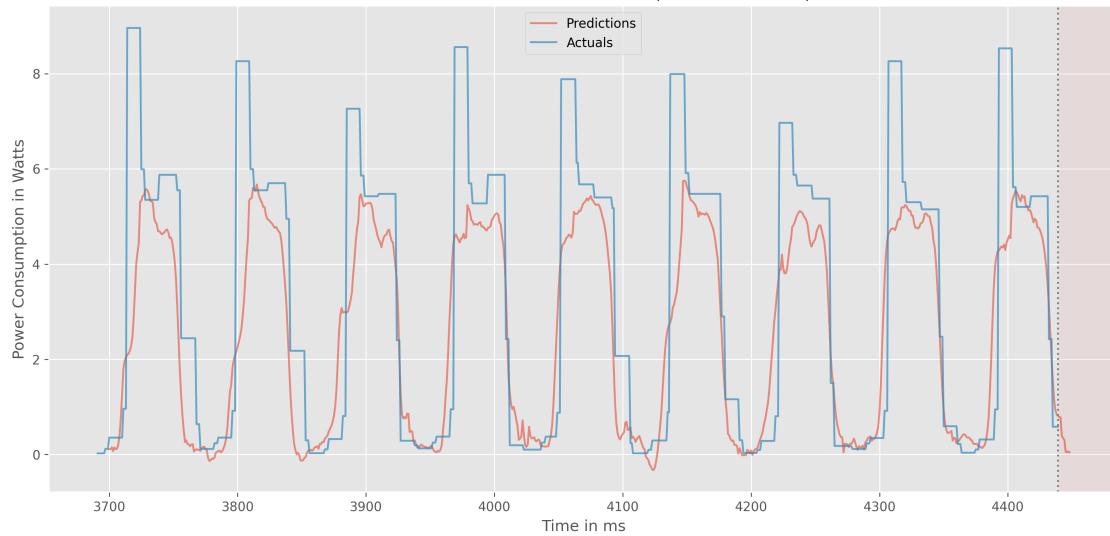
Pow500 Predictions and Actuals (Discrete Motion)



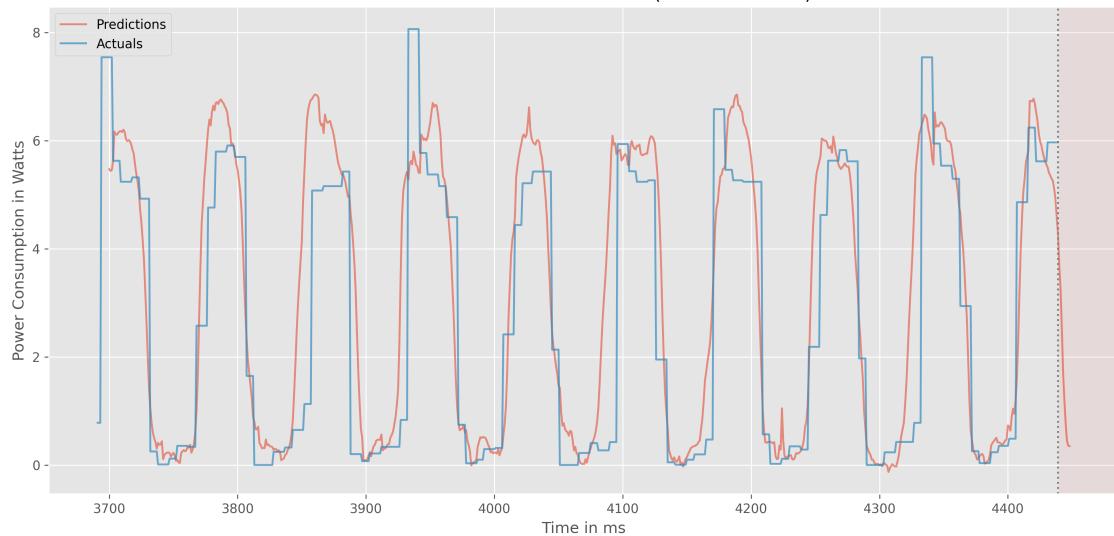
Pow600 Predictions and Actuals (Discrete Motion)



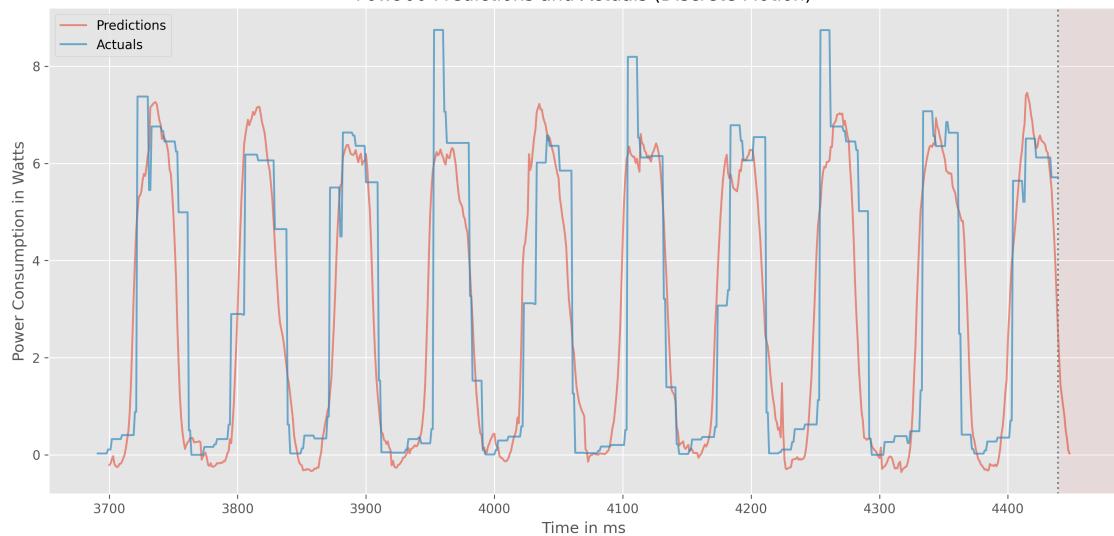
Pow700 Predictions and Actuals (Discrete Motion)

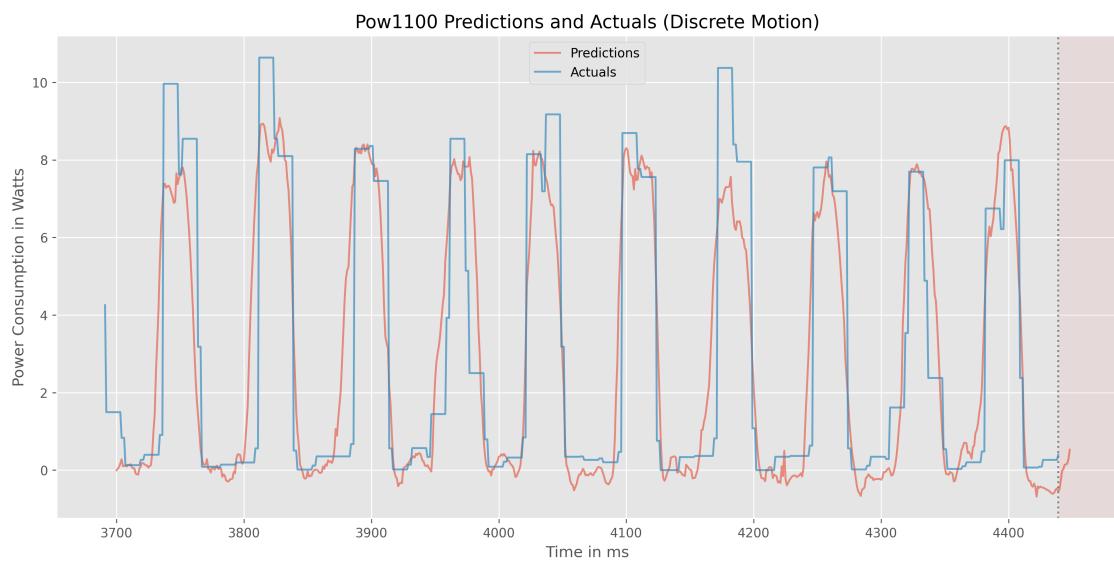
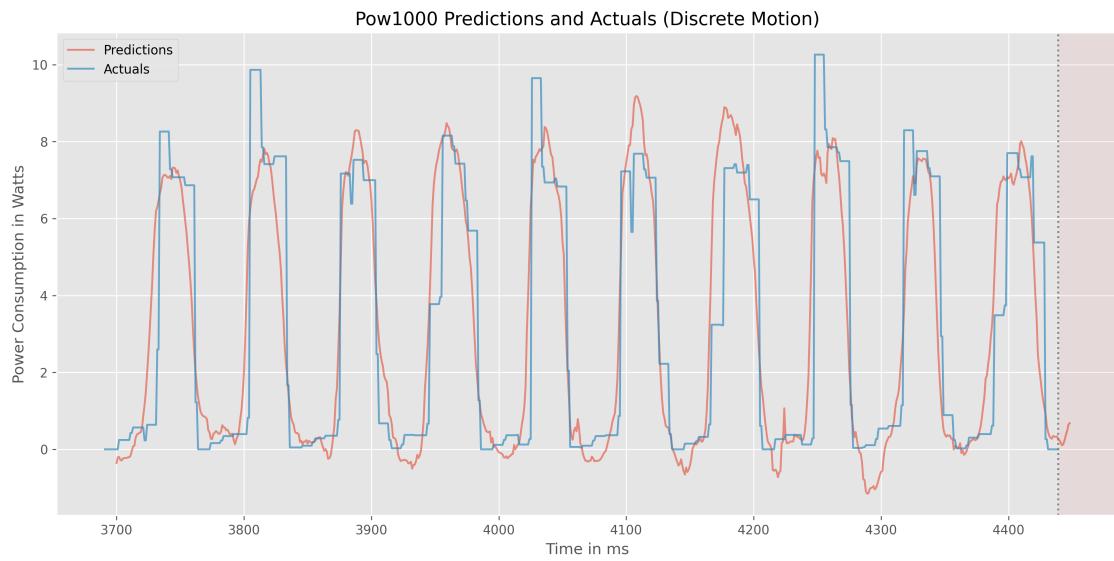


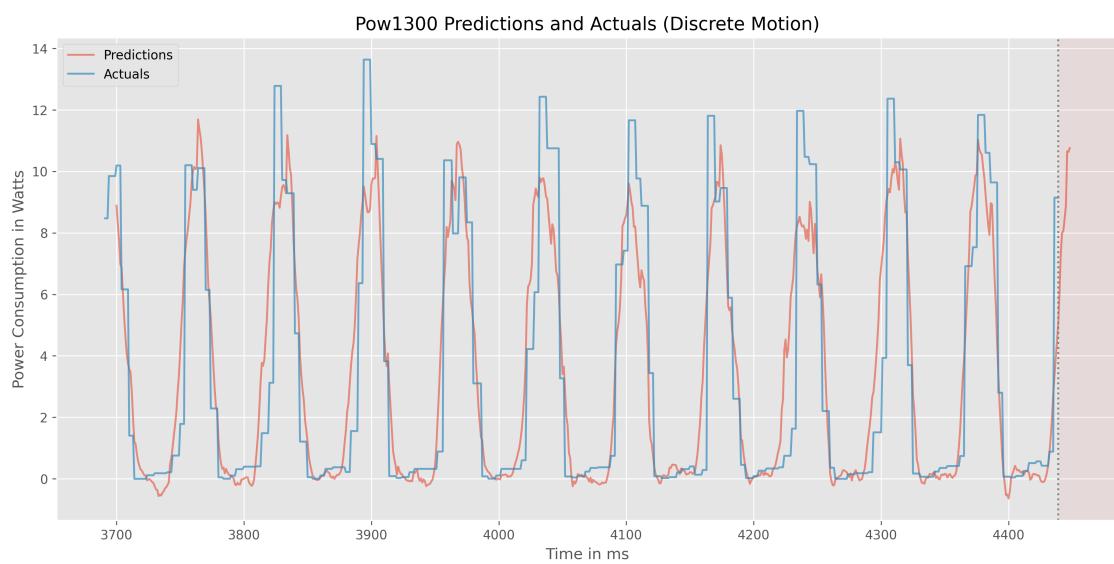
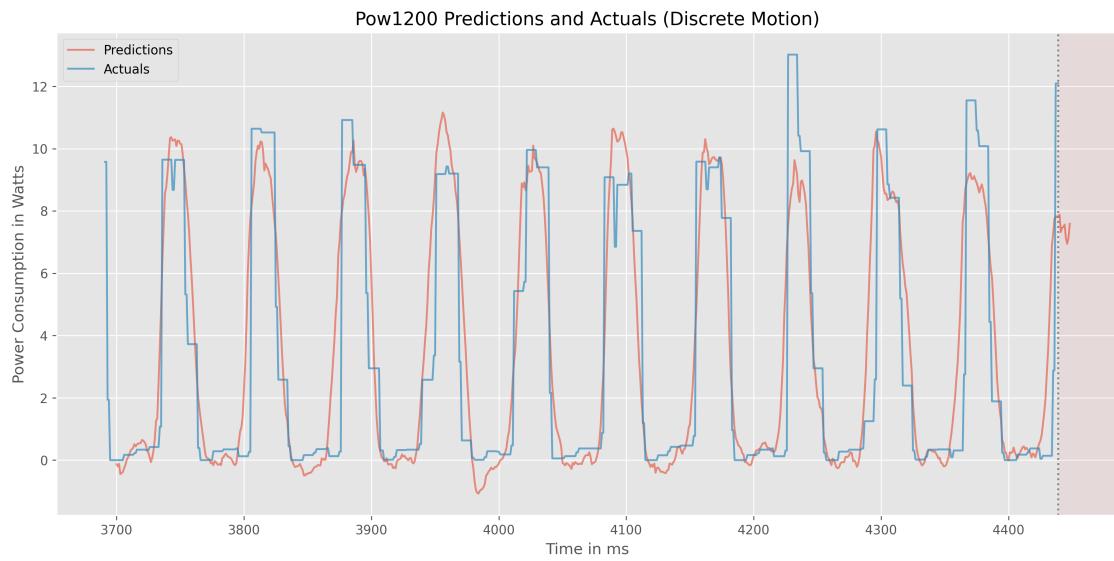
Pow800 Predictions and Actuals (Discrete Motion)



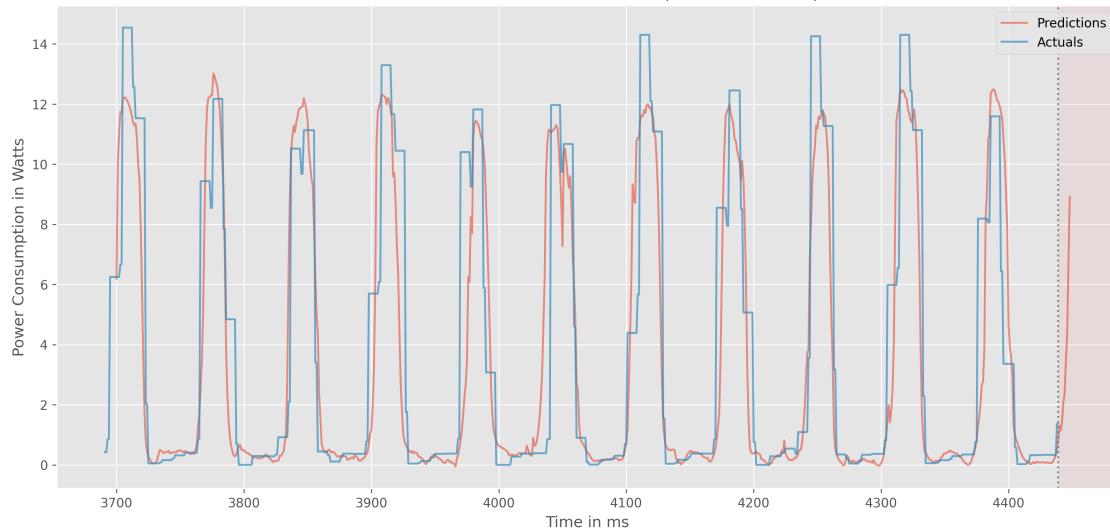
Pow900 Predictions and Actuals (Discrete Motion)



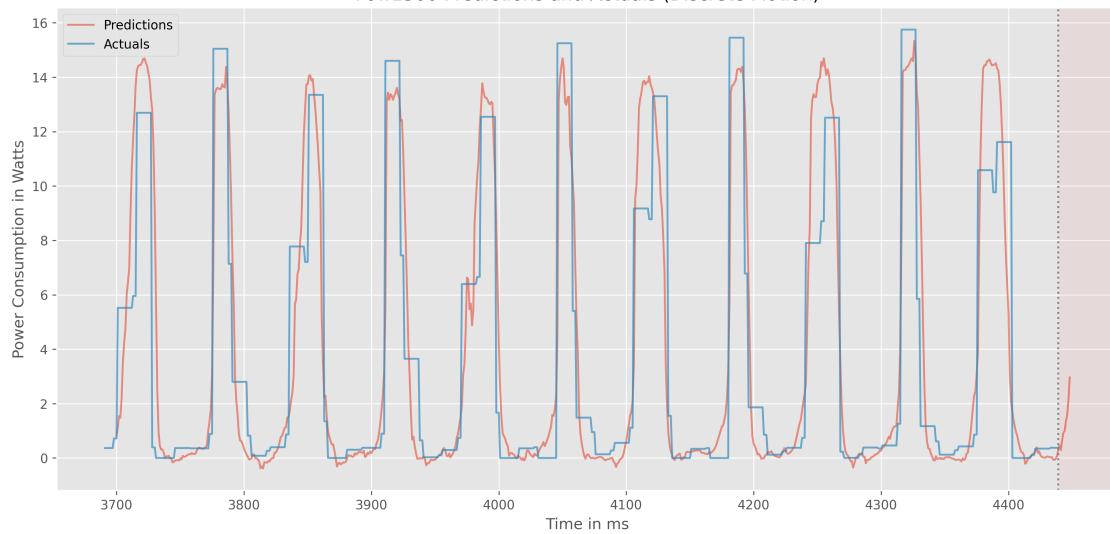


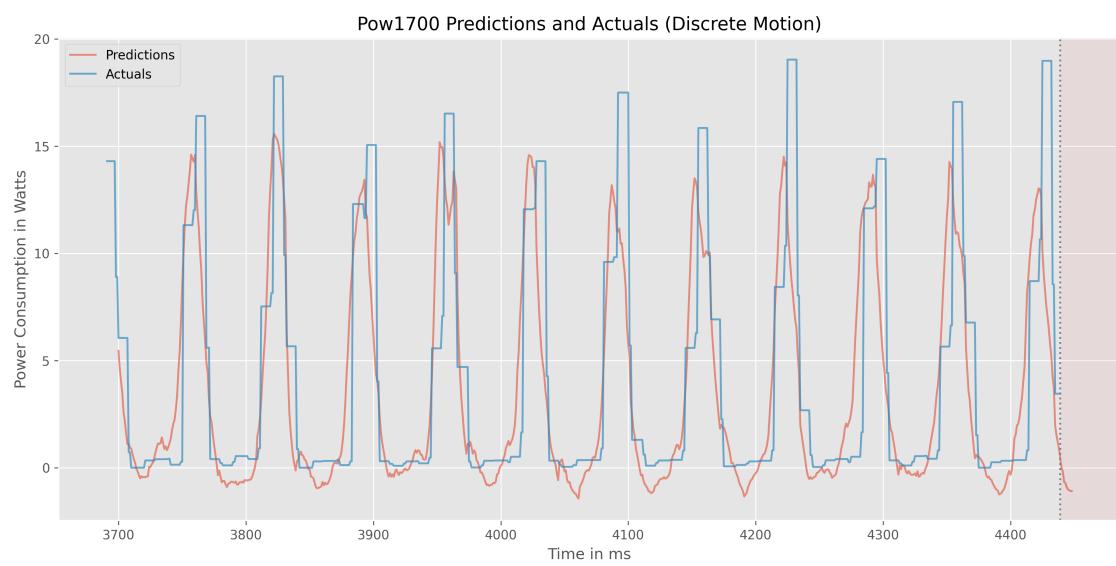
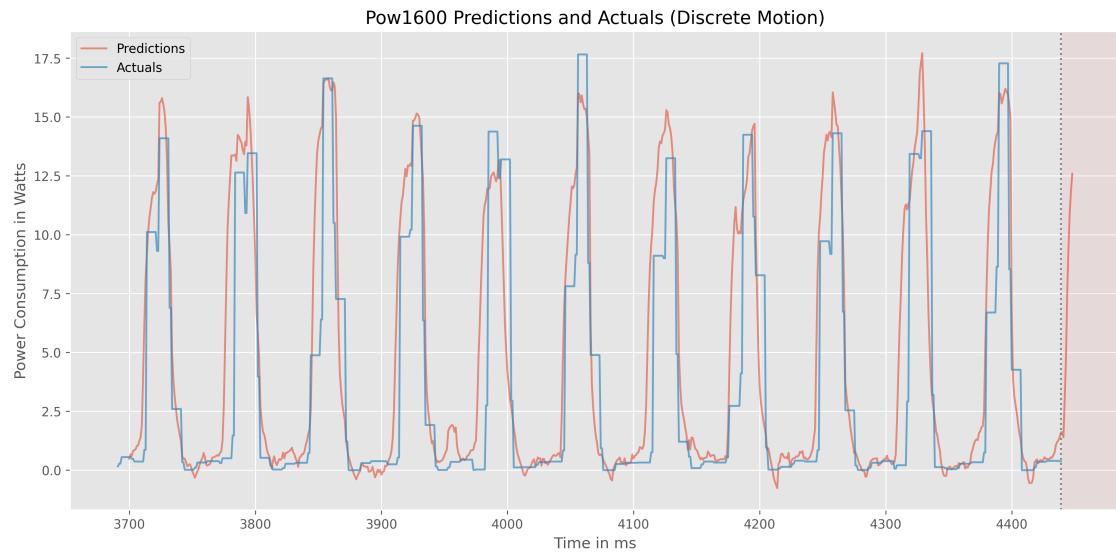


Pow1400 Predictions and Actuals (Discrete Motion)

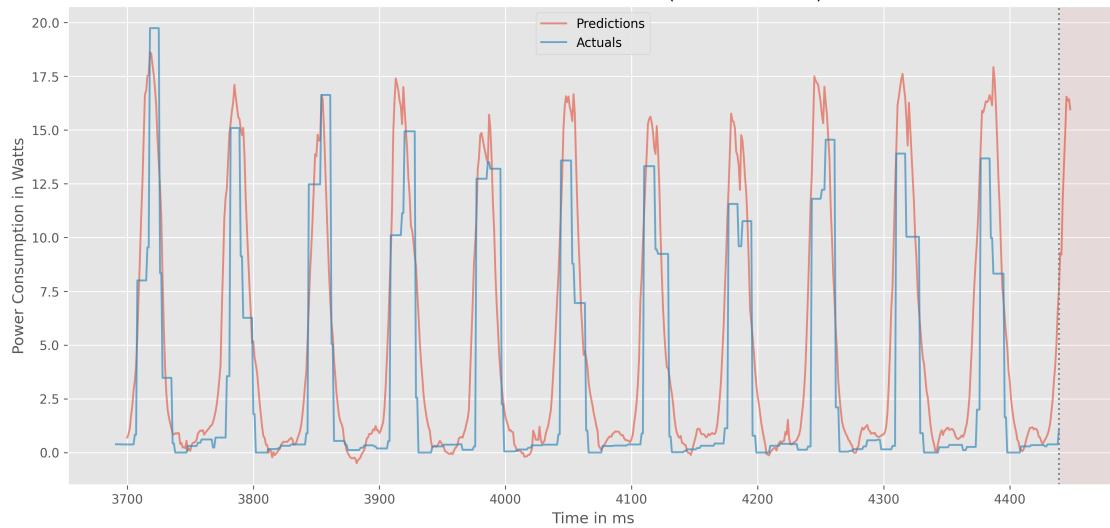


Pow1500 Predictions and Actuals (Discrete Motion)

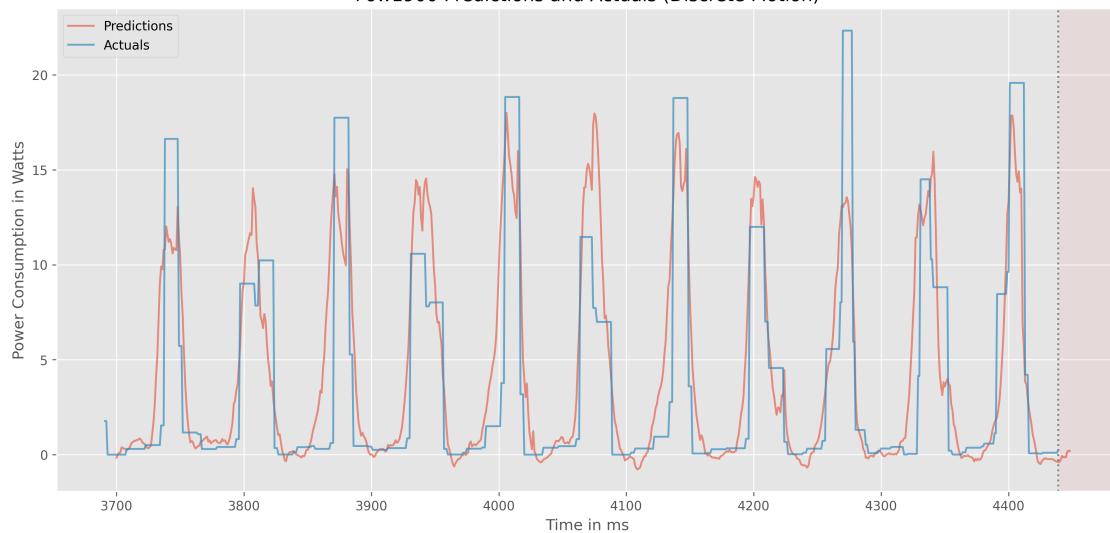


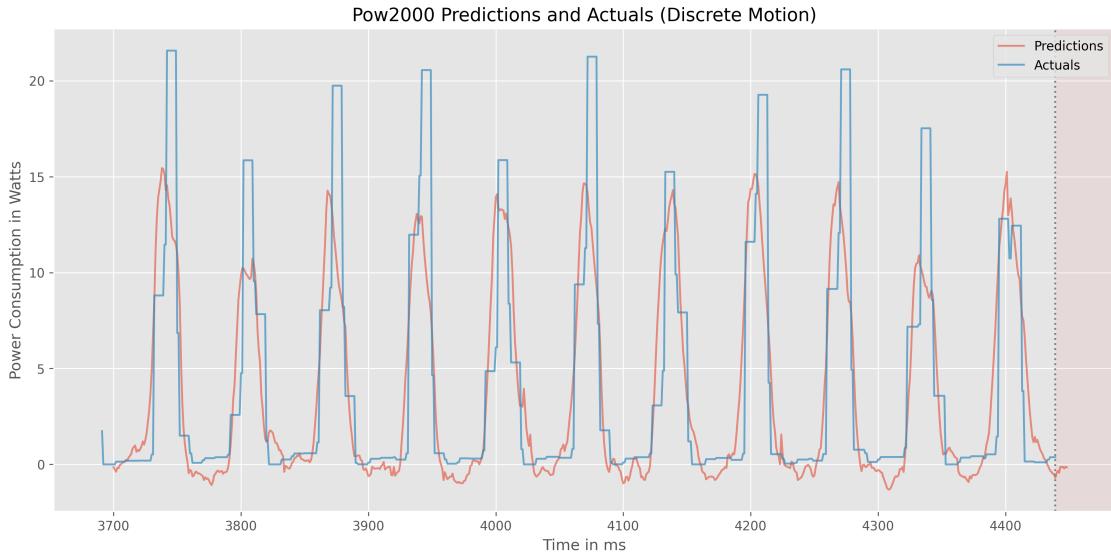


Pow1800 Predictions and Actuals (Discrete Motion)



Pow1900 Predictions and Actuals (Discrete Motion)





2.2.3 With Forecast Horizon of t+20 [Range: All Timestamps]

```
[39]: from sklearn.metrics import r2_score
def print_act_and_pred_tables_all(yhat,ytest,forecast_horizon,✉
    ↵start_graph,end_graph):
    pow_preds = []
    pow_actuals = []
    for i in range(20):
        pow_preds.append(yhat[:, forecast_horizon-1, i]) # 1 refers to the forecast✉
        ↵horizon --> t+1; shape of yhat: [(length), (n_outputs), (n_features)]
        pow_actuals.append(ytest[:, 0, i])

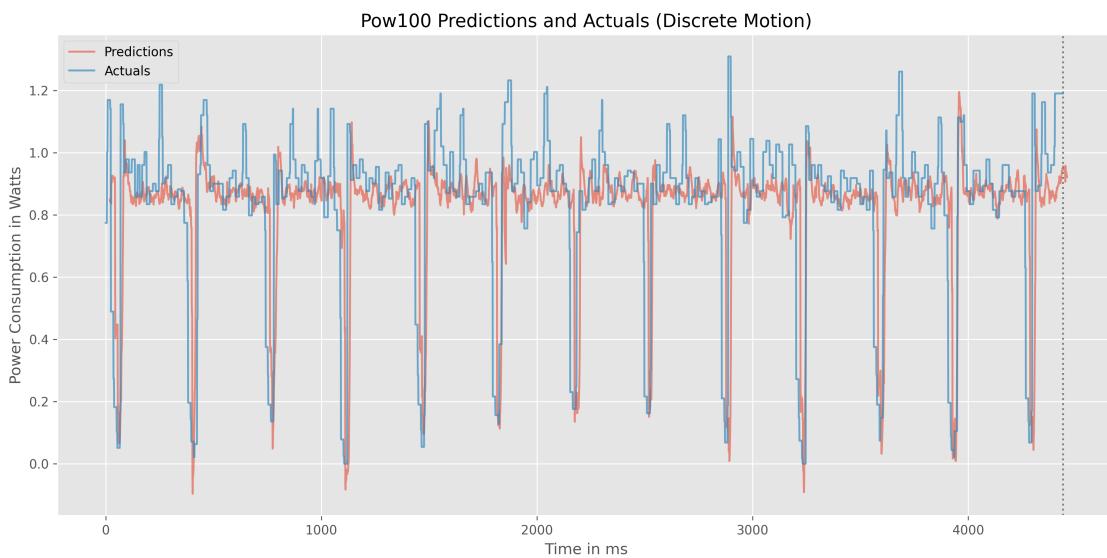
    data = {}
    for i in range(20):
        pow_pred_label = f"Pow{100*(i+1)} Predictions"
        pow_act_label = f"Pow{100*(i+1)} Actuals"
        data[pow_pred_label] = pow_preds[i]
        data[pow_act_label] = pow_actuals[i]
    df_new = pd.DataFrame(data=data)
    #Plot
    for i in range(20):
        fig = plt.figure(figsize=(15, 7))
        plt.style.use("ggplot")
        # Select the actuals and predictions columns
        actuals = df_new[f"Pow{100*(i+1)} Actuals"] [start_graph:end_graph]
        predictions = df_new[f"Pow{100*(i+1)} Predictions"] [start_graph:end_graph]
        # shift the t+20 prediction 20 (relating to the wished forecast horizon) to✉
        ↵the right
```

```

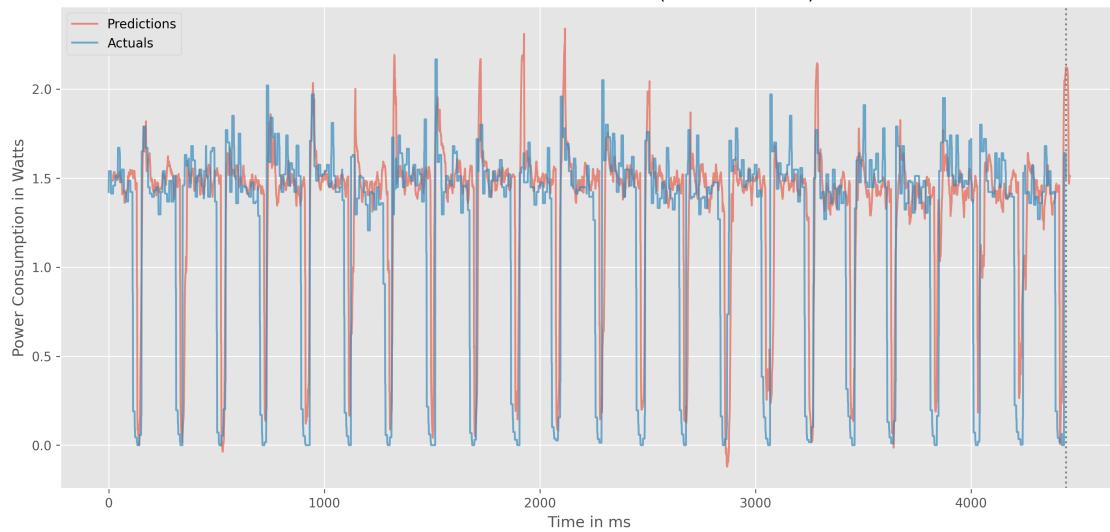
p_temp = predictions.to_frame()
p_temp['time_index_shift'] = predictions.index+forecast_horizon-1
p_temp.set_index('time_index_shift', inplace=True)
# Plot the data
plt.plot(p_temp, alpha=0.6)
plt.plot(actuals, alpha=0.7)
# Set the plot title and axis labels
plt.title(f'Pow{100*(i+1)} Predictions and Actuals (Discrete Motion)')
plt.xlabel('Time in ms')
plt.ylabel('Power Consumption in Watts')
# Draw horizontal Lines for better comparison
plt.axvline(x=4439, linestyle=":", color="grey")
# Set the legend
plt.legend(['Predictions', 'Actuals'])
# Show the plot
plt.show()

```

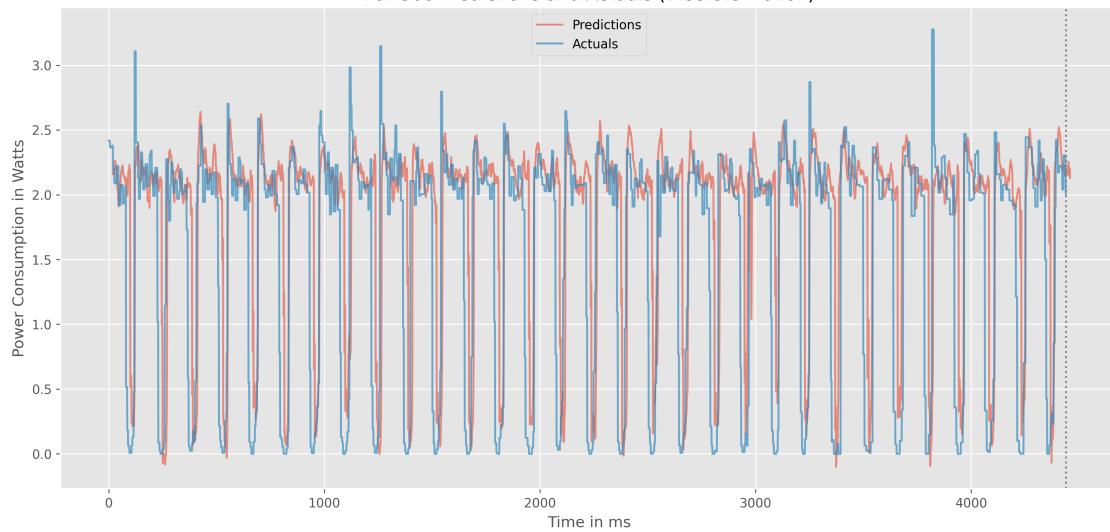
[40]: print_act_and_pred_tables_all(re_yhat2,re_y_test2,20,0,-1)



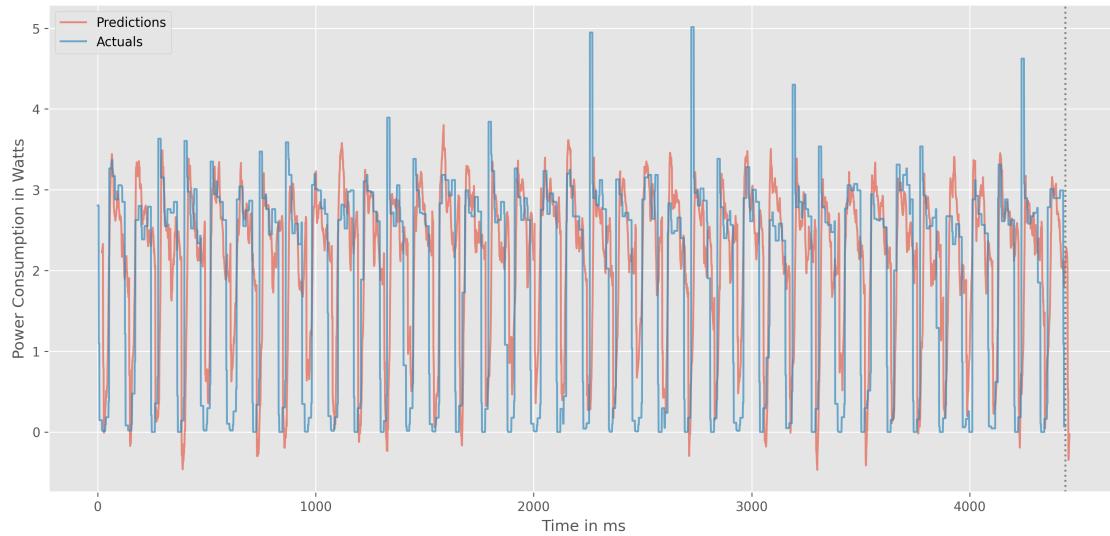
Pow200 Predictions and Actuals (Discrete Motion)



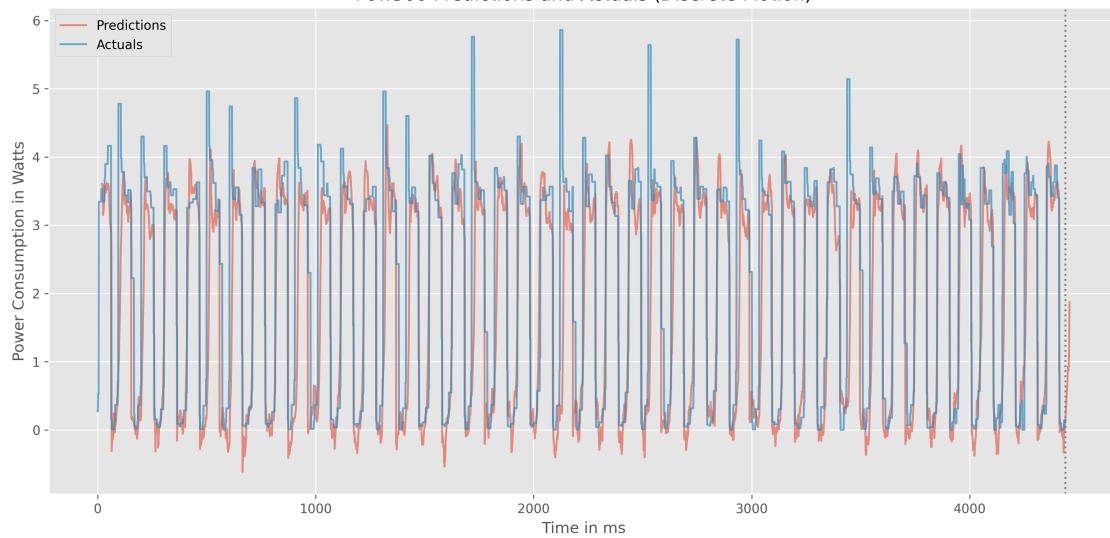
Pow300 Predictions and Actuals (Discrete Motion)

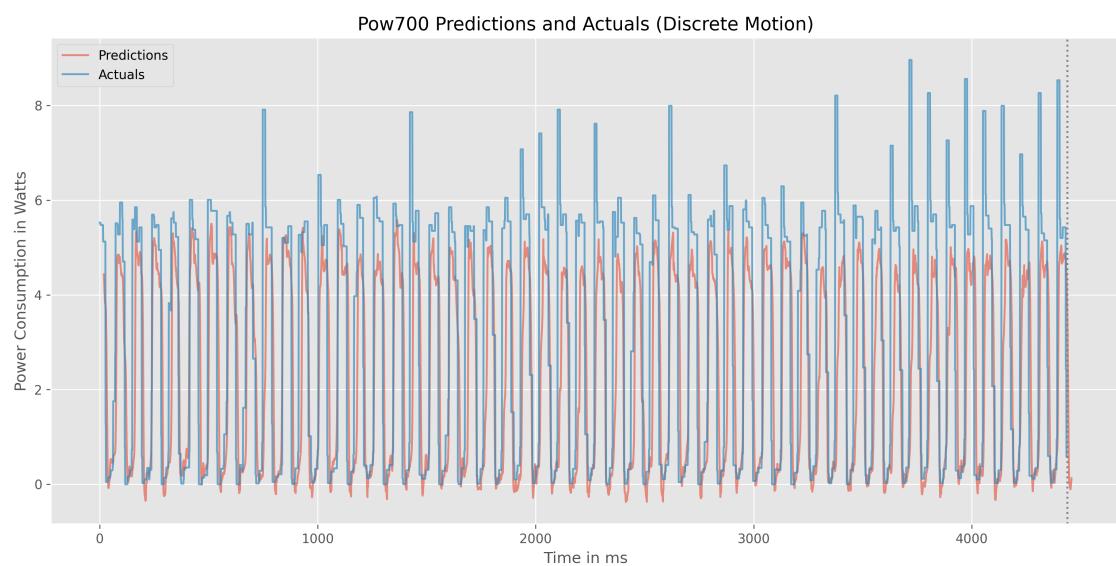
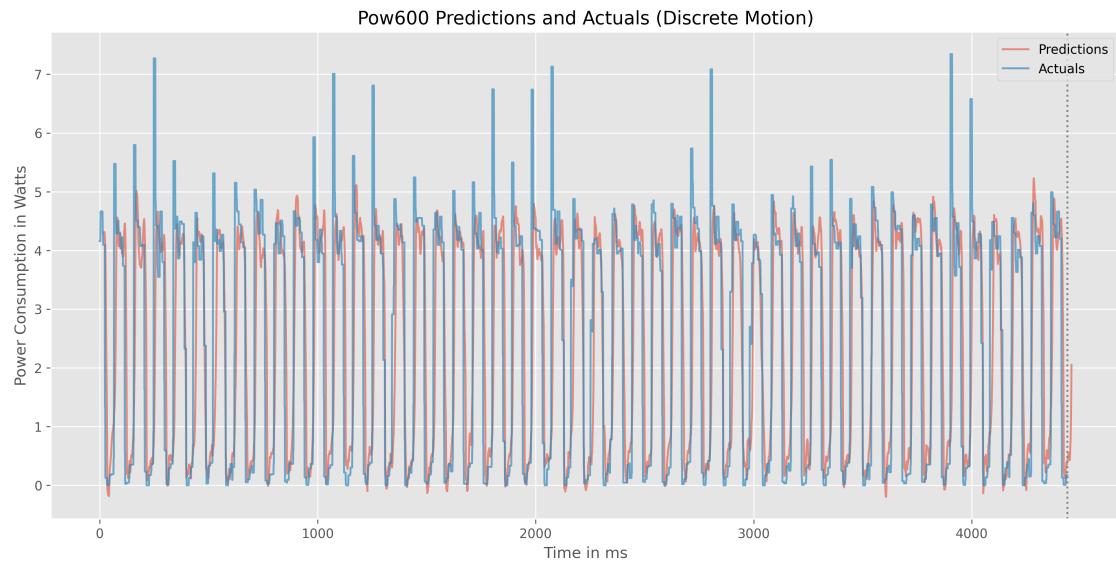


Pow400 Predictions and Actuals (Discrete Motion)

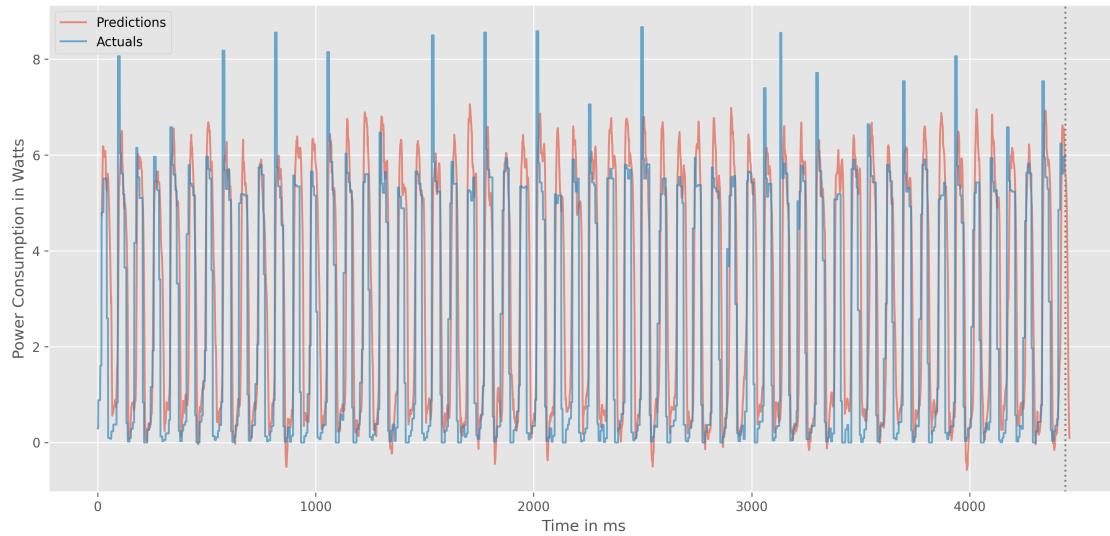


Pow500 Predictions and Actuals (Discrete Motion)

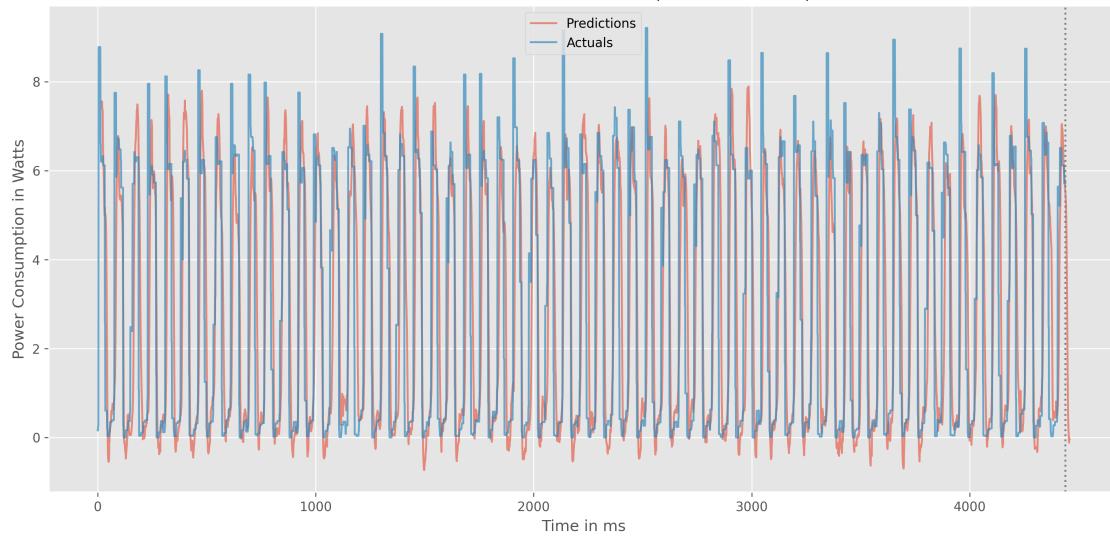


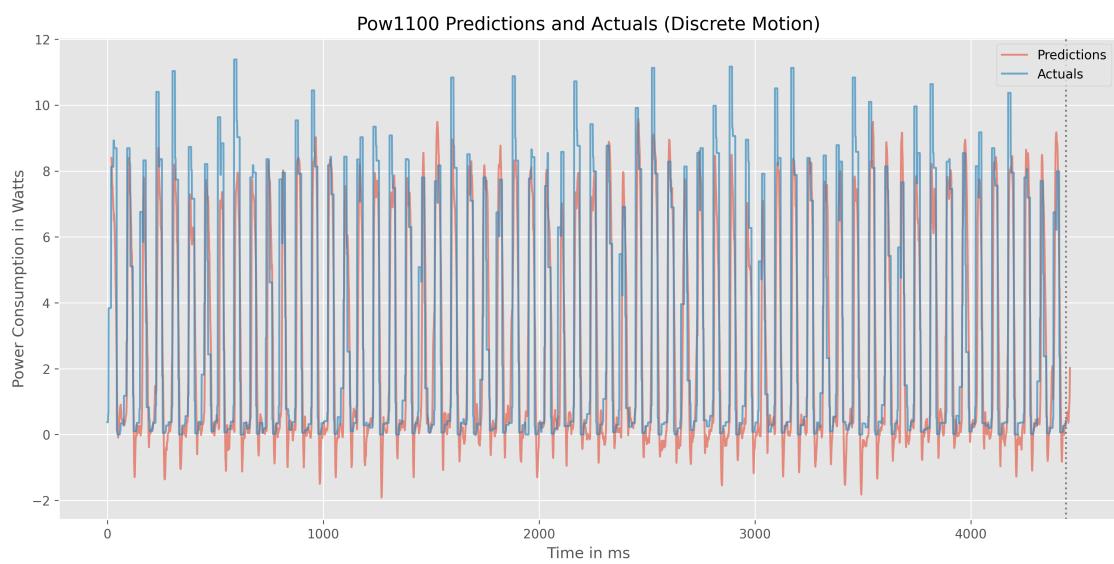
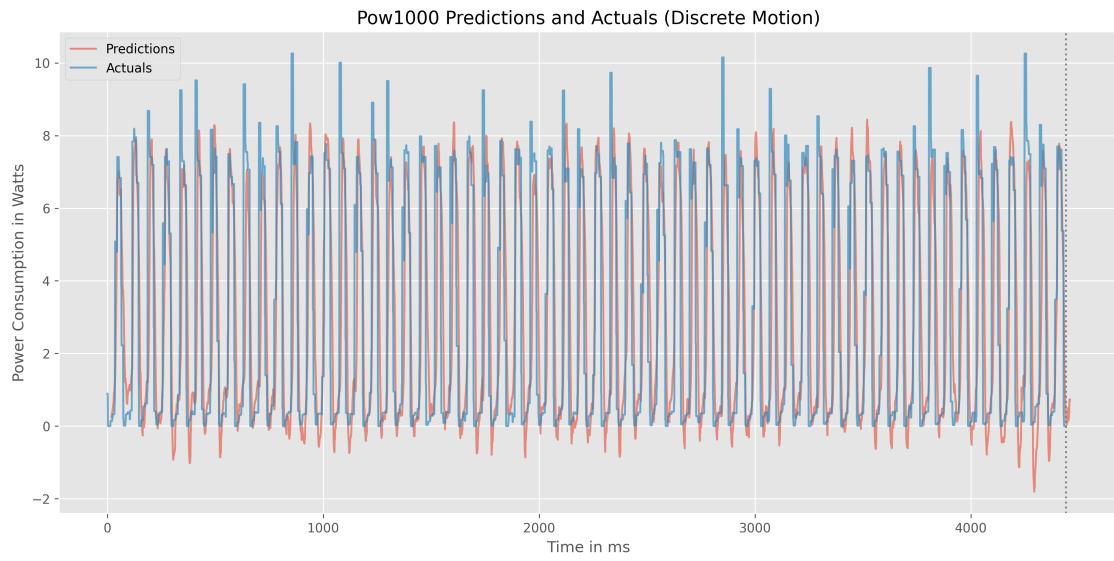


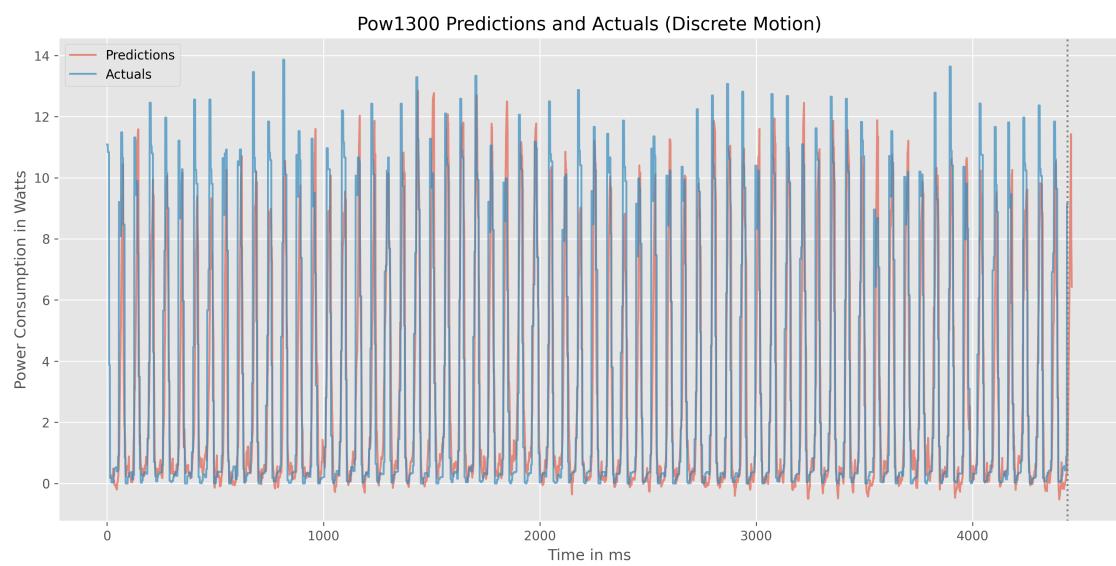
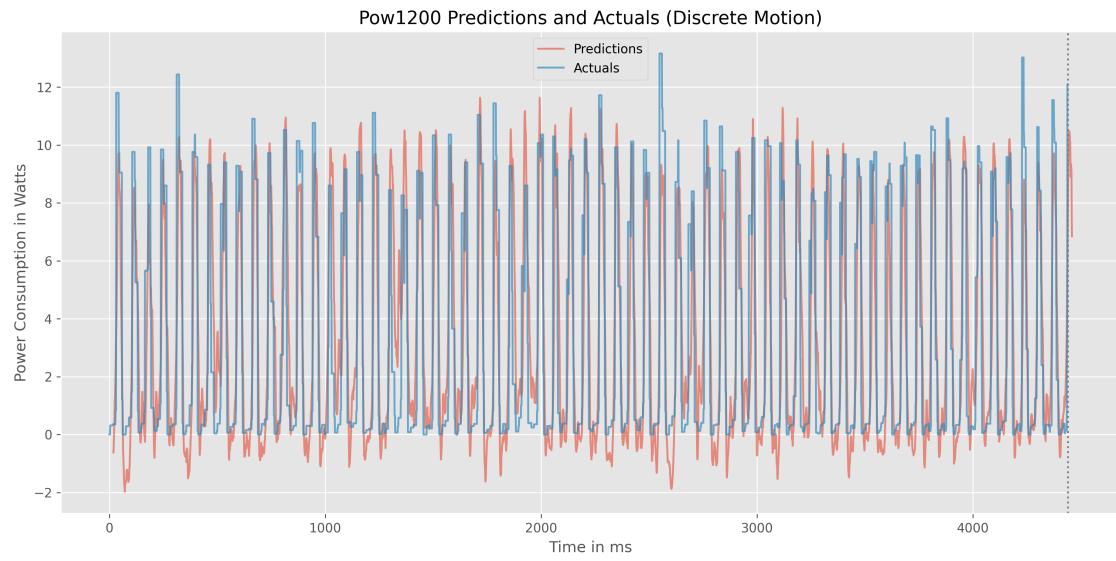
Pow800 Predictions and Actuals (Discrete Motion)

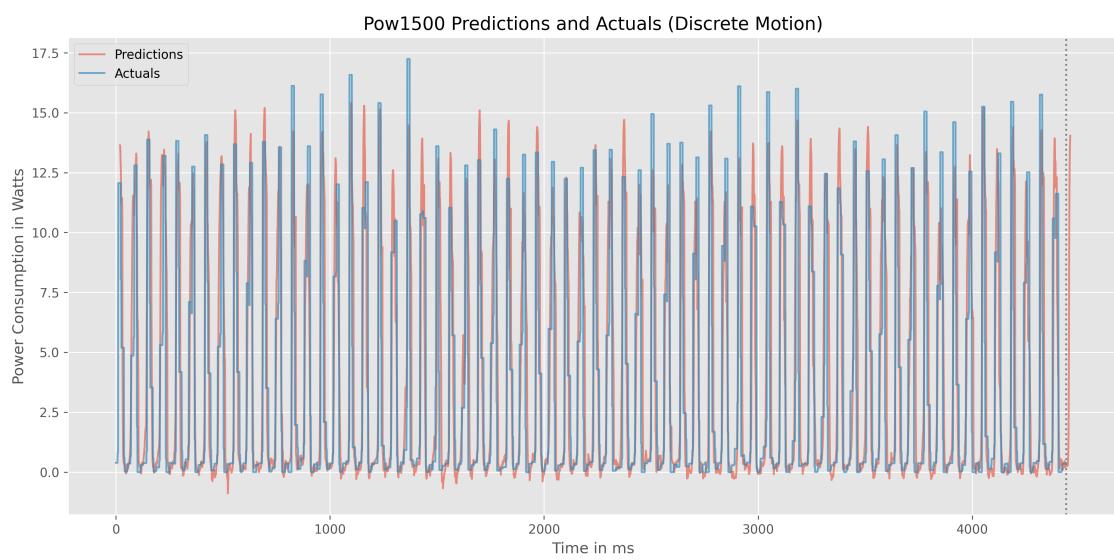
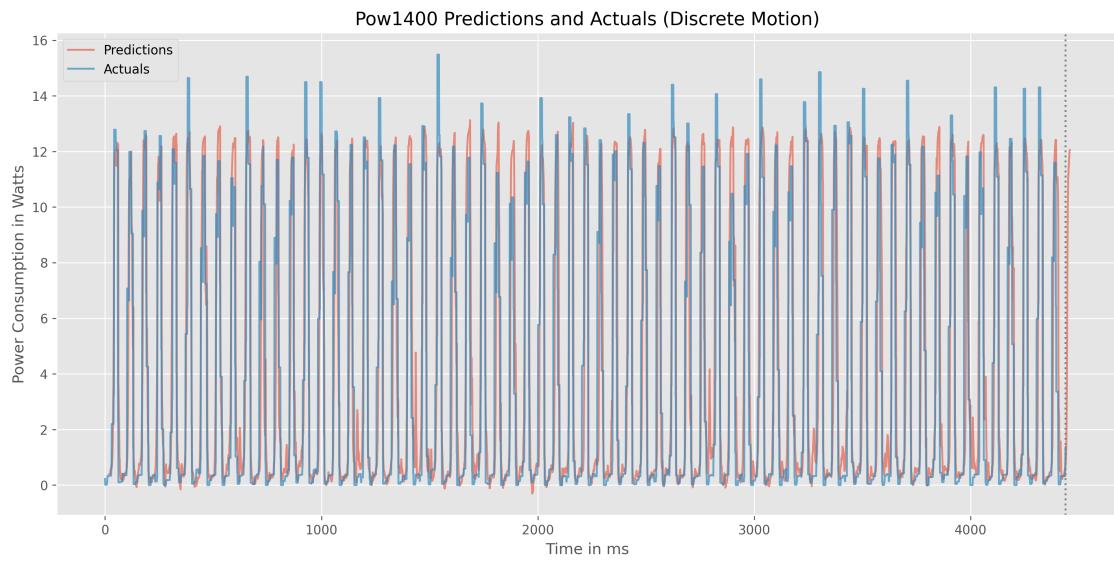


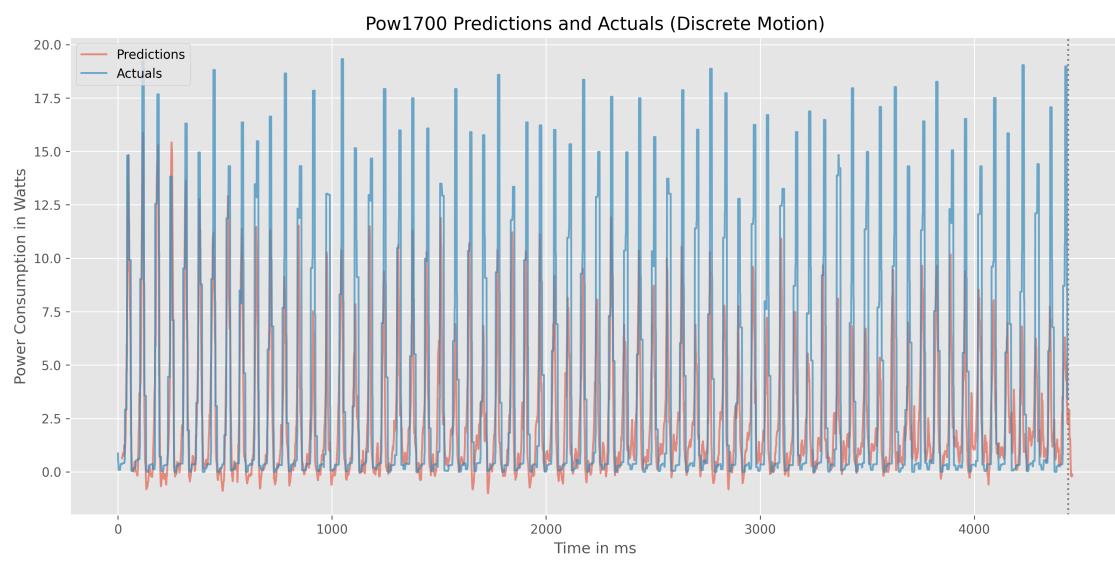
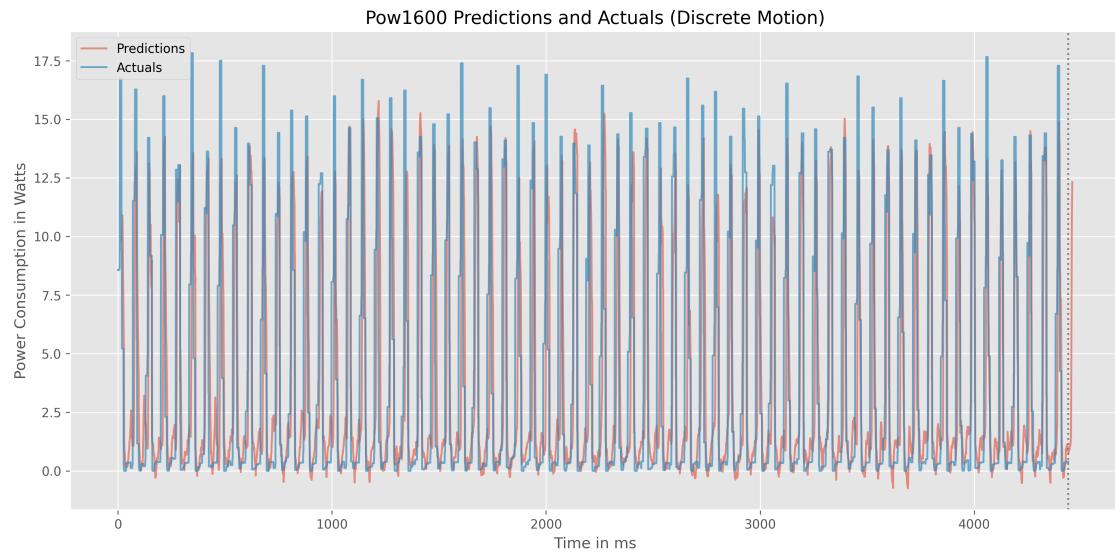
Pow900 Predictions and Actuals (Discrete Motion)

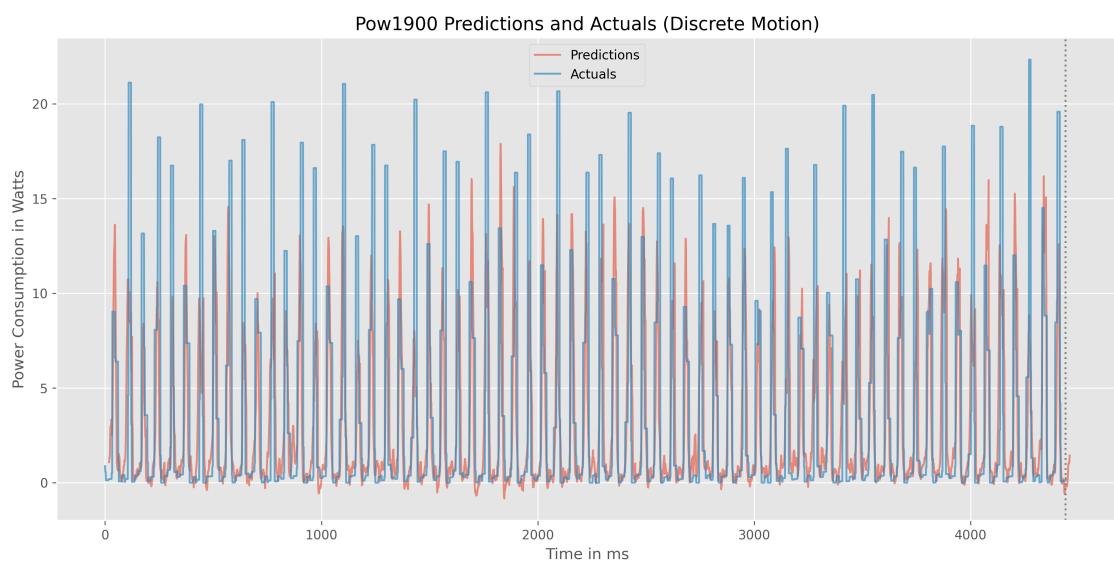
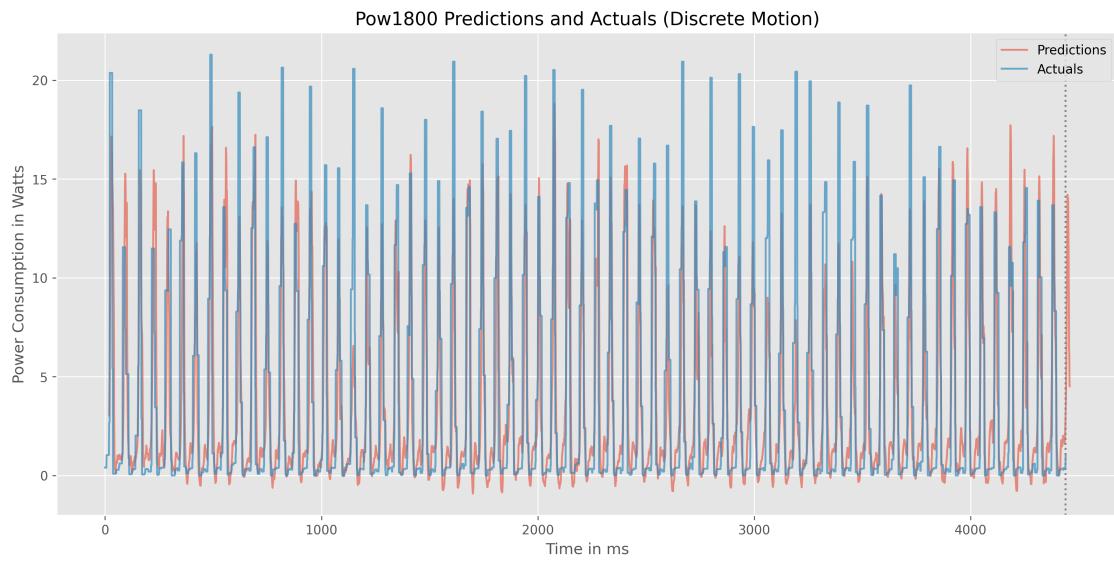


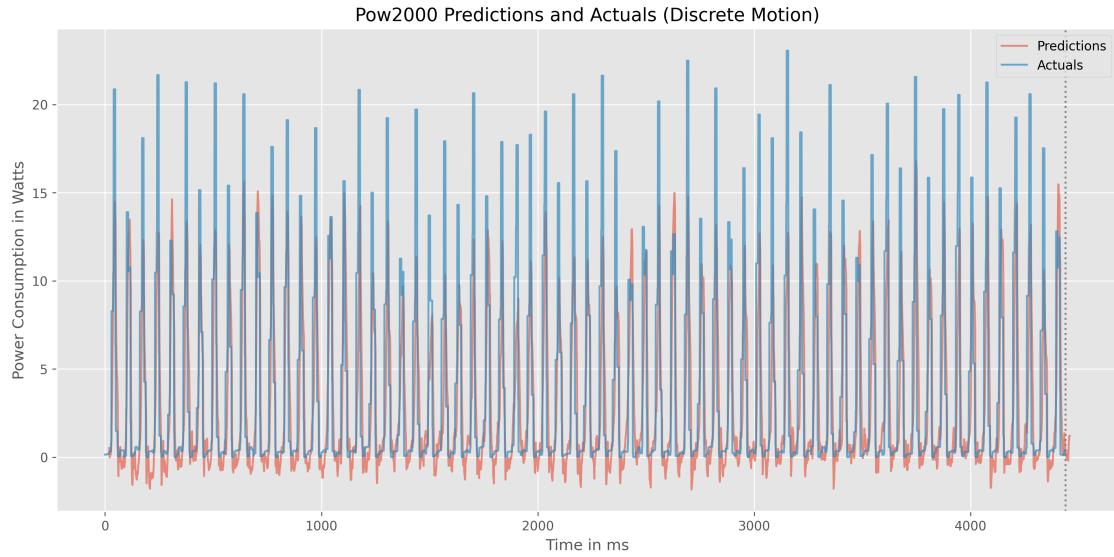












2.2.4 With Forecast Horizon of $t+20$ [Range: last 750 Timestamps] & Further 600ms Prediction (Addon – Recursive Method)

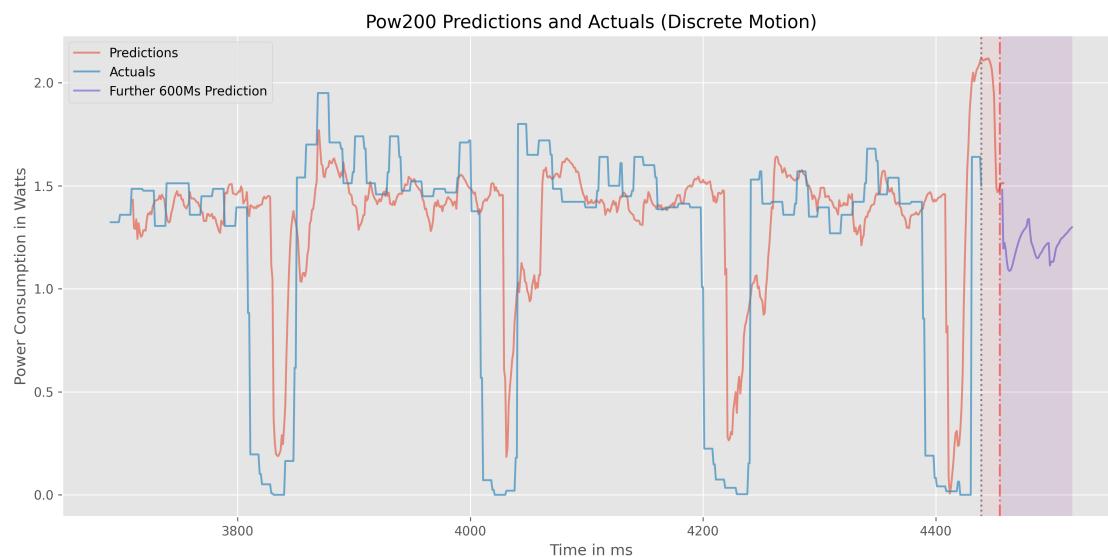
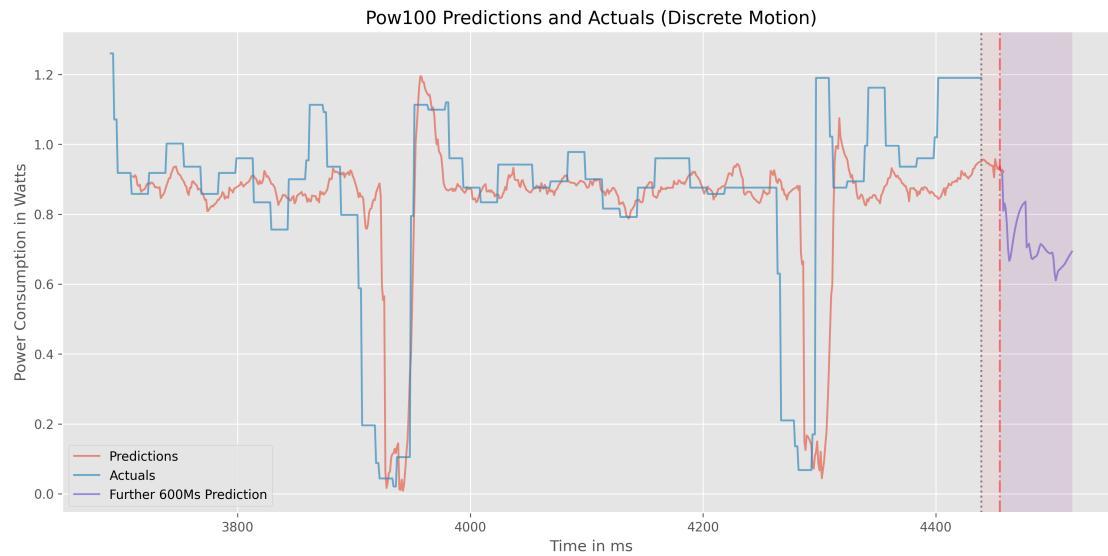
```
[41]: def print_act_and_pred_tables_withFurther(yhat, ytest, forecast_horizon, start_graph, end_graph):
    pow_preds = [yhat[:, forecast_horizon-1, i] for i in range(20)]
    pow_actuals = [ytest[:, 0, i] for i in range(20)]
    data = {}
    for i in range(20):
        data[f"Pow{100*(i+1)} Predictions"] = pow_preds[i]
        data[f"Pow{100*(i+1)} Actuals"] = pow_actuals[i]
    df_new = pd.DataFrame(data=data)
    for i in range(20):
        plt.figure(figsize=(15, 7))
        plt.style.use("ggplot")
        predictions = df_new[f"Pow{100*(i+1)} Predictions"][start_graph:end_graph]
        p_temp = predictions.to_frame()
        p_temp.index = p_temp.index + forecast_horizon - 1
        plt.plot(p_temp, alpha=0.6)
        plt.plot(df_new[f"Pow{100*(i+1)} Actuals"][start_graph:end_graph], alpha=0.
    ↵7)
        plt.plot(f_p[i])
        plt.title(f'Pow{100*(i+1)} Predictions and Actuals (Discrete Motion)')
        plt.xlabel('Time in ms')
        plt.ylabel('Power Consumption in Watts')
        plt.axvline(x=4439, linestyle=":", color="grey")
        plt.axvline(x=4478, linestyle="--", linewidth = 73, color="red", alpha=0.05)
        plt.axvline(x=4486, linestyle="--", linewidth = 57, color="blue", alpha=0.05)
```

```

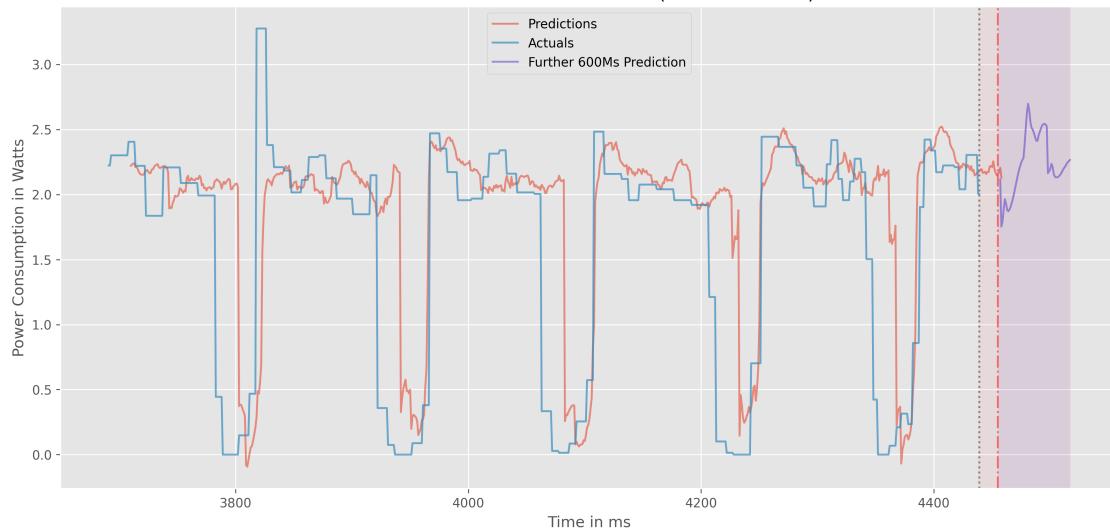
plt.axvline(x=4455, linestyle="-.", color="red", alpha=0.5)
plt.legend(['Predictions', 'Actuals', "Further 600Ms Prediction"])
plt.show()

```

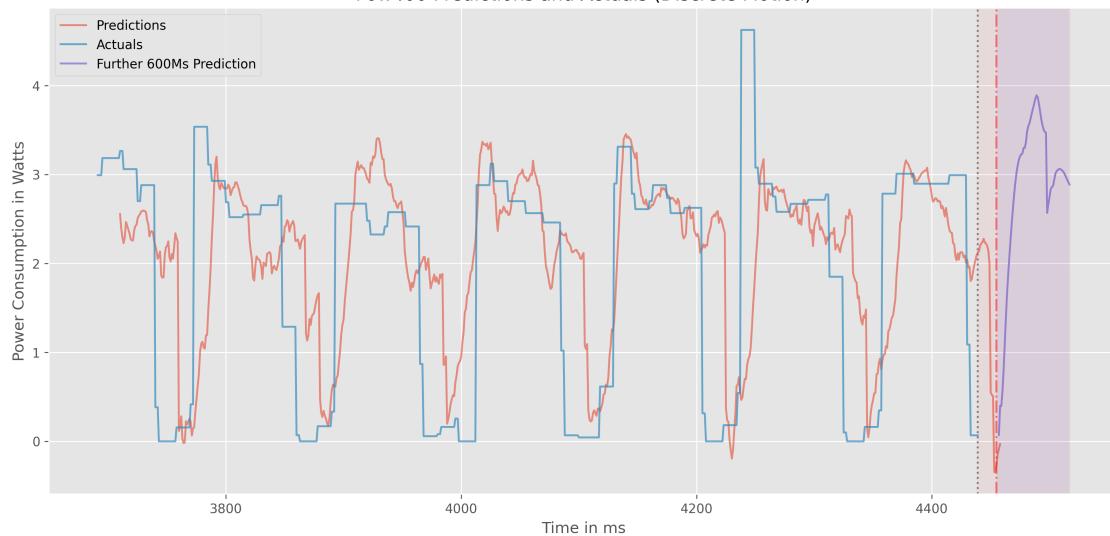
[42]: print_act_and_pred_tables_withFurther(re_yhat2,re_y_test2,20, -750,-1)



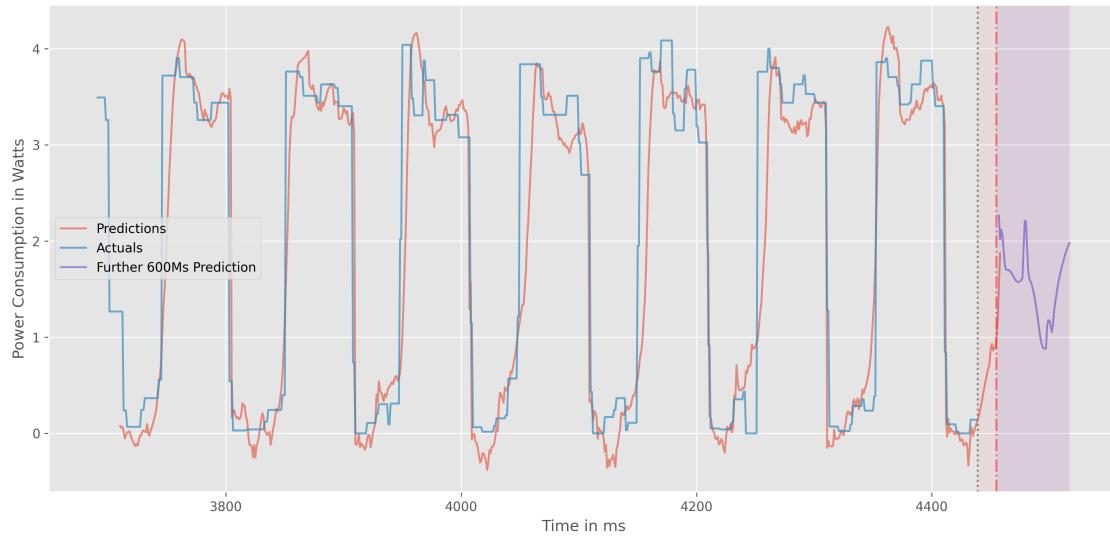
Pow300 Predictions and Actuals (Discrete Motion)



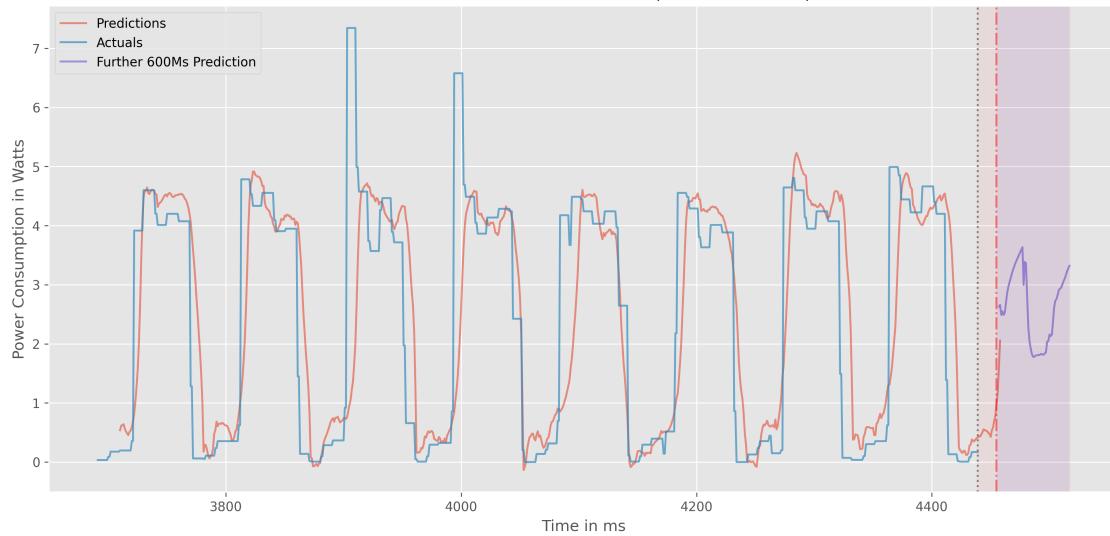
Pow400 Predictions and Actuals (Discrete Motion)

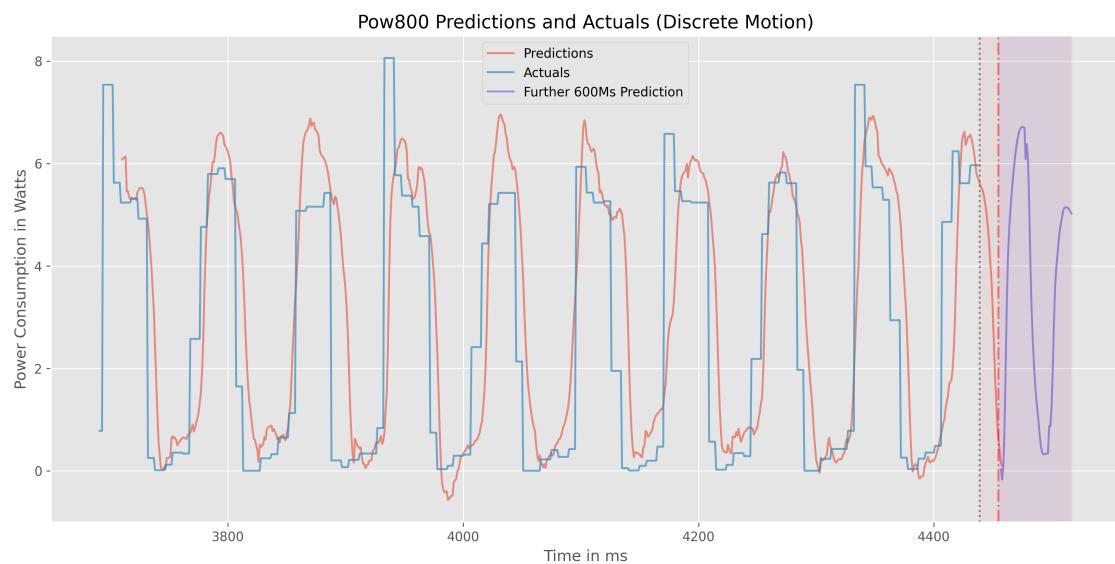
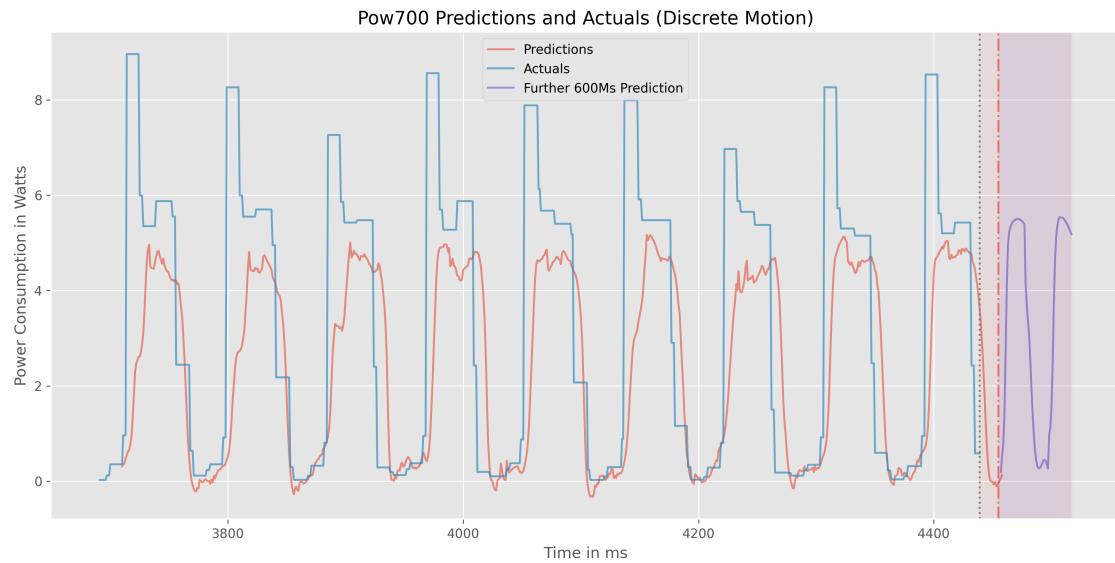


Pow500 Predictions and Actuals (Discrete Motion)

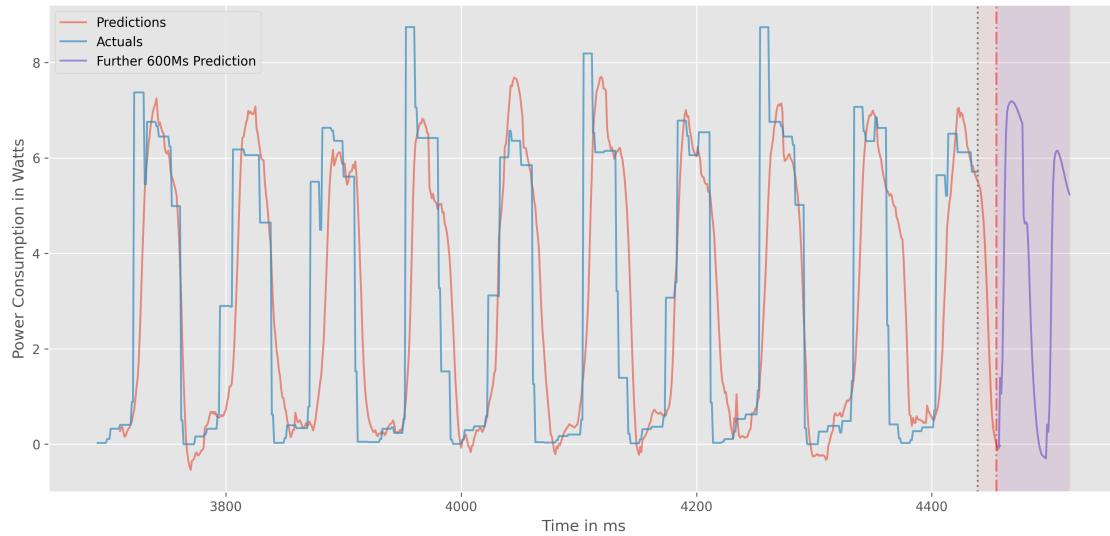


Pow600 Predictions and Actuals (Discrete Motion)





Pow900 Predictions and Actuals (Discrete Motion)



Pow1000 Predictions and Actuals (Discrete Motion)

