FINAL_lstm_discrete_DAiA

February 2, 2023

1 Model Building

1.1 Imports

[ ]:
```
# 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 Resulution

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

Read In the different Datasets

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

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

[ ]:
```
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)
```

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

[ ]:

	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

1.2 Defining Functions

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

[ ]:
```
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

[ ]:
```
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)**

[ ]:
```
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

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

Scale the Data betweet 0 and 1 using the MinMaxScaler

[ ]:
```
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**)

[ ]:
```
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

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

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

1.4 Build LSTM Model

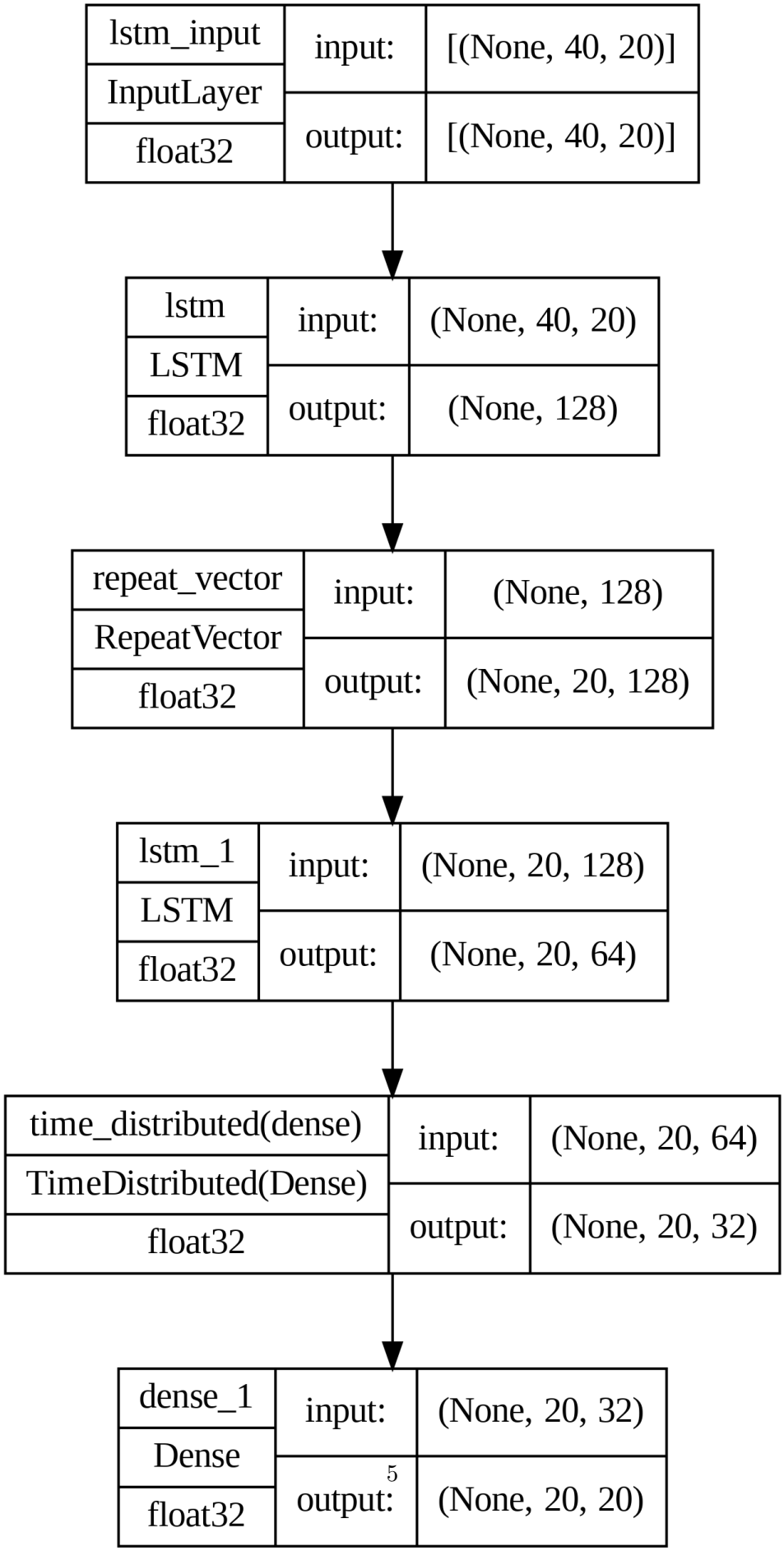
1.4.1 Model Summary

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 128)               76288

 repeat_vector (RepeatVector  (None, 20, 128)          0
 )

 lstm_1 (LSTM)               (None, 20, 64)            49408

 time_distributed (TimeDistr  (None, 20, 32)           2080
 ibuted)

 dense_1 (Dense)             (None, 20, 20)            660

=================================================================
Total params: 128,436
Trainable params: 128,436
Non-trainable params: 0
_________________________________________________________________
```

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

[ ]:


lstm_input
InputLayer
float32
input: [(None, 40, 20)]
output: [(None, 40, 20)]
lstm
LSTM
float32
input: (None, 40, 20)
output: (None, 128)
repeat_vector
RepeatVector
float32
input: (None, 128)
output: (None, 20, 128)
lstm_1
LSTM
float32
input: (None, 20, 128)
output: (None, 20, 64)
time_distributed(dense)
TimeDistributed(Dense)
float32
input: (None, 20, 64)
output: (None, 20, 32)
dense_1
Dense
float32
input: (None, 20, 32)
output: (None, 20, 20)

1.4.2 Compile the LSTM Model

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

Create Callback and Fit the Training Dataset

[ ]:
```
stop_early_lstm = 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_lstm])
```

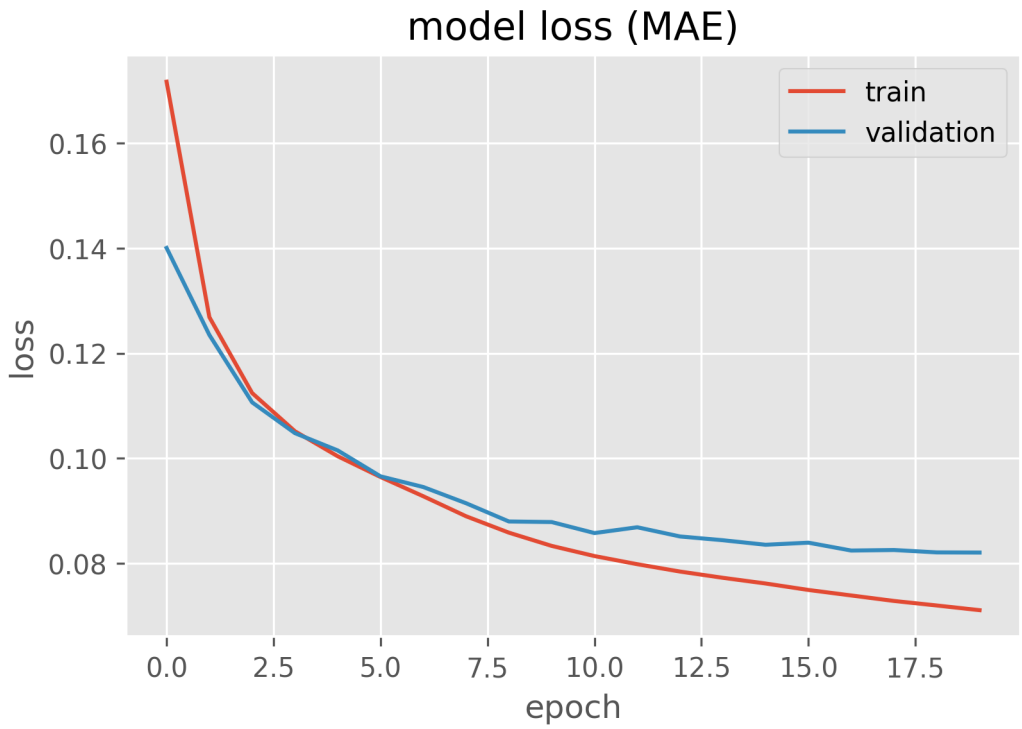
```
Epoch 1/20
1124/1124 [==============================] - 83s 71ms/step - loss: 0.1717 - mse:
0.0631 - val_loss: 0.1401 - val_mse: 0.0419
Epoch 2/20
1124/1124 [==============================] - 81s 72ms/step - loss: 0.1269 - mse:
0.0357 - val_loss: 0.1235 - val_mse: 0.0322
Epoch 3/20
1124/1124 [==============================] - 79s 70ms/step - loss: 0.1124 - mse:
0.0302 - val_loss: 0.1107 - val_mse: 0.0285
Epoch 4/20
1124/1124 [==============================] - 79s 71ms/step - loss: 0.1051 - mse:
0.0282 - val_loss: 0.1048 - val_mse: 0.0273
Epoch 5/20
1124/1124 [==============================] - 81s 72ms/step - loss: 0.1004 - mse:
0.0271 - val_loss: 0.1015 - val_mse: 0.0265
Epoch 6/20
1124/1124 [==============================] - 78s 70ms/step - loss: 0.0964 - mse:
0.0259 - val_loss: 0.0966 - val_mse: 0.0250
Epoch 7/20
1124/1124 [==============================] - 78s 69ms/step - loss: 0.0928 - mse:
0.0245 - val_loss: 0.0945 - val_mse: 0.0244
Epoch 8/20
1124/1124 [==============================] - 79s 70ms/step - loss: 0.0890 - mse:
0.0232 - val_loss: 0.0914 - val_mse: 0.0237
Epoch 9/20
1124/1124 [==============================] - 78s 69ms/step - loss: 0.0858 - mse:
0.0223 - val_loss: 0.0880 - val_mse: 0.0228
Epoch 10/20
1124/1124 [==============================] - 78s 69ms/step - loss: 0.0833 - mse:
0.0216 - val_loss: 0.0879 - val_mse: 0.0233
Epoch 11/20
1124/1124 [==============================] - 78s 70ms/step - loss: 0.0814 - mse:
0.0210 - val_loss: 0.0858 - val_mse: 0.0227
Epoch 12/20
```

```
1124/1124 [==============================] - 77s 69ms/step - loss: 0.0798 - mse:
0.0205 - val_loss: 0.0869 - val_mse: 0.0233
Epoch 13/20
1124/1124 [==============================] - 78s 70ms/step - loss: 0.0784 - mse:
0.0200 - val_loss: 0.0851 - val_mse: 0.0226
Epoch 14/20
1124/1124 [==============================] - 79s 71ms/step - loss: 0.0772 - mse:
0.0196 - val_loss: 0.0844 - val_mse: 0.0223
Epoch 15/20
1124/1124 [==============================] - 78s 70ms/step - loss: 0.0762 - mse:
0.0192 - val_loss: 0.0835 - val_mse: 0.0223
Epoch 16/20
1124/1124 [==============================] - 78s 69ms/step - loss: 0.0749 - mse:
0.0188 - val_loss: 0.0839 - val_mse: 0.0224
Epoch 17/20
1124/1124 [==============================] - 79s 71ms/step - loss: 0.0739 - mse:
0.0184 - val_loss: 0.0824 - val_mse: 0.0223
Epoch 18/20
1124/1124 [==============================] - 78s 70ms/step - loss: 0.0728 - mse:
0.0181 - val_loss: 0.0825 - val_mse: 0.0227
Epoch 19/20
1124/1124 [==============================] - 78s 70ms/step - loss: 0.0720 - mse:
0.0177 - val_loss: 0.0821 - val_mse: 0.0224
Epoch 20/20
1124/1124 [==============================] - 79s 70ms/step - loss: 0.0711 - mse:
0.0174 - val_loss: 0.0820 - val_mse: 0.0223
```

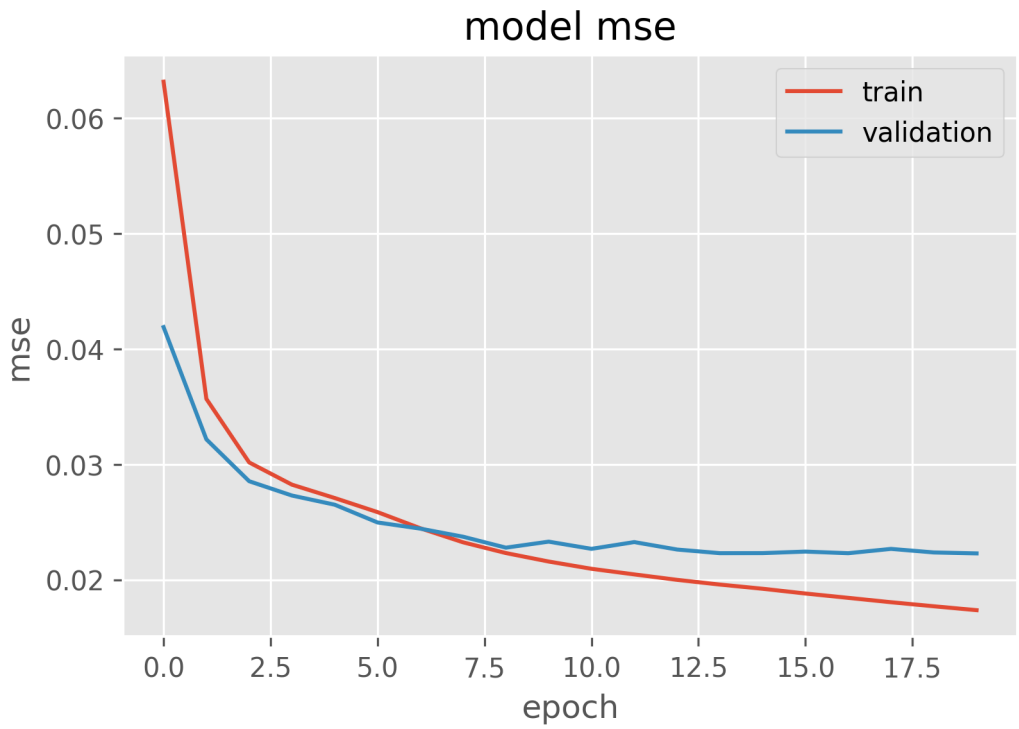
1.4.3 Model History

Summarize and plot the history of the loss and val_loss

[ ]:
```
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()
```



[ ]:
```
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

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

```
139/139 [==============================] - 3s 23ms/step - loss: 0.0876 - mse:
0.0249
```

[ ]:
```
[0.08759897947311401, 0.024937016889452934]
```

1.4.4 Predicting LSTM

Predict LSTM Model

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

```
139/139 [==============================] - 4s 23ms/step
```

Saving the shape of y_test

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

Print RMSE Plots from t+1 to t+20

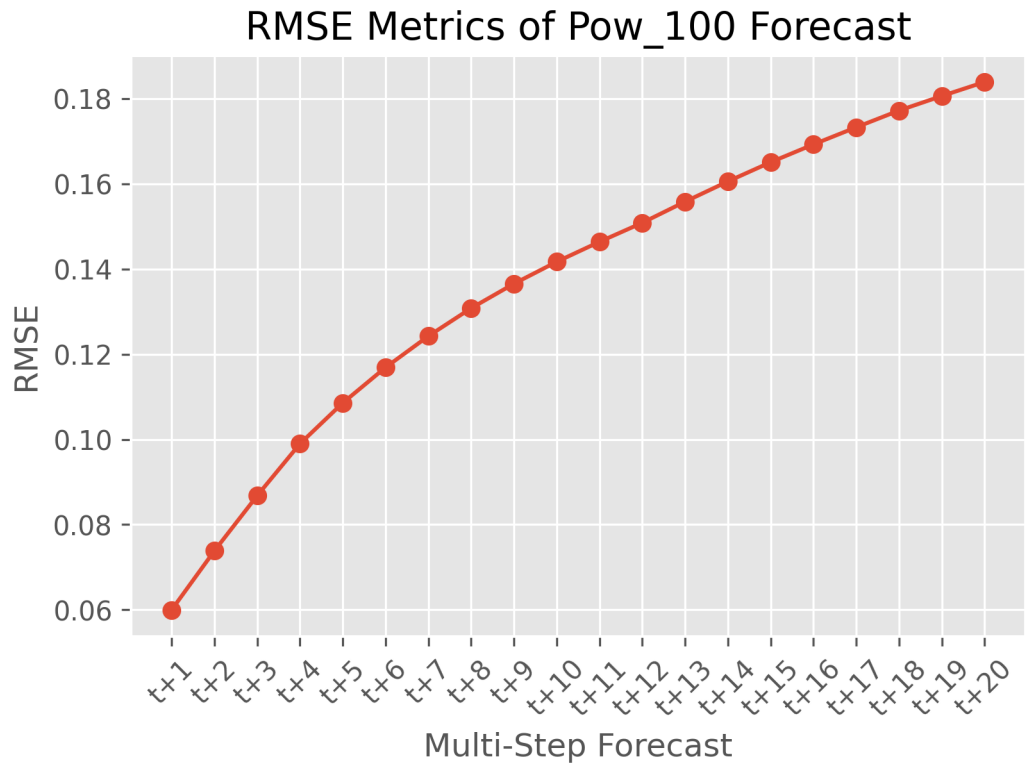
[ ]:
```
from math import sqrt
# evaluate forecasts against expected values
```

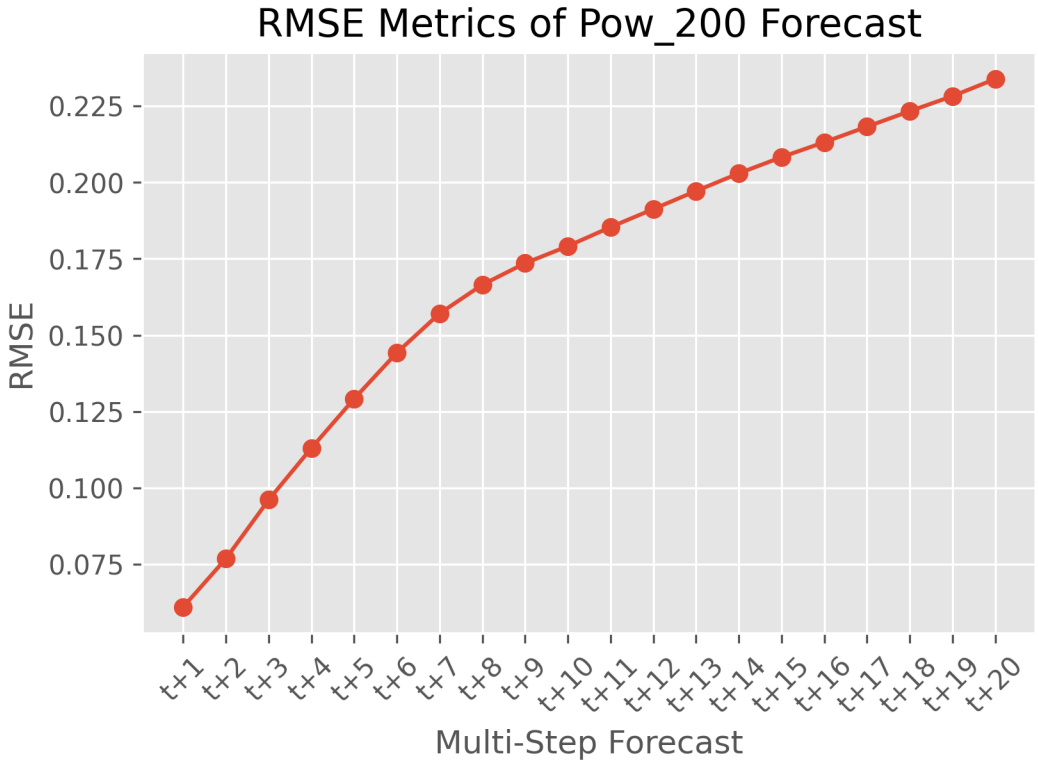
```
def evaluate_forecasts(actual, predicted):
    scores_pow = results = np.zeros((20, 20))
    scores = []
 # calculate an RMSE score for each prediction period
    for i in range(actual.shape[1]):
      for t in range(actual.shape[2]):
       # calculate mse
       scores_pow[i, t] = sqrt(mean_squared_error(actual[:, 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)
```

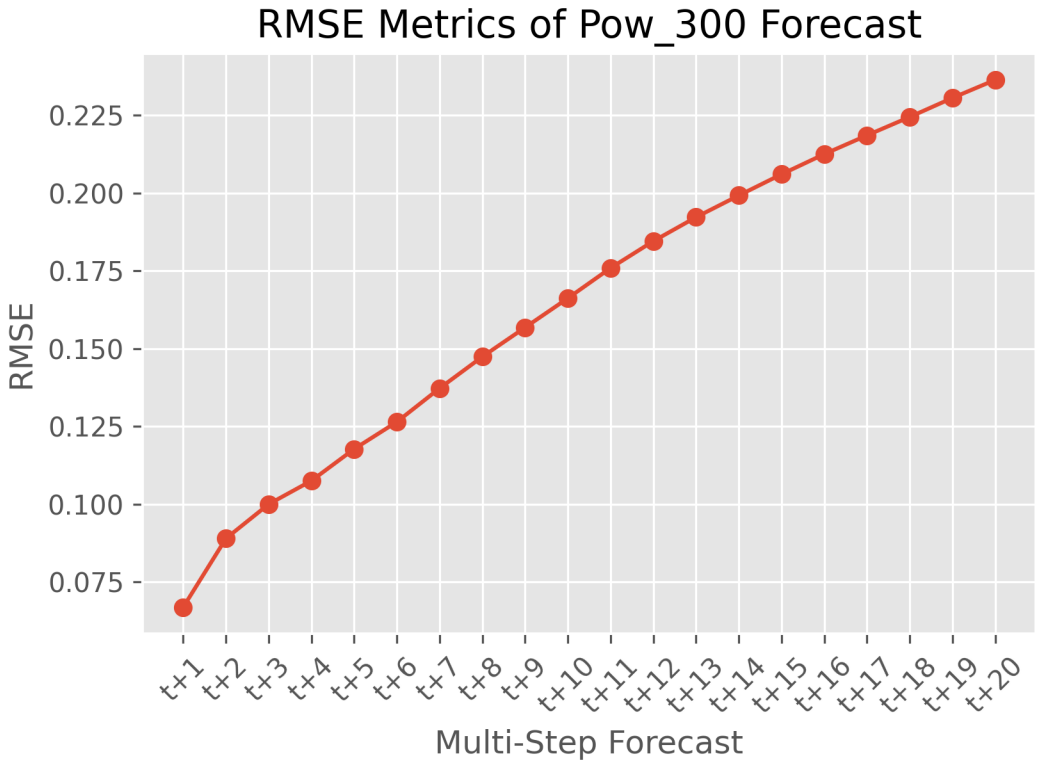
[ ]:
```
def summarize_scores(name, score, scores):
    s_scores = ', '.join(['%.3f'% s for s in scores])
    print('%s: [%.3f] %s' % (name, score, s_scores))
```

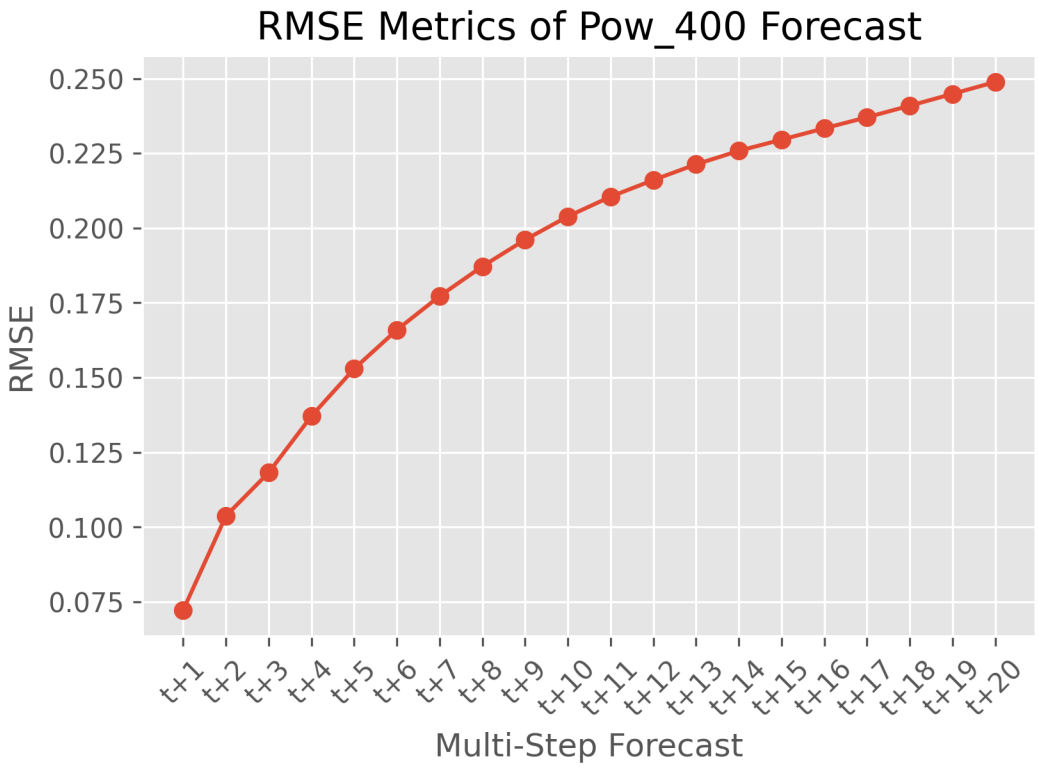
[ ]:
```
overall_RMSE, all_RMSEs = evaluate_forecasts(y_test, yhat)
```

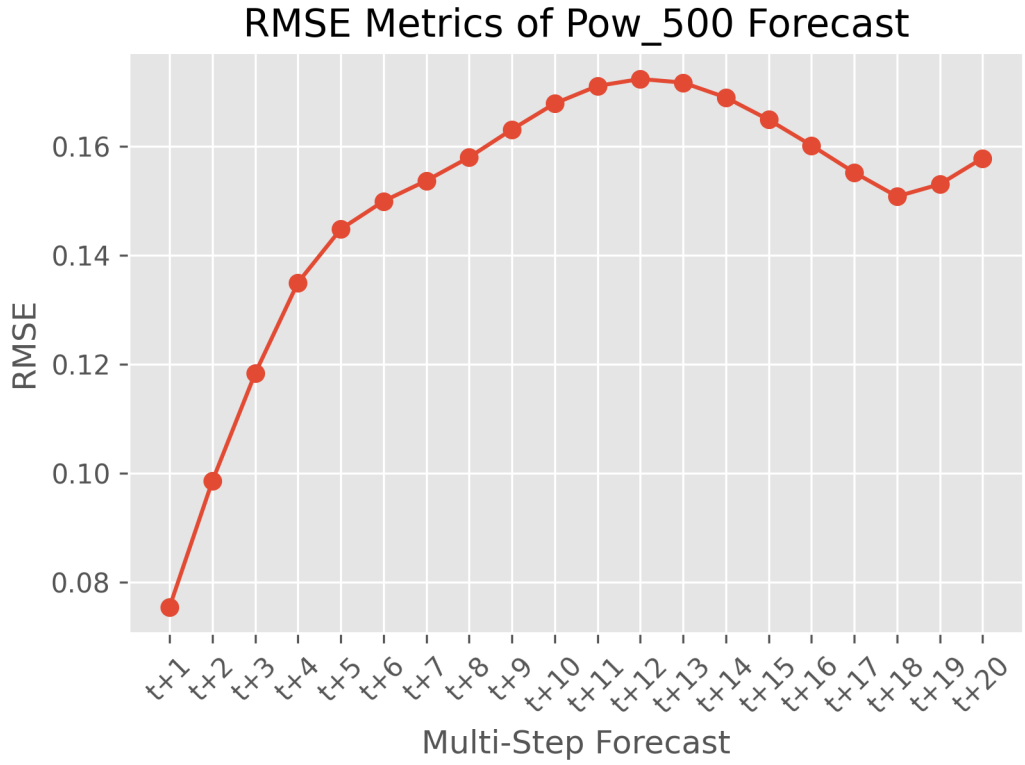
[ ]:
```
forecast = []
for i in range(1,21):
    title = f't+{i}'
    forecast.append(title)
for i in range(20):
  plt.style.use("ggplot")
  plt.plot(forecast, all_RMSEs[i], marker='o', label='lstm')
  plt.xticks(rotation=45)
  plt.title('RMSE Metrics of Pow_' +str(100*(i+1))+ ' Forecast')
  plt.xlabel('Multi-Step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

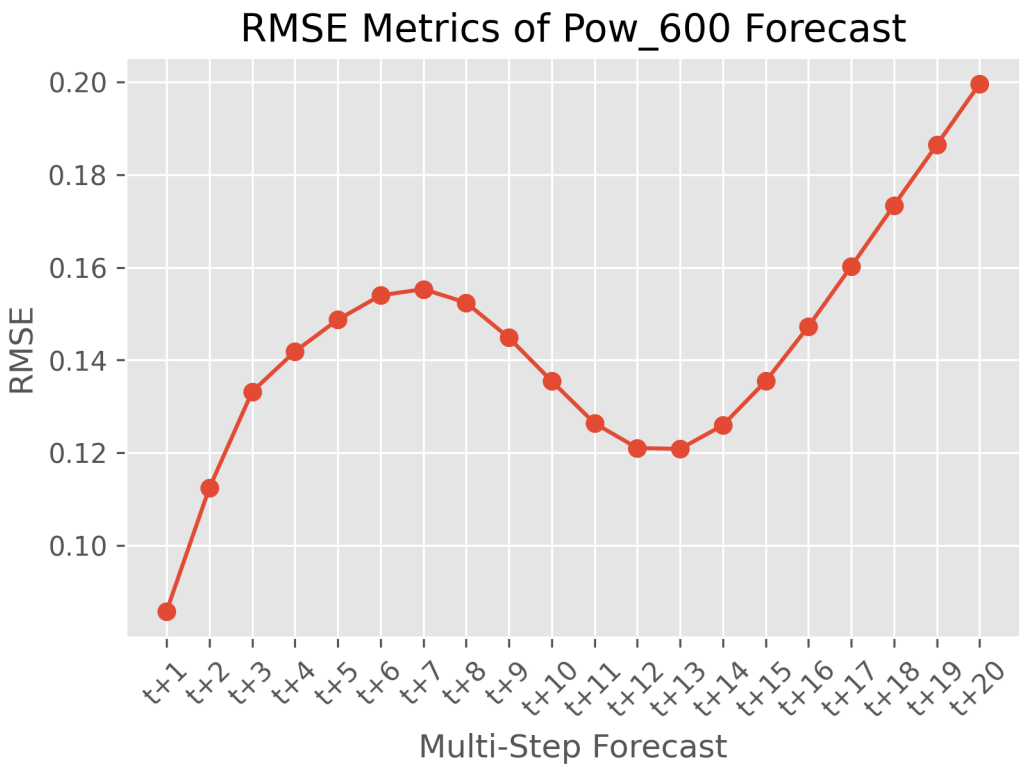

RMSE Metrics of Pow_100 Forecast
RMSE
0.18
0.16
0.14
0.12
0.10
0.08
0.06
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

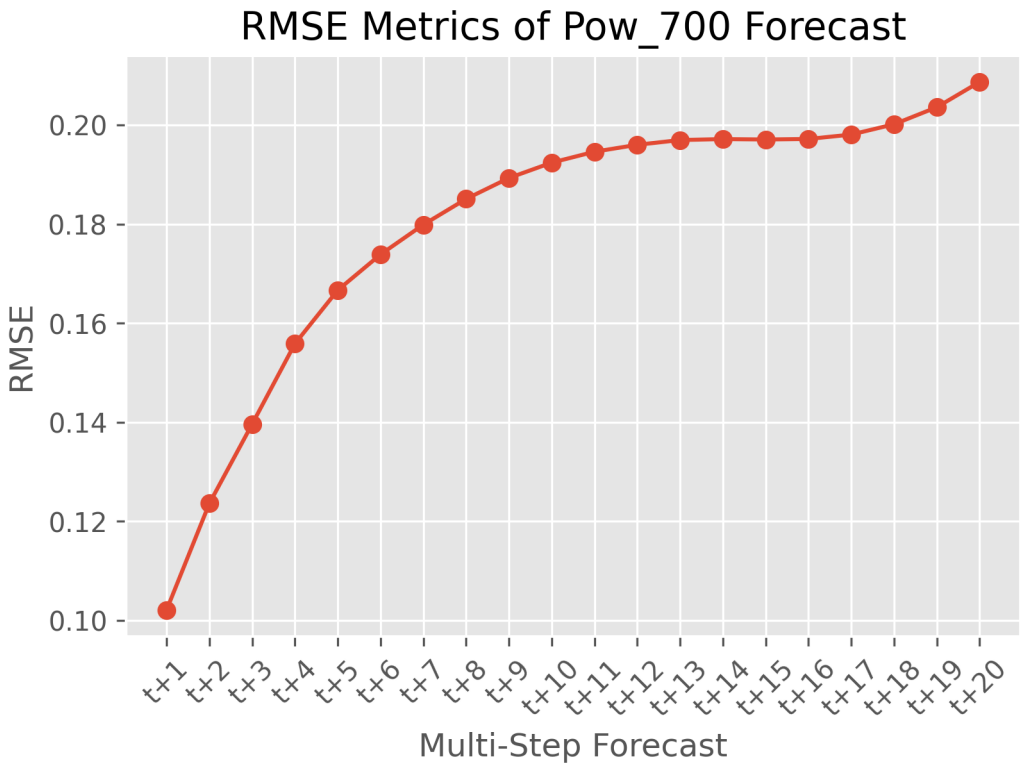

RMSE Metrics of Pow_200 Forecast
RMSE
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

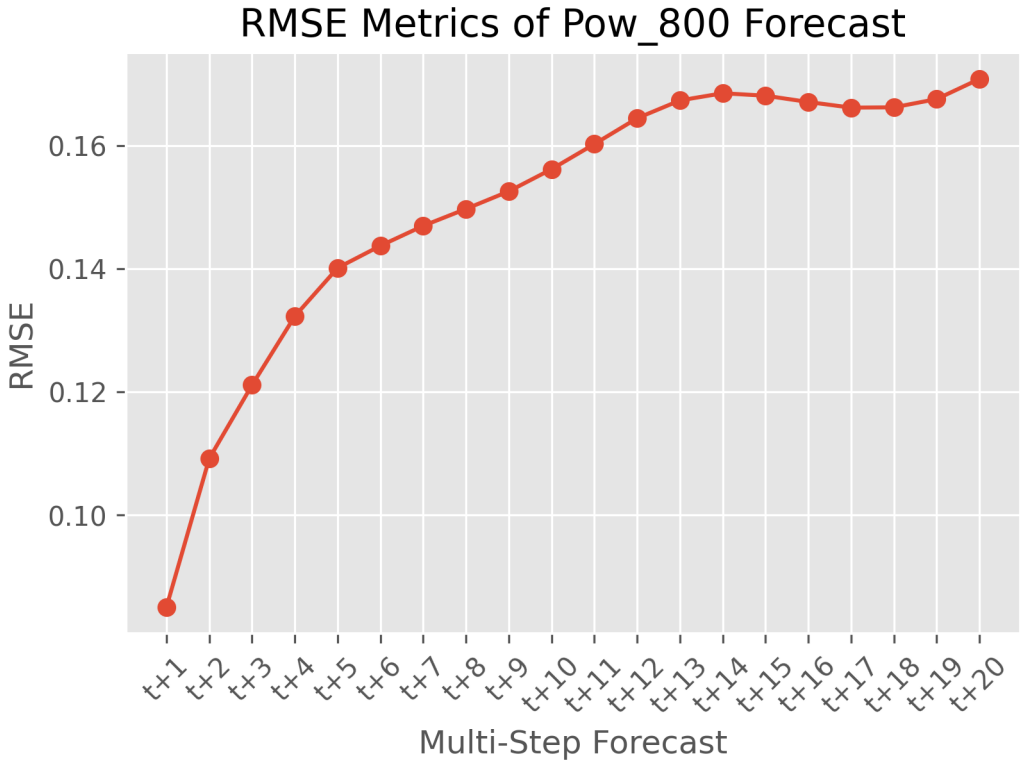

RMSE Metrics of Pow_300 Forecast
RMSE
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

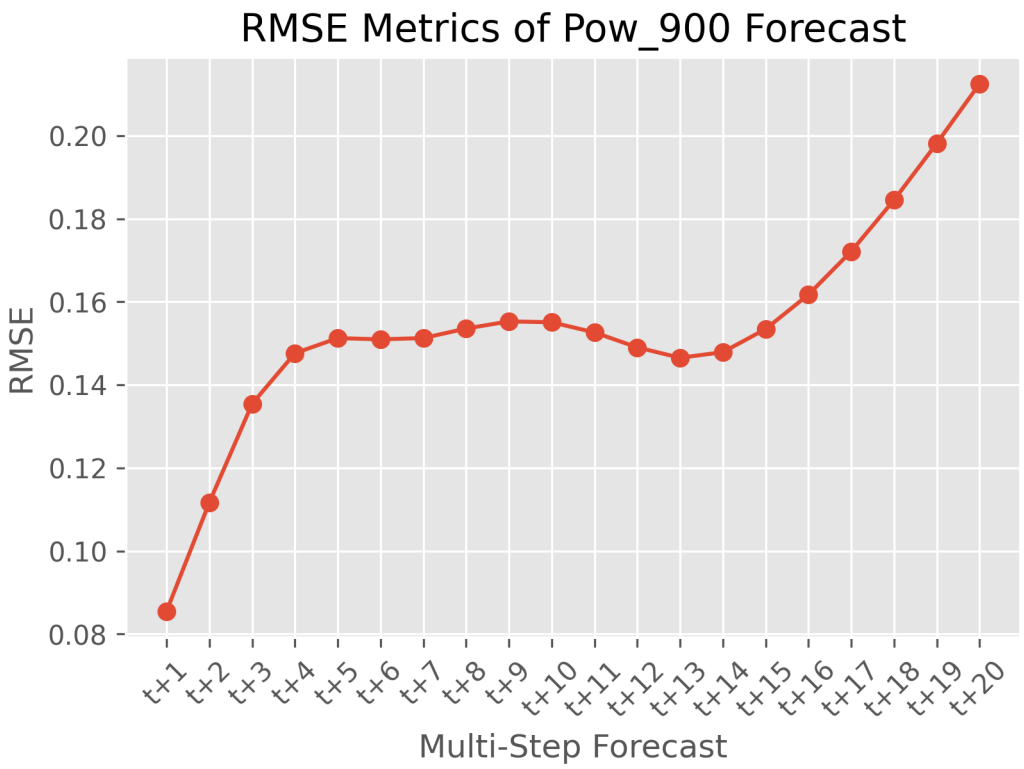

RMSE Metrics of Pow_400 Forecast
RMSE
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

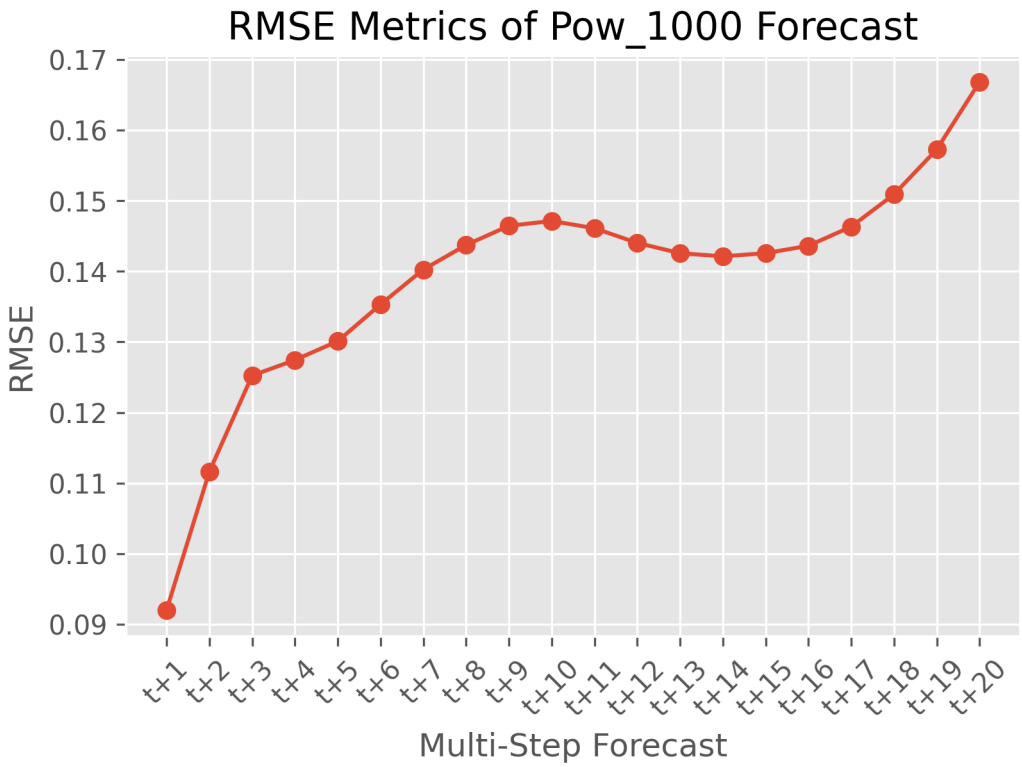

RMSE Metrics of Pow_500 Forecast
RMSE
0.16
0.14
0.12
0.10
0.08
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

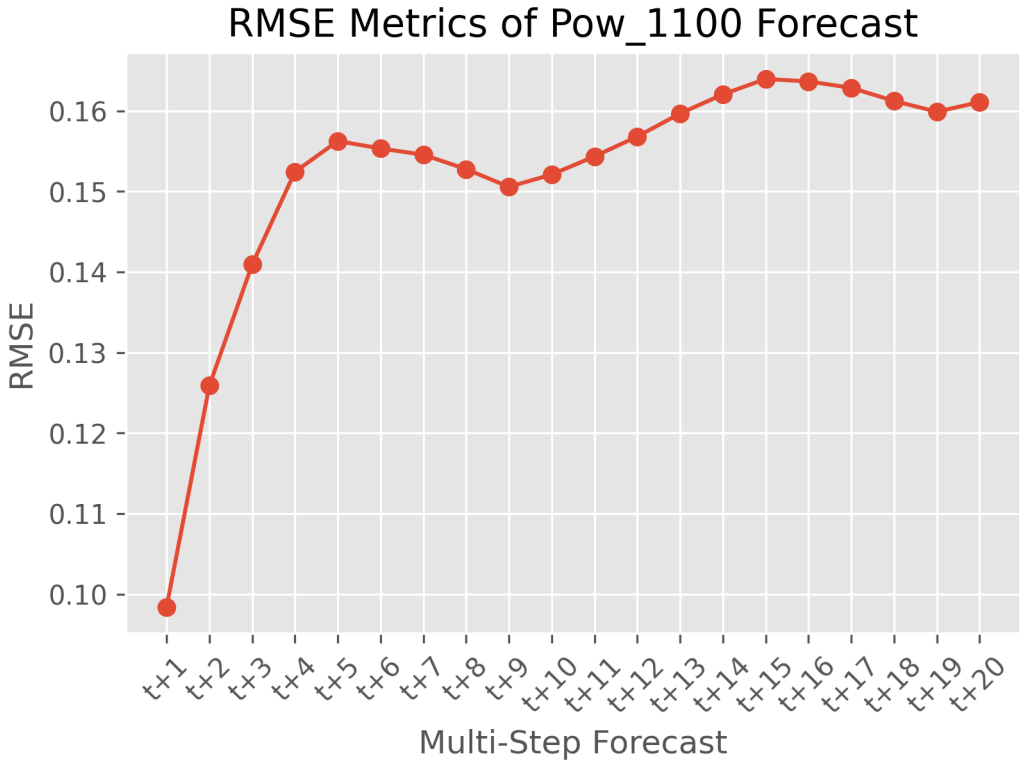

RMSE Metrics of Pow_600 Forecast
RMSE
0.20
0.18
0.16
0.14
0.12
0.10
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

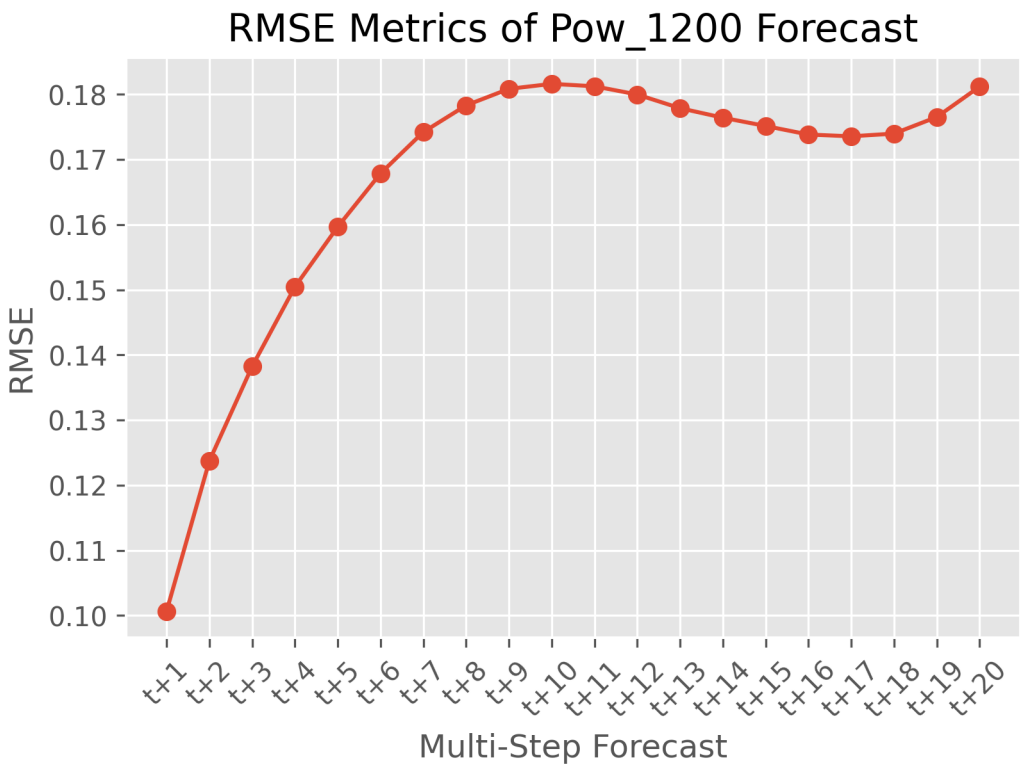

RMSE Metrics of Pow_700 Forecast
RMSE
0.20
0.18
0.16
0.14
0.12
0.10
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

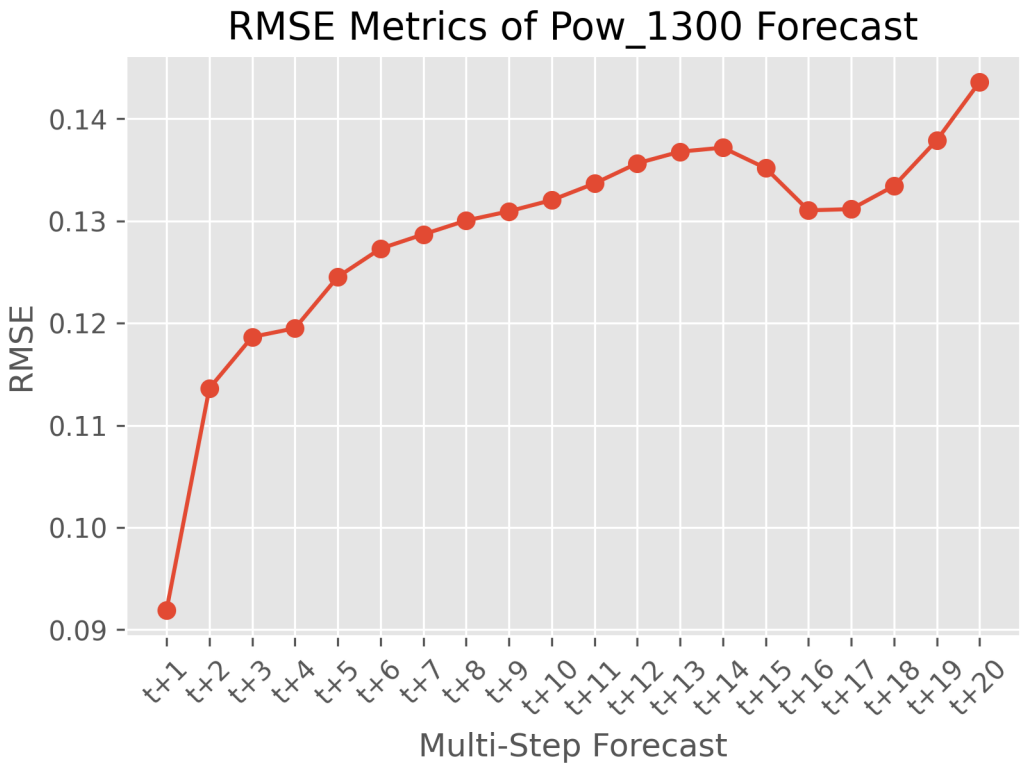

RMSE Metrics of Pow_800 Forecast
RMSE
0.16
0.14
0.12
0.10
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

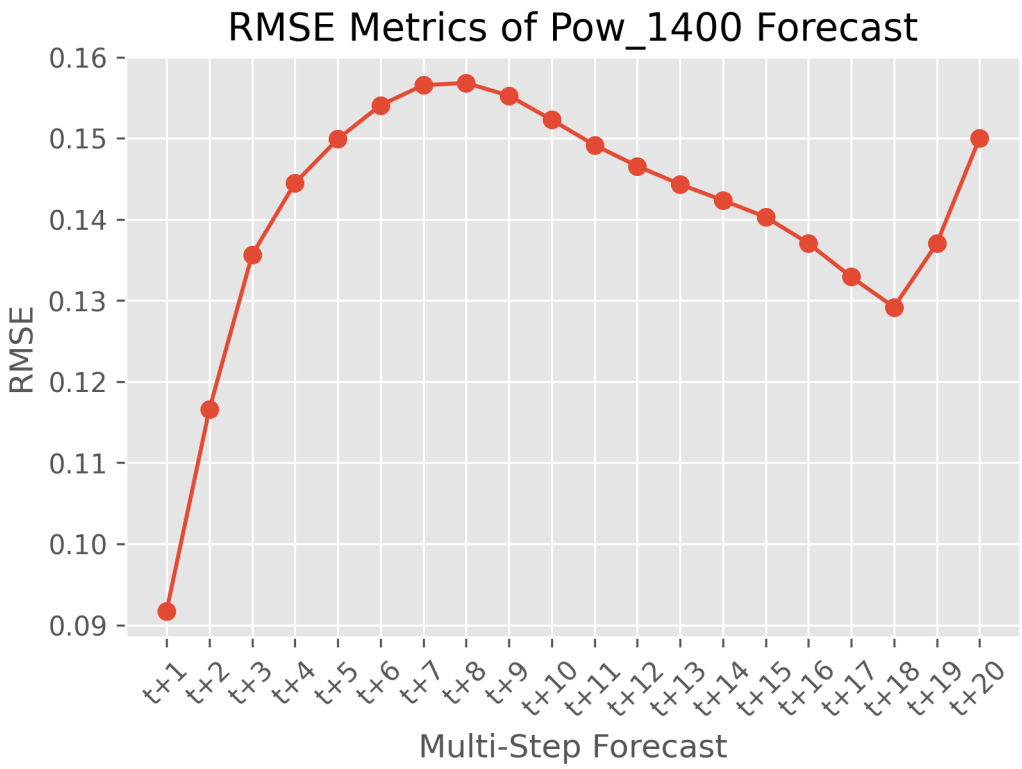

RMSE Metrics of Pow_900 Forecast
RMSE
0.20
0.18
0.16
0.14
0.12
0.10
0.08
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

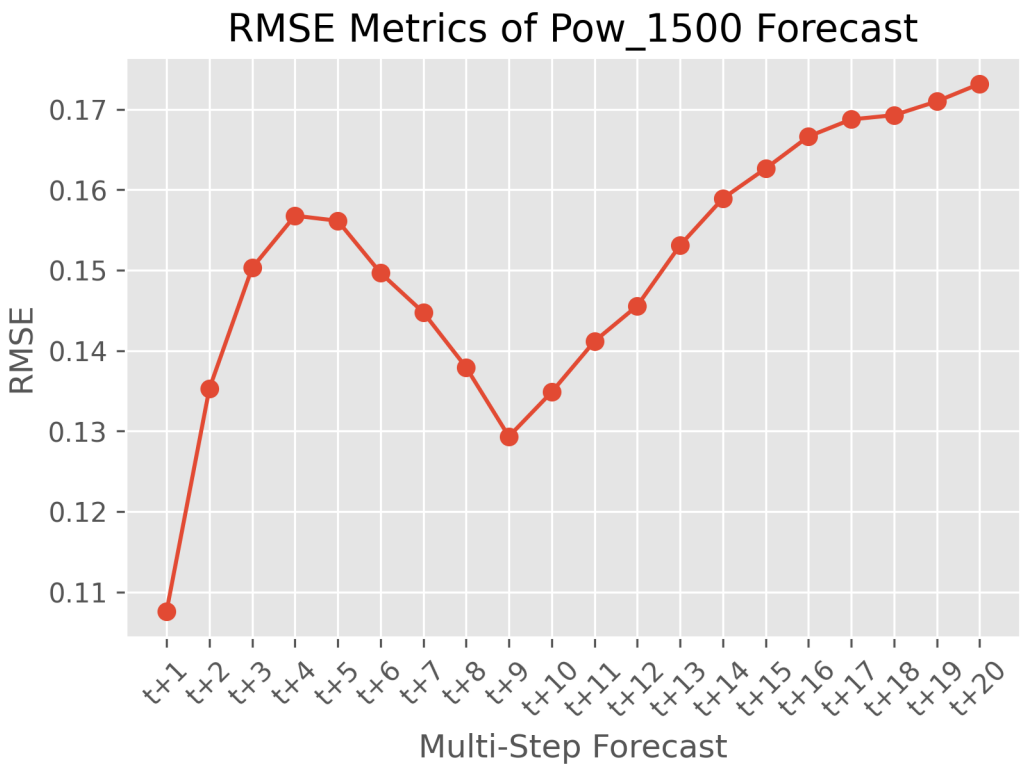

RMSE Metrics of Pow_1000 Forecast
RMSE
0.17
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.09
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

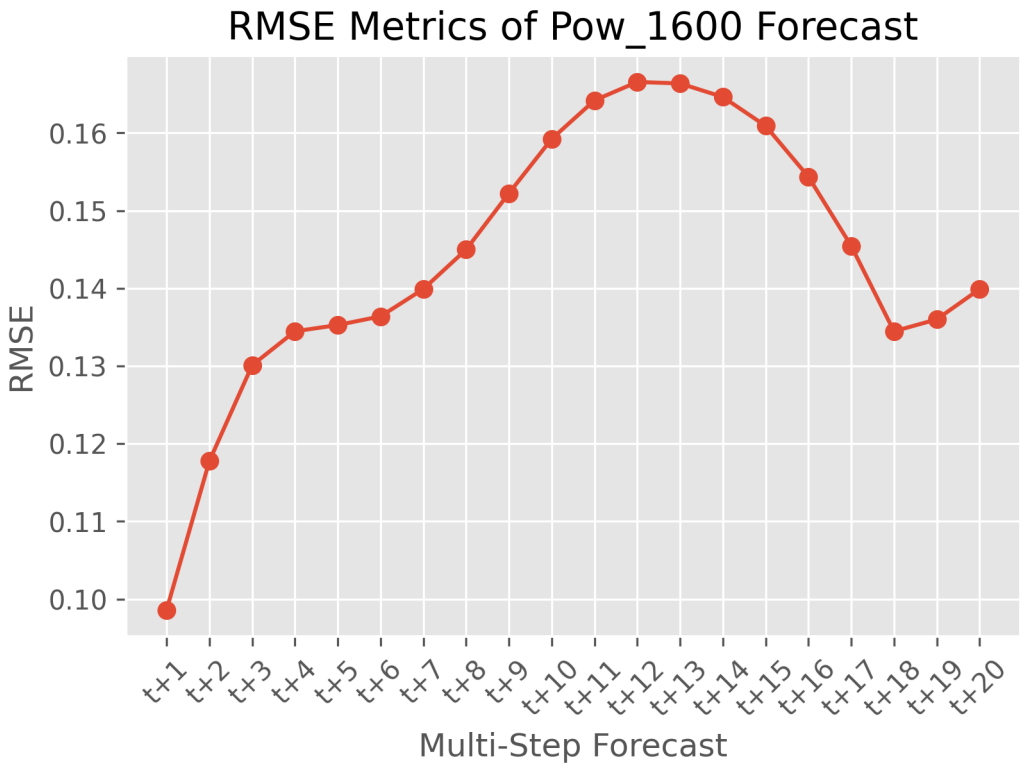

RMSE Metrics of Pow_1100 Forecast
RMSE
0.16
0.15
0.14
0.13
0.12
0.11
0.10
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

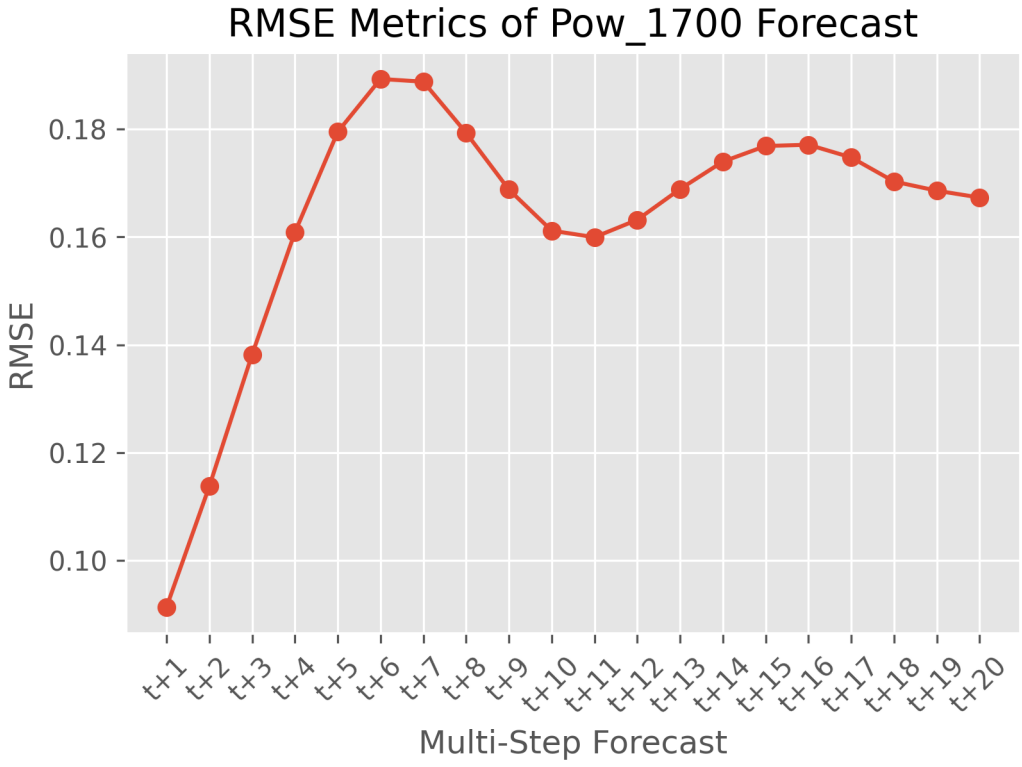

RMSE Metrics of Pow_1200 Forecast
RMSE
0.18
0.17
0.16
0.15
0.14
0.13
0.12
0.11
0.10
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

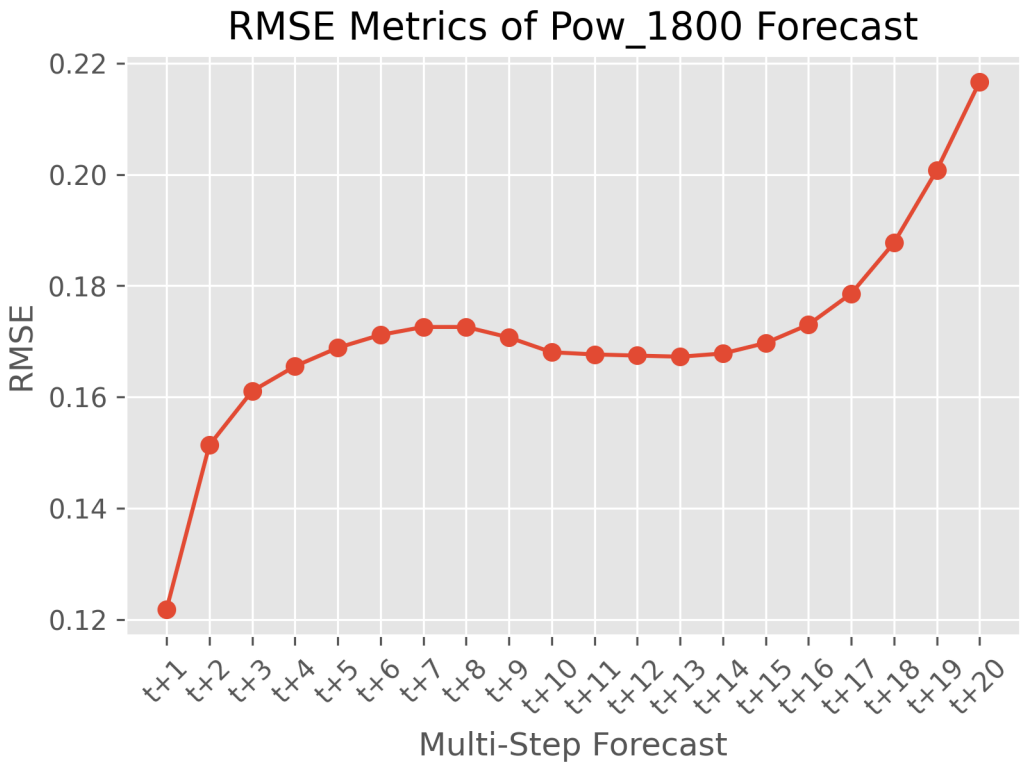

RMSE Metrics of Pow_1300 Forecast
RMSE
0.14
0.13
0.12
0.11
0.10
0.09
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

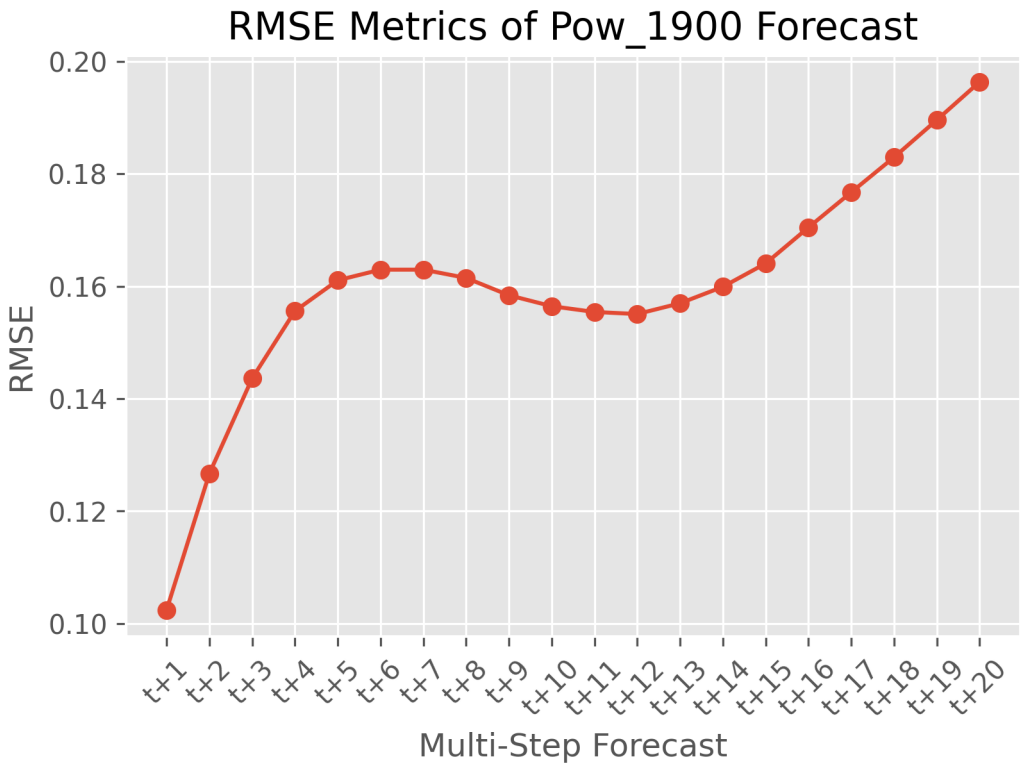

RMSE Metrics of Pow_1400 Forecast
RMSE
0.16
0.15
0.14
0.13
0.12
0.11
0.10
0.09
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

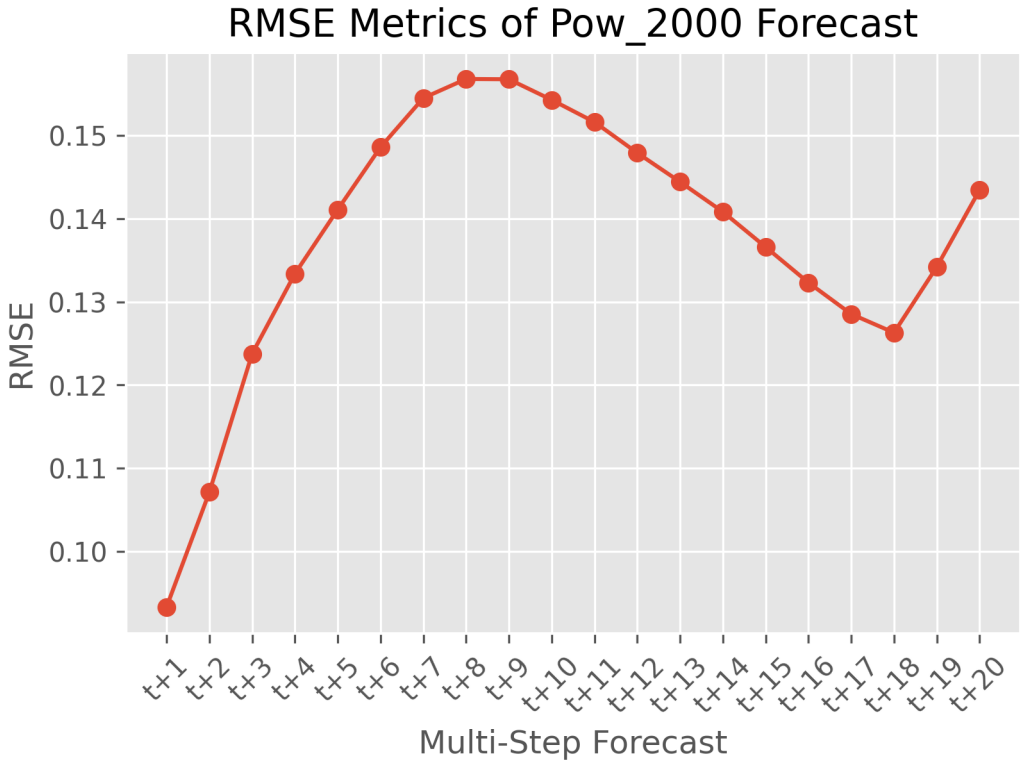

RMSE Metrics of Pow_1500 Forecast
RMSE
0.17
0.16
0.15
0.14
0.13
0.12
0.11
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast


RMSE Metrics of Pow_1600 Forecast
RMSE
0.16
0.15
0.14
0.13
0.12
0.11
0.10
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast


RMSE Metrics of Pow_1700 Forecast
RMSE
0.18
0.16
0.14
0.12
0.10
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast


RMSE Metrics of Pow_1800 Forecast
RMSE
0.22
0.20
0.18
0.16
0.14
0.12
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast


RMSE Metrics of Pow_1900 Forecast
RMSE
0.20
0.18
0.16
0.14
0.12
0.10
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast


RMSE Metrics of Pow_2000 Forecast
RMSE
0.15
0.14
0.13
0.12
0.11
0.10
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

Backing up the data

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

New Unseen Predictions

[ ]:
```
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 573ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
```

Rescale

[ ]:
```
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
```

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

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

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

[ ]:
```
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)
```

[ ]:
```
# 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)
```

```
[ ]: 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)

```
[ ]: 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

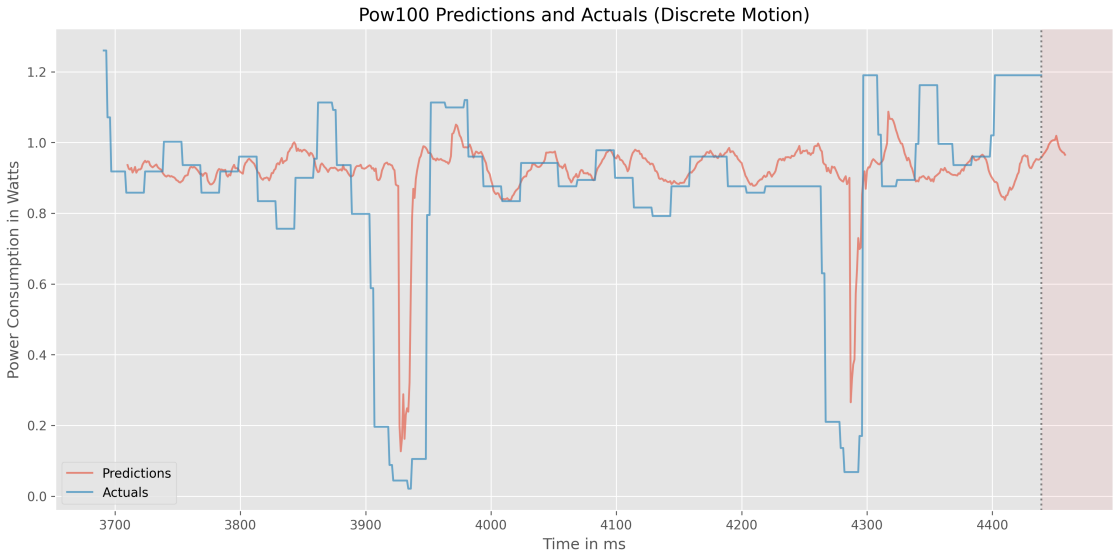
```
[ ]: from sklearn.metrics import r2_score
     def print_act_and_pred_tables(yhat,ytest,forecast_horizon,␣
      ↪start_graph,end_graph):
       pow_preds = []
       pow_actuals = []
       for i in range(20):
         pow_preds.append(yhat[:, forecast_horizon-1, i]) # 1 refers to the forecast␣
      ↪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
```

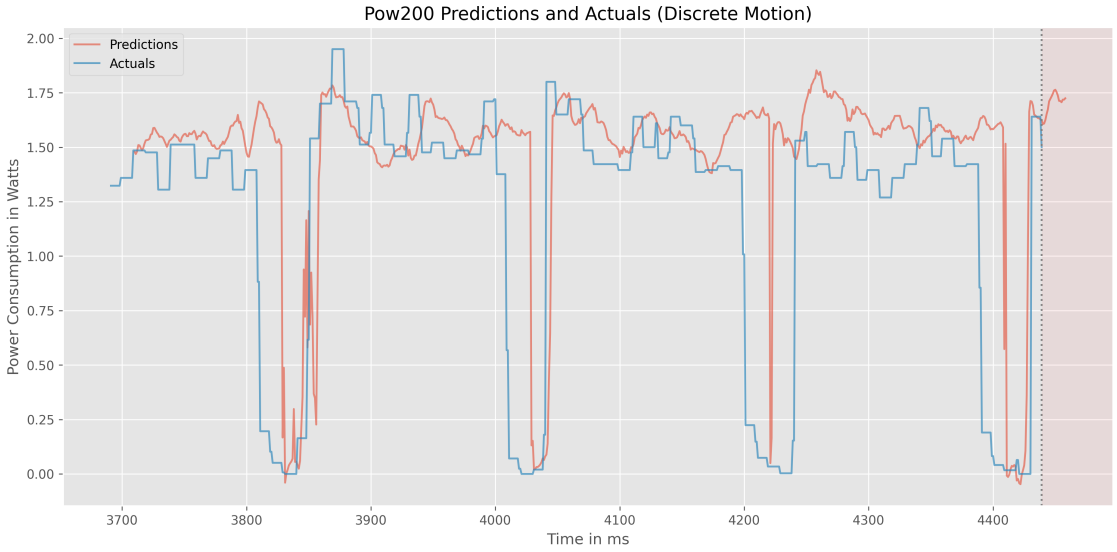
```
    actuals = df_new[f"Pow{100*(i+1)} Actuals"][start_graph:end_graph]
    predictions = df_new[f"Pow{100*(i+1)} Predictions"][start_graph:end_graph]
    # shift the t+20 prediction 20 (relating to the wished forecast horizon) to␣
↪the right
    p_temp = predictions.to_frame()
    p_temp['time_index_shift'] = 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()
```

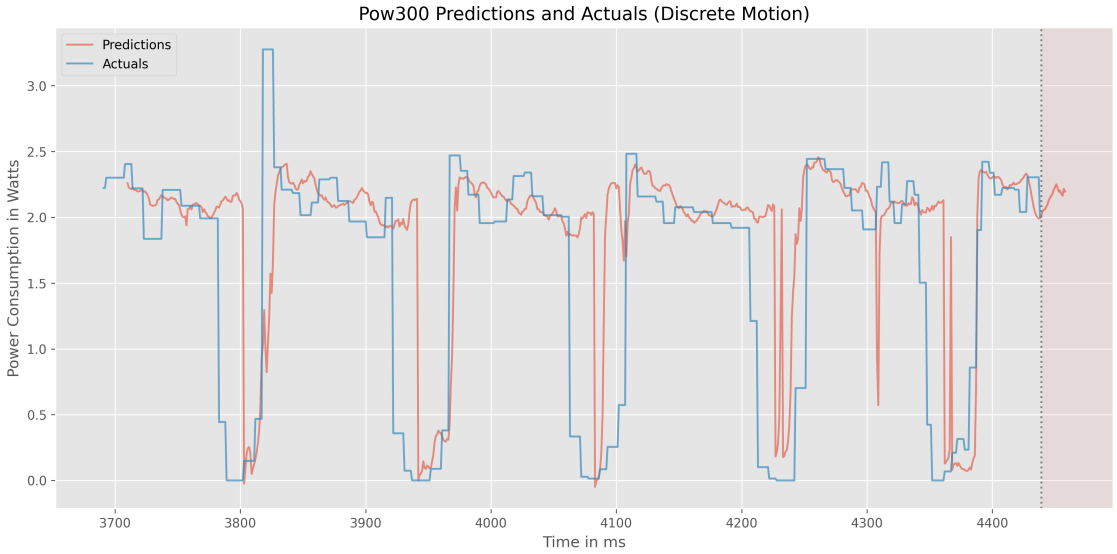
2.2 Print Actuals and Predictions

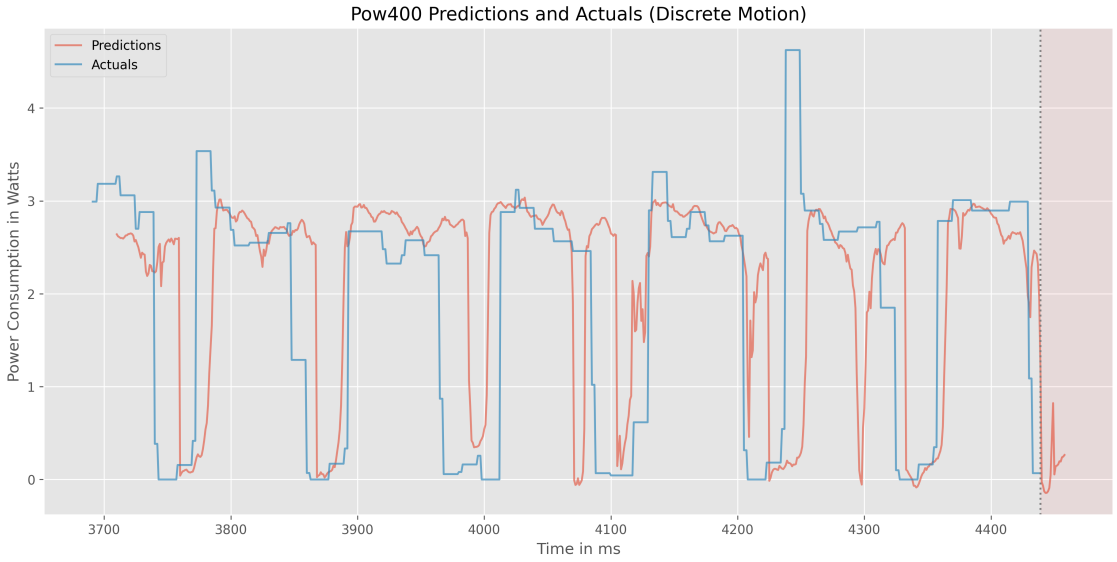
2.2.1 With Forecast Horizon of t+20 [Range: last 750 Timestamps]

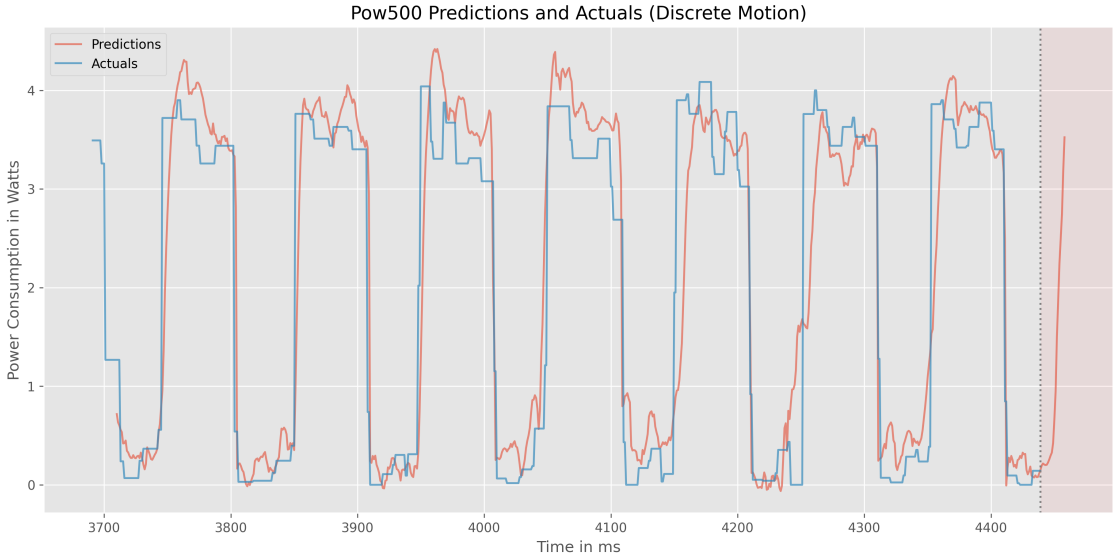
[ ]:
```
print_act_and_pred_tables(re_yhat2,re_y_test2,20, -750,-1)
```

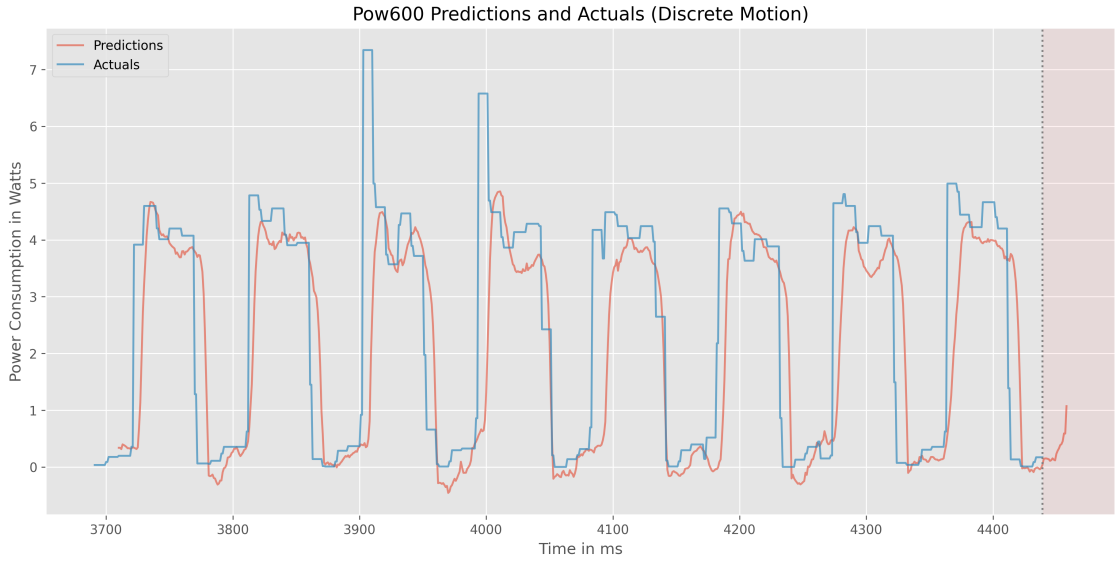


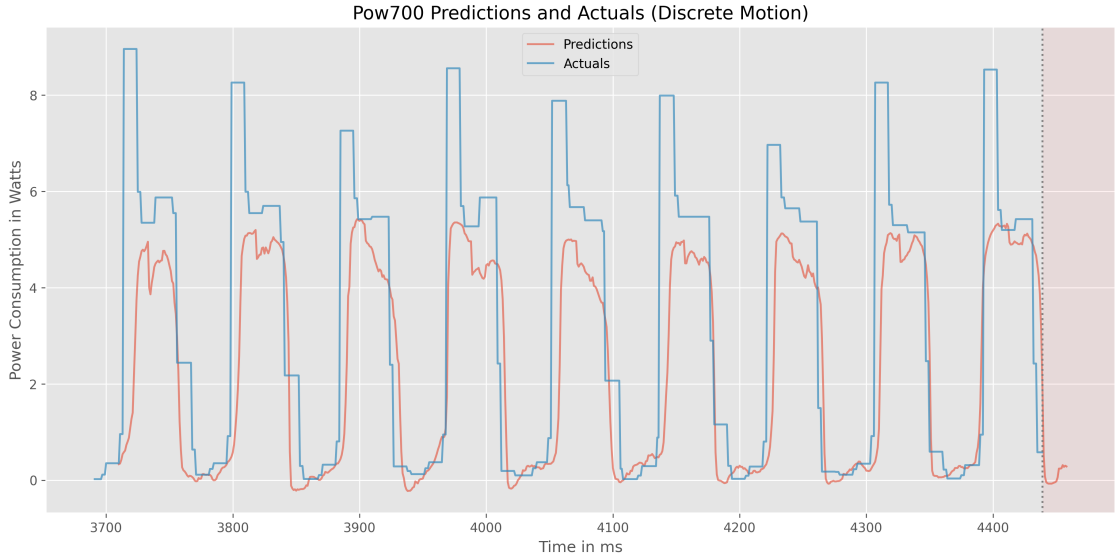

Pow200 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

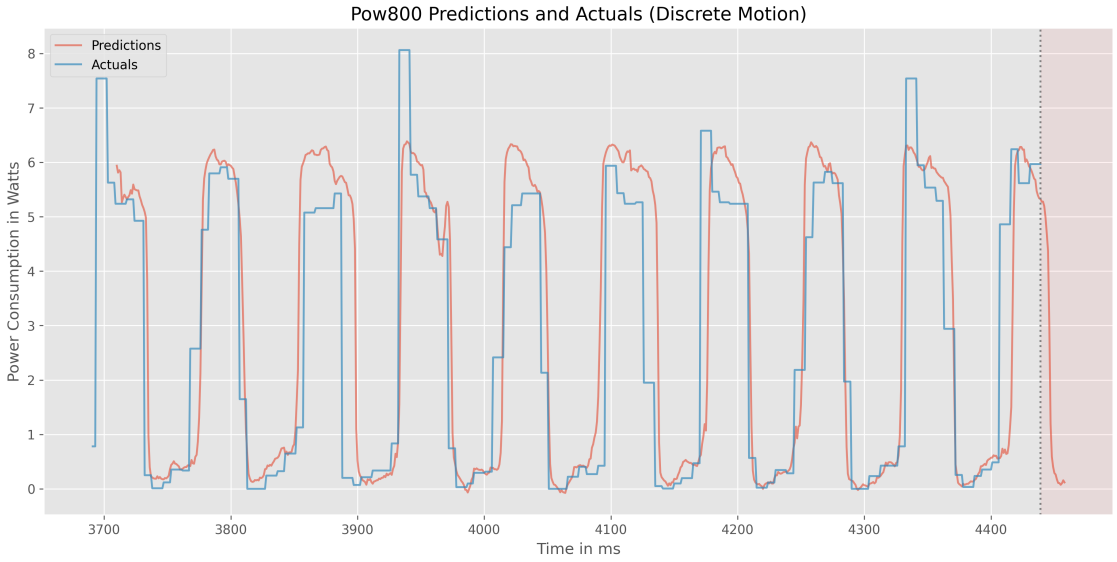

Pow300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

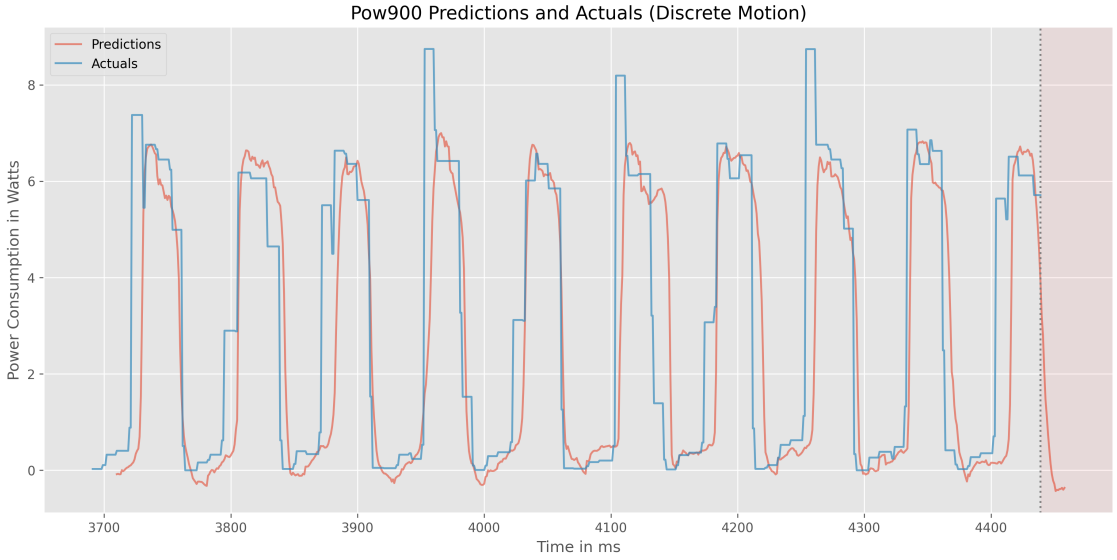

Pow400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
1
2
3
4
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

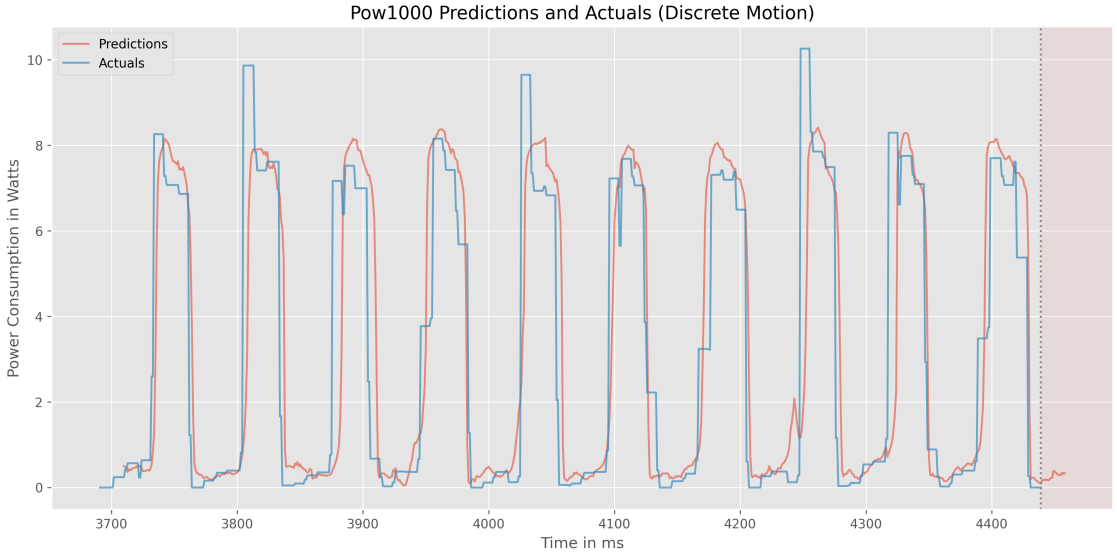

Pow500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
1
2
3
4
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

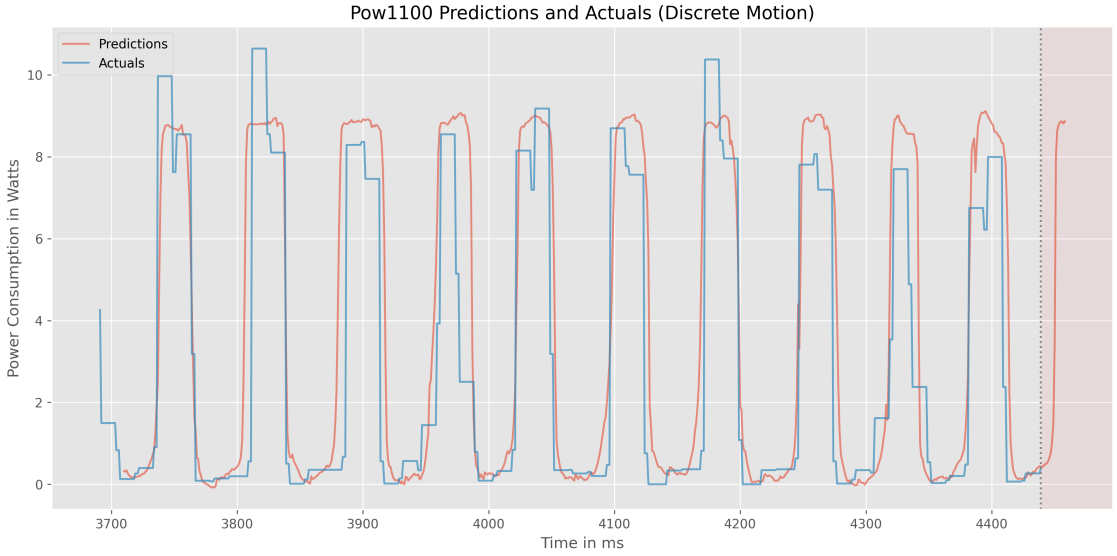

Pow600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

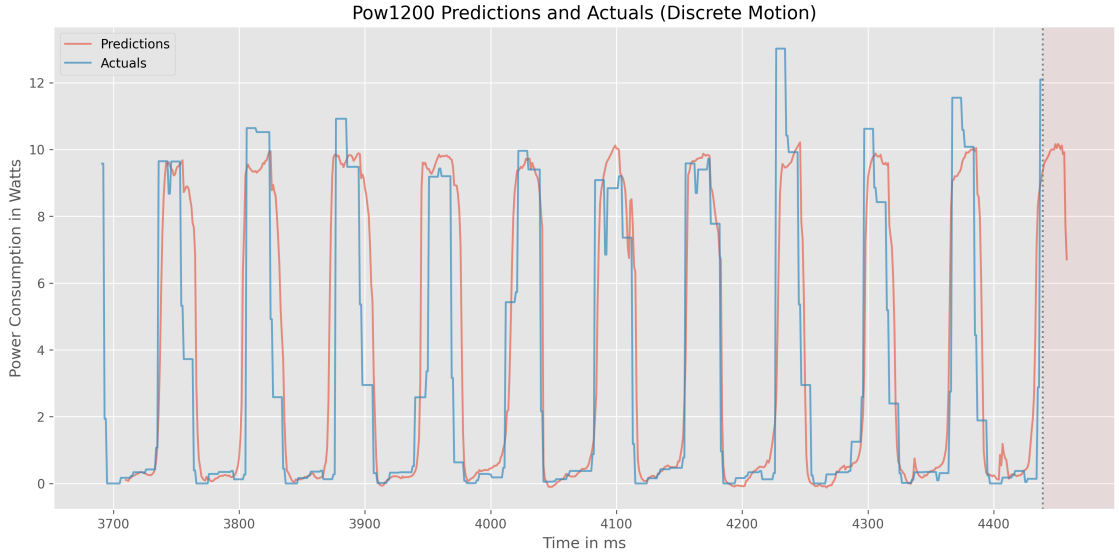

Pow700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

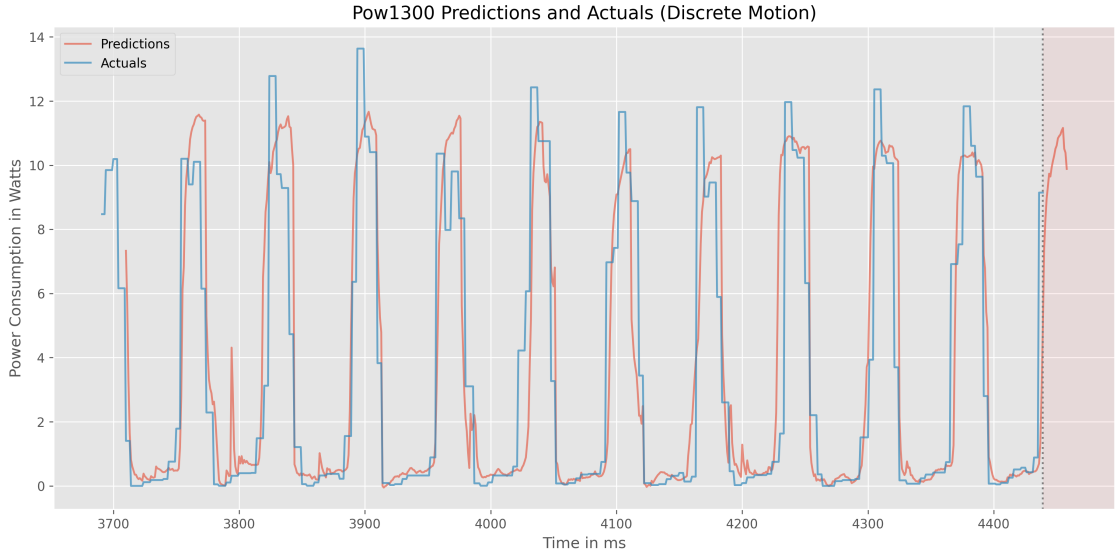

Pow800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

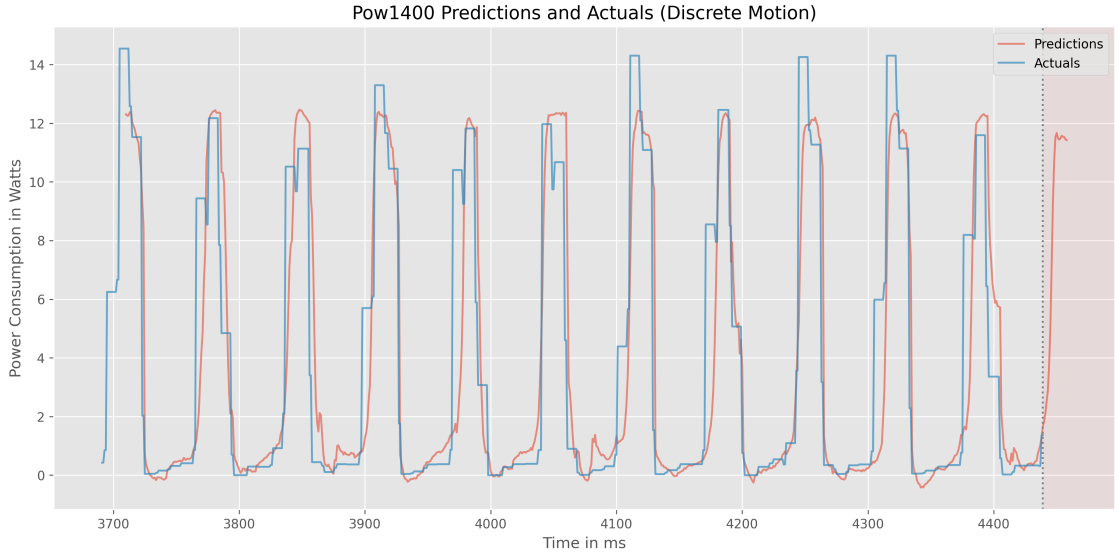

Pow900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

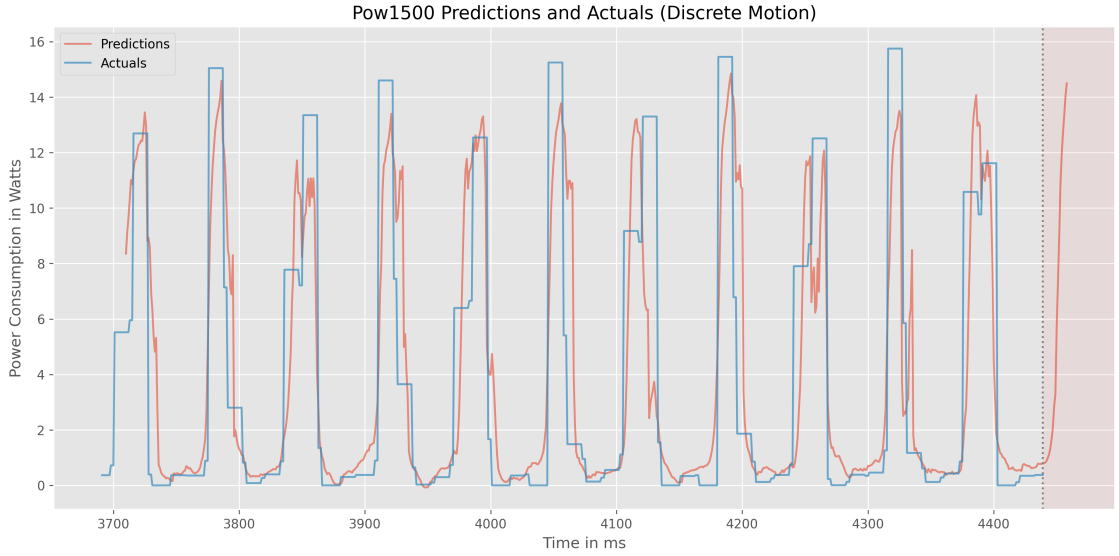

Pow1000 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
3700
3800
3900
4000
4100
4200
4300
4400

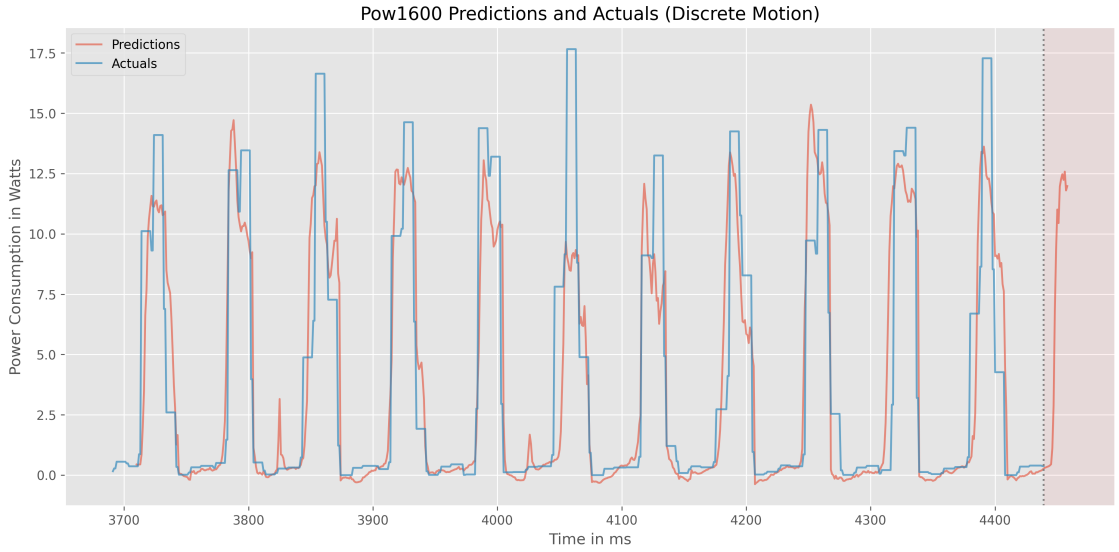

Pow1100 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
3700
3800
3900
4000
4100
4200
4300
4400

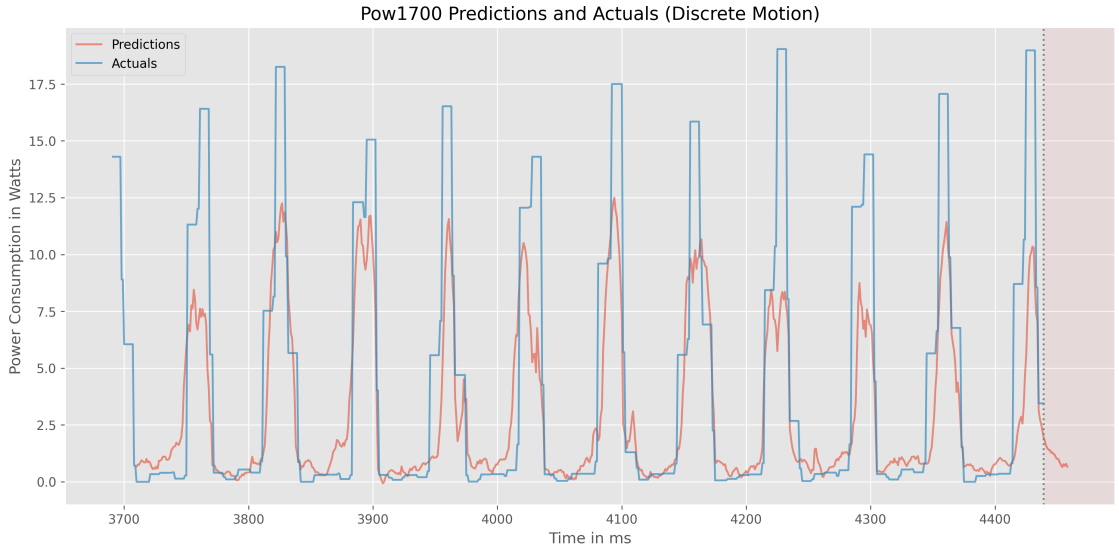

Pow1200 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
12
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

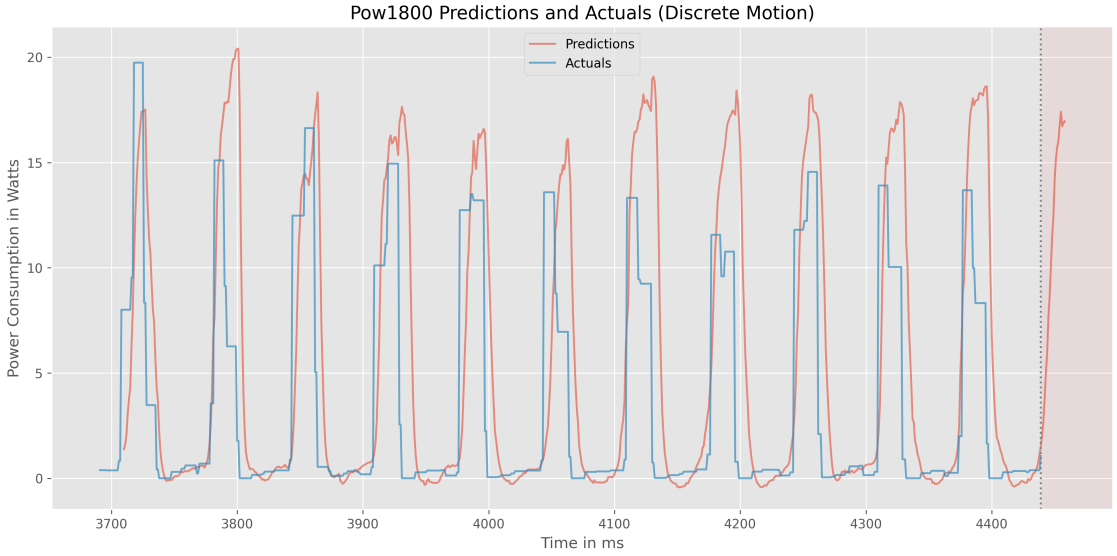

Pow1300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
12
14
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

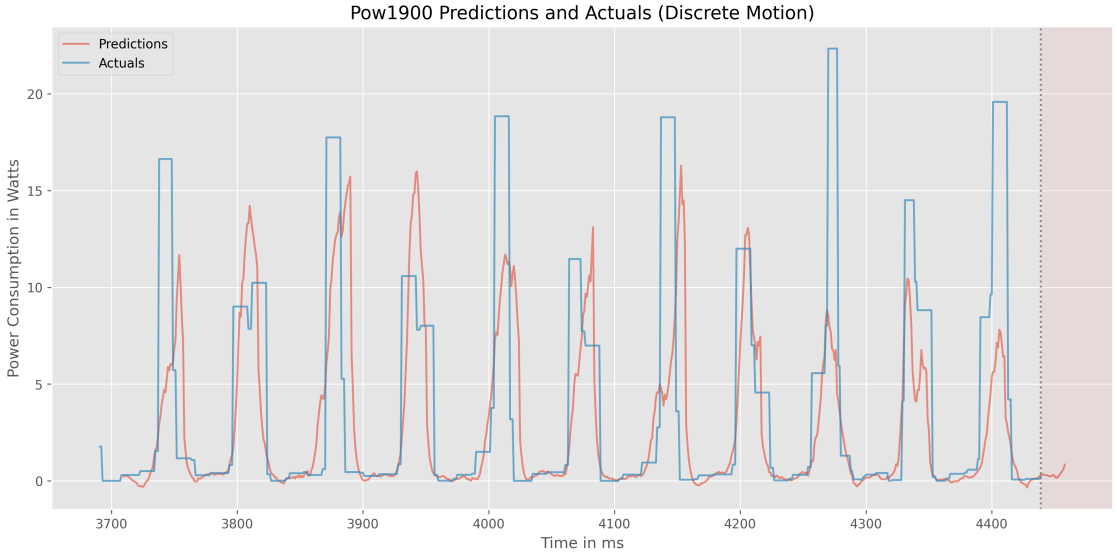

Pow1400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

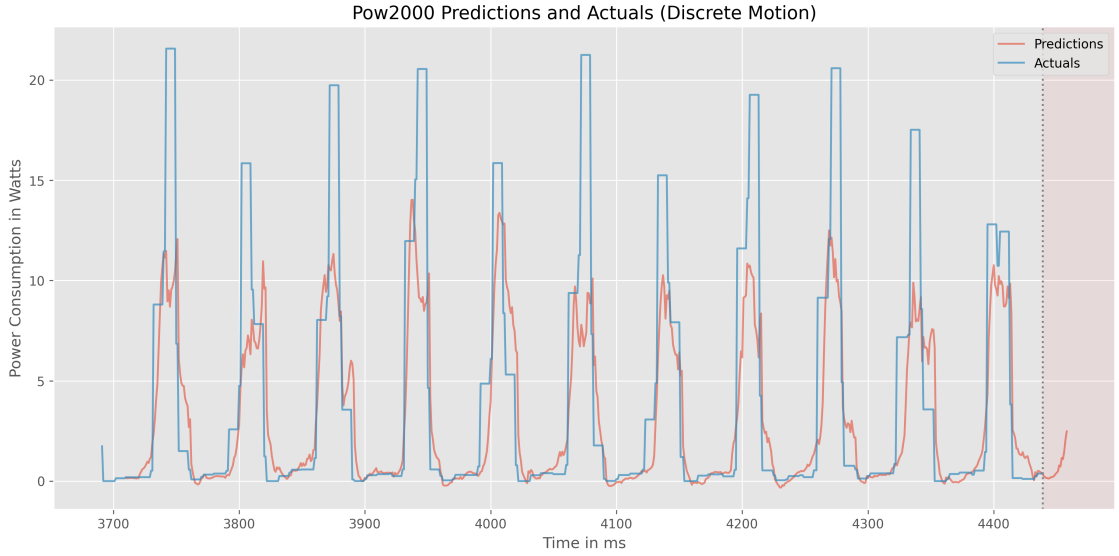

Pow1500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms


Pow1600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms


Pow1700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

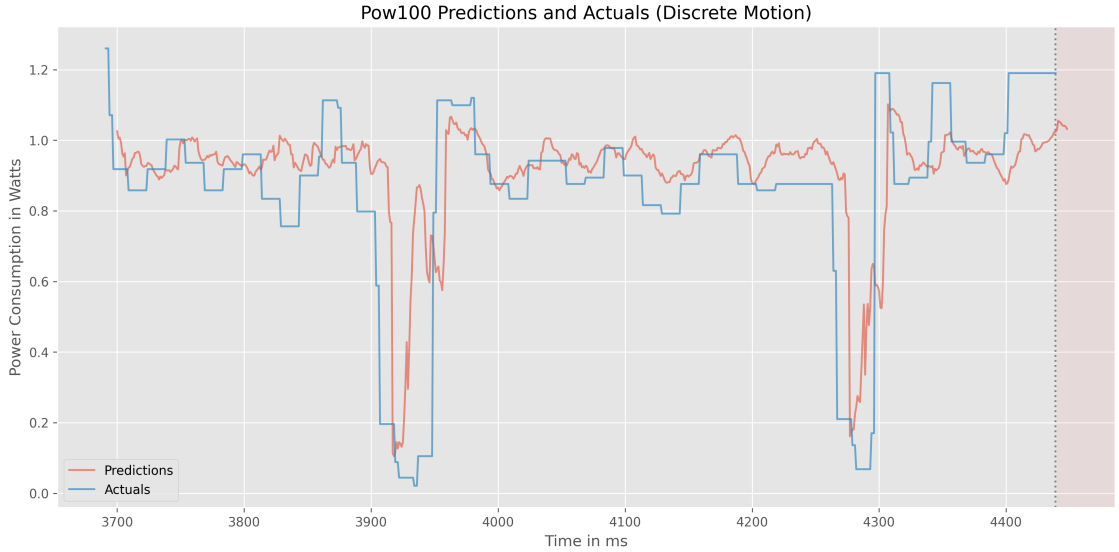

Pow1800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
5
10
15
20
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

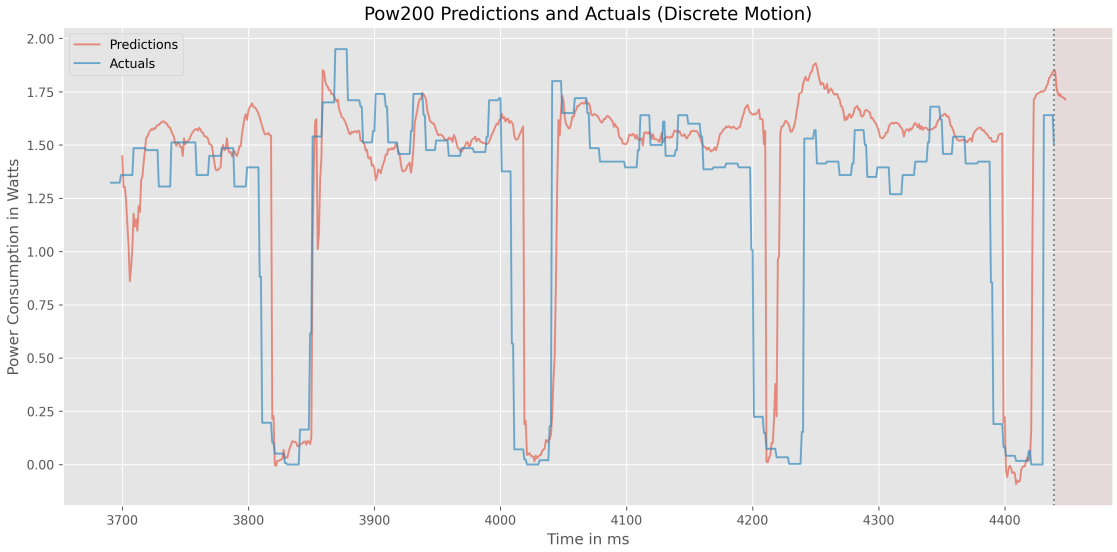

Pow1900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
5
10
15
20
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

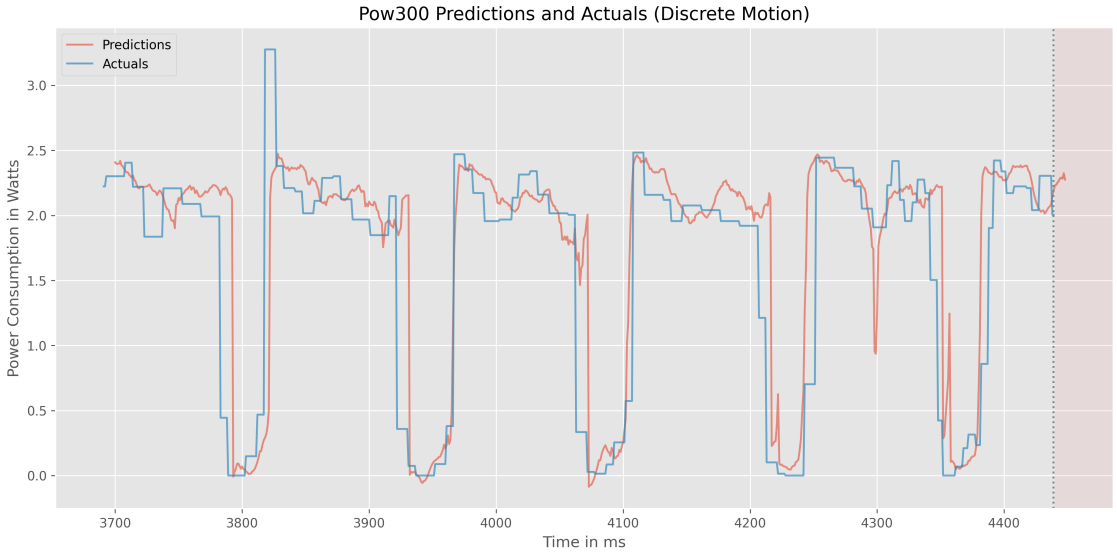


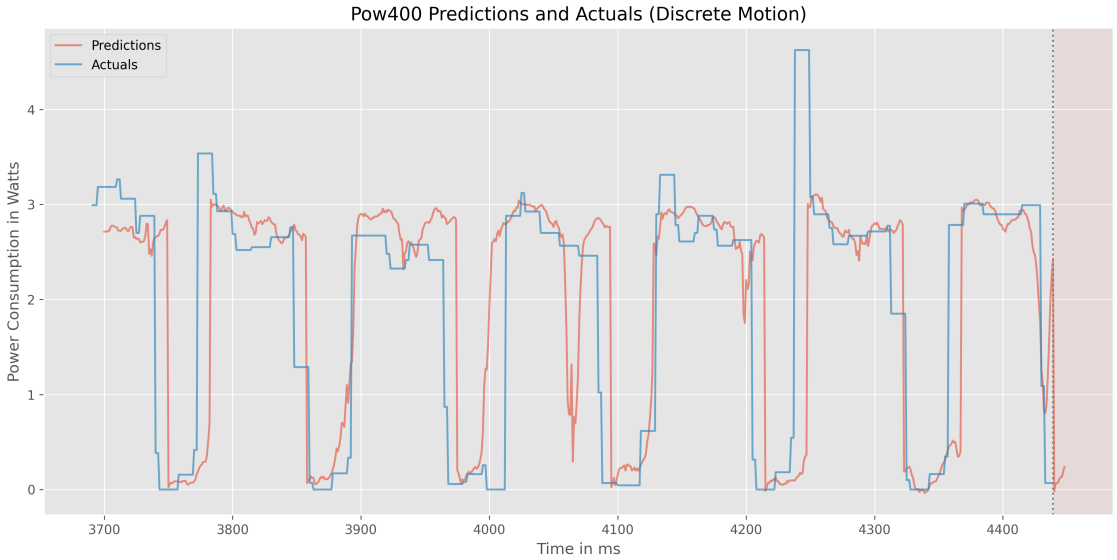
2.2.2 With Forecast Horizon of t+10 [Range: last 750 Timestamps]

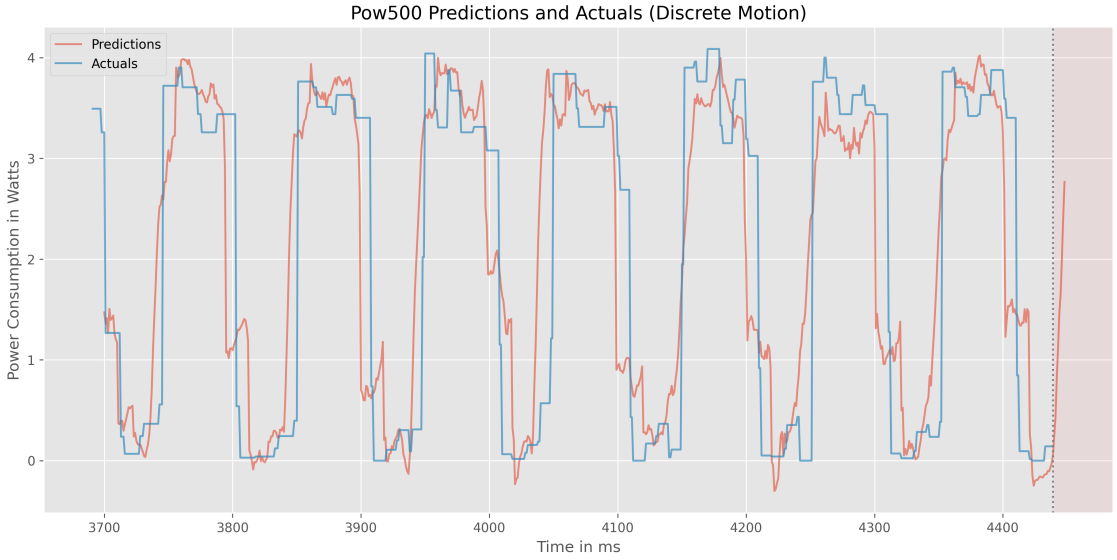
[ ]:
```
print_act_and_pred_tables(re_yhat2,re_y_test2,10, -750,-1)
```

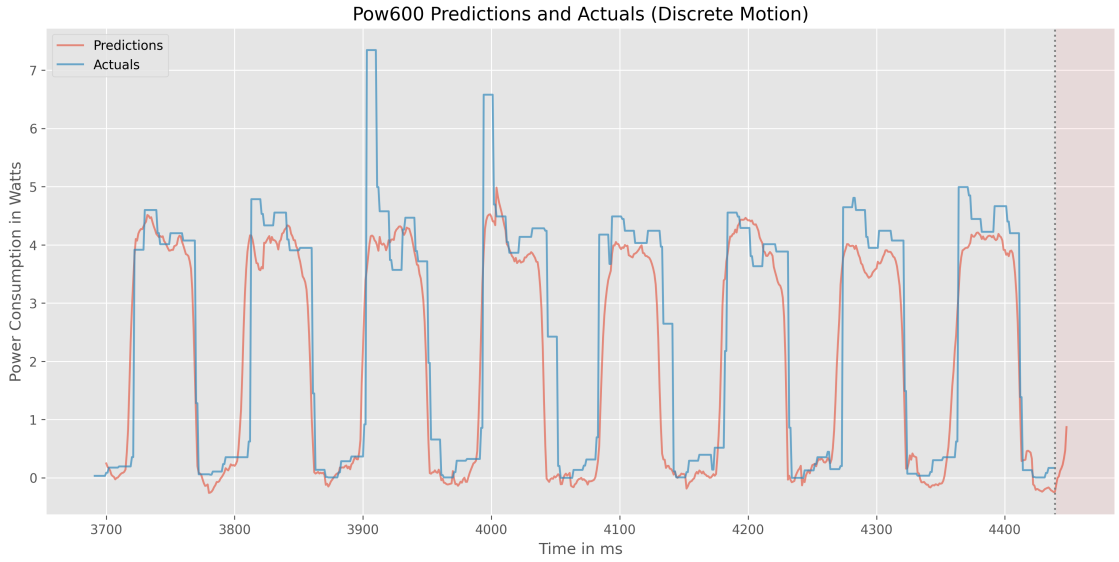


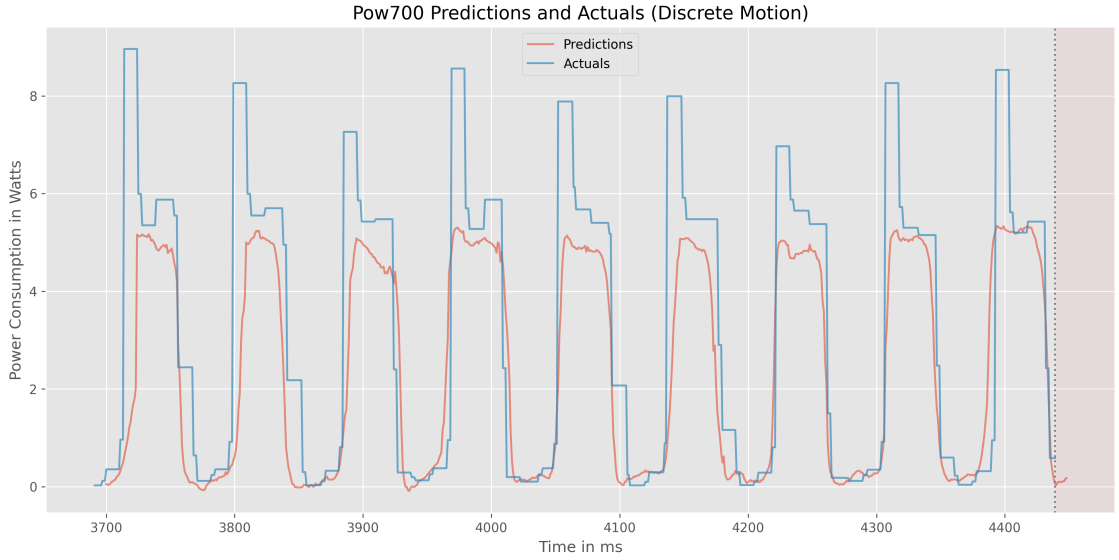

Pow200 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

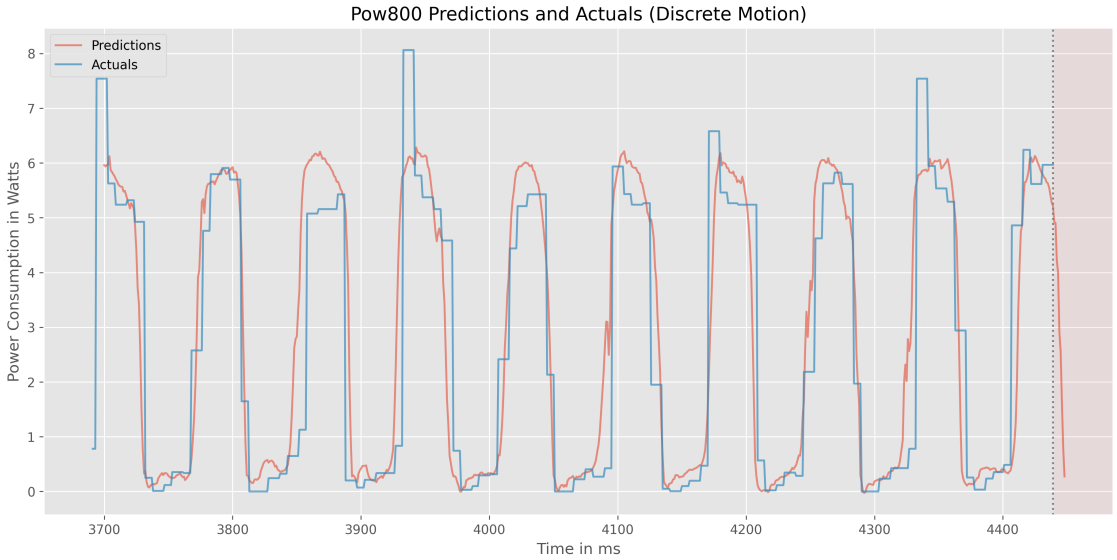

Pow300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

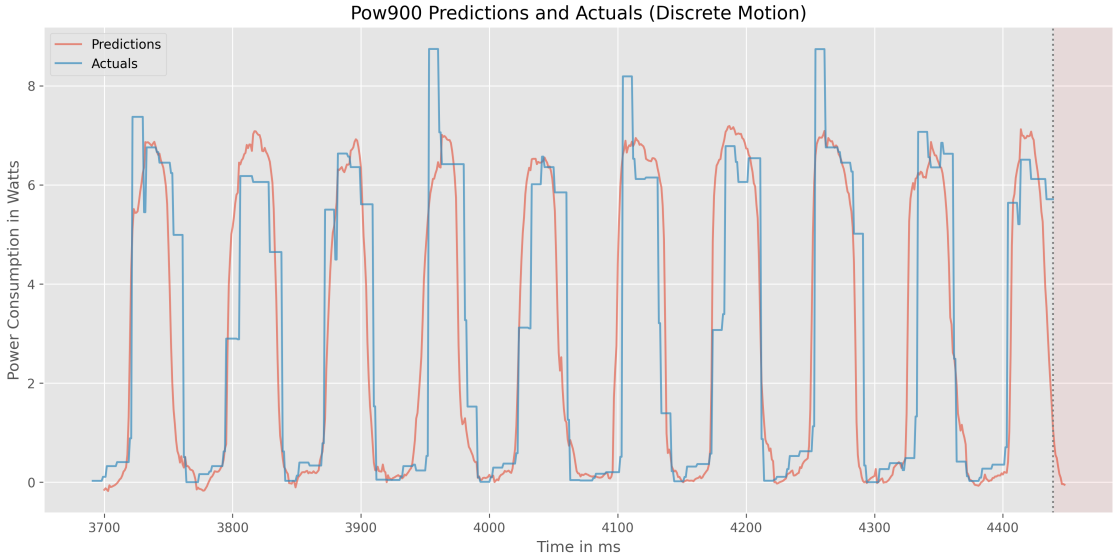

Pow400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
1
2
3
4
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

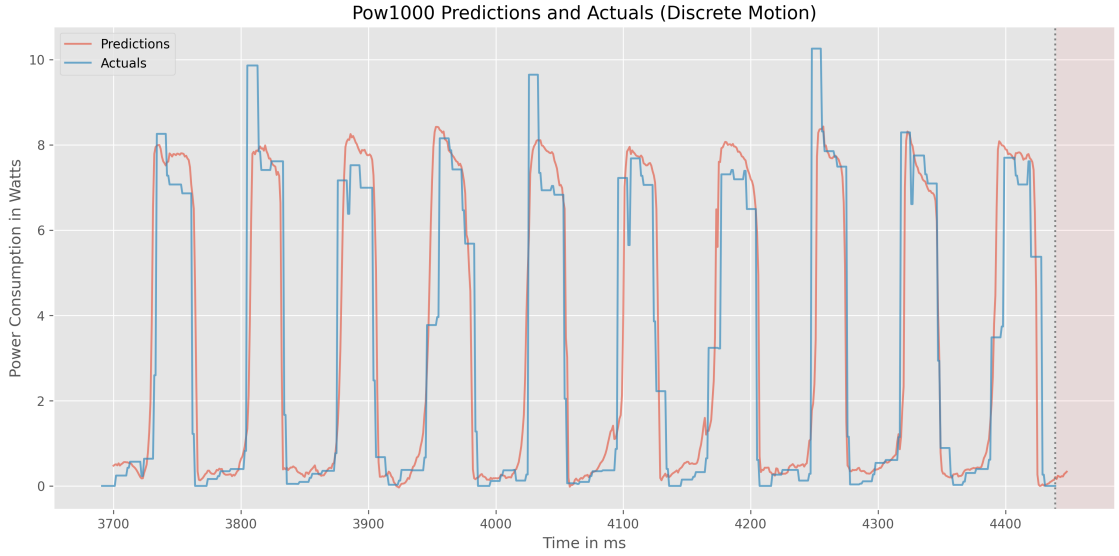

Pow500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
1
2
3
4
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

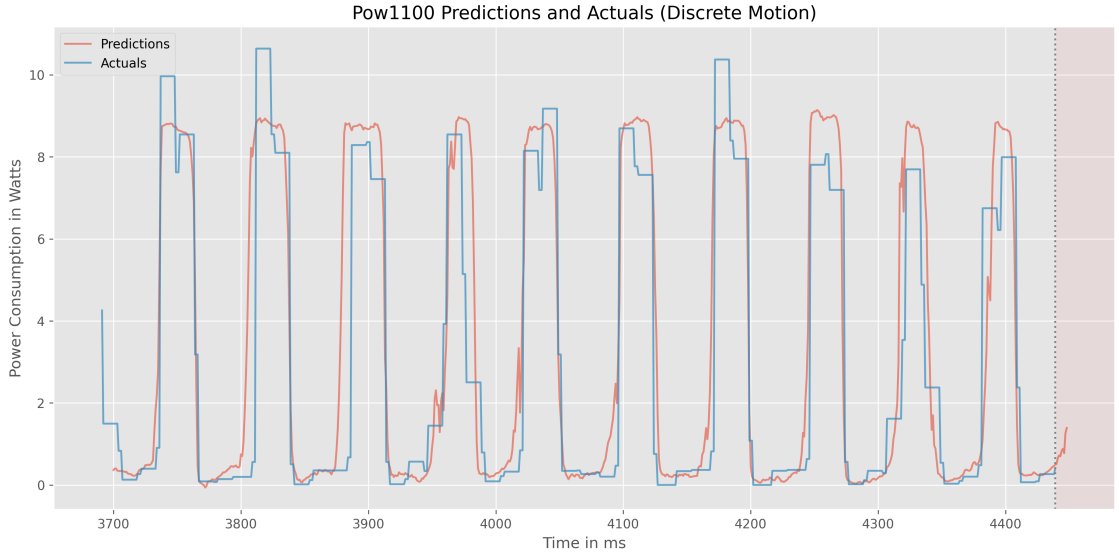

Pow600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

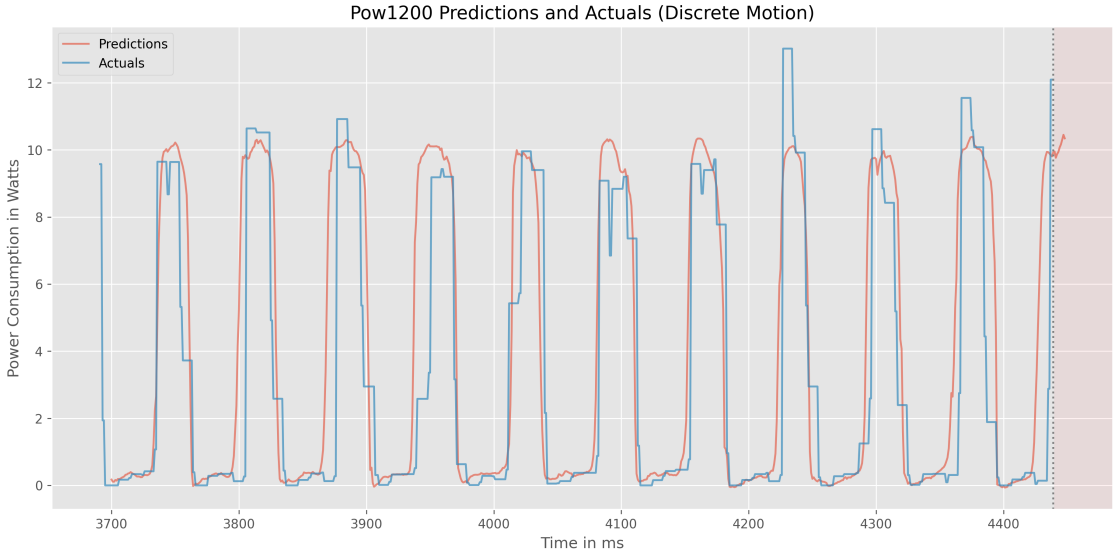

Pow700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

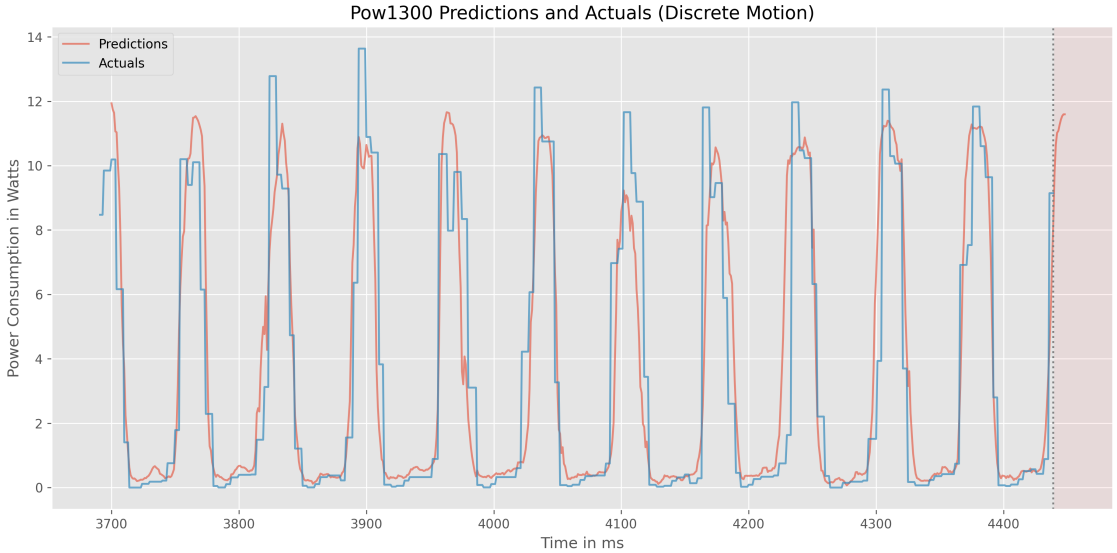

Pow800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
7
8
3700
3800
3900
4000
4100
4200
4300
4400

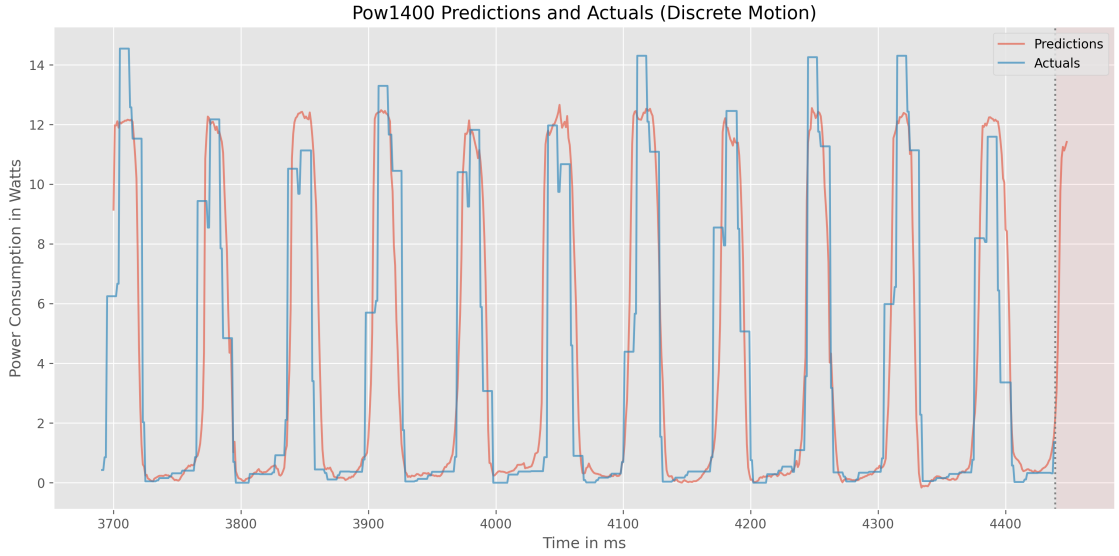

Pow900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
3700
3800
3900
4000
4100
4200
4300
4400

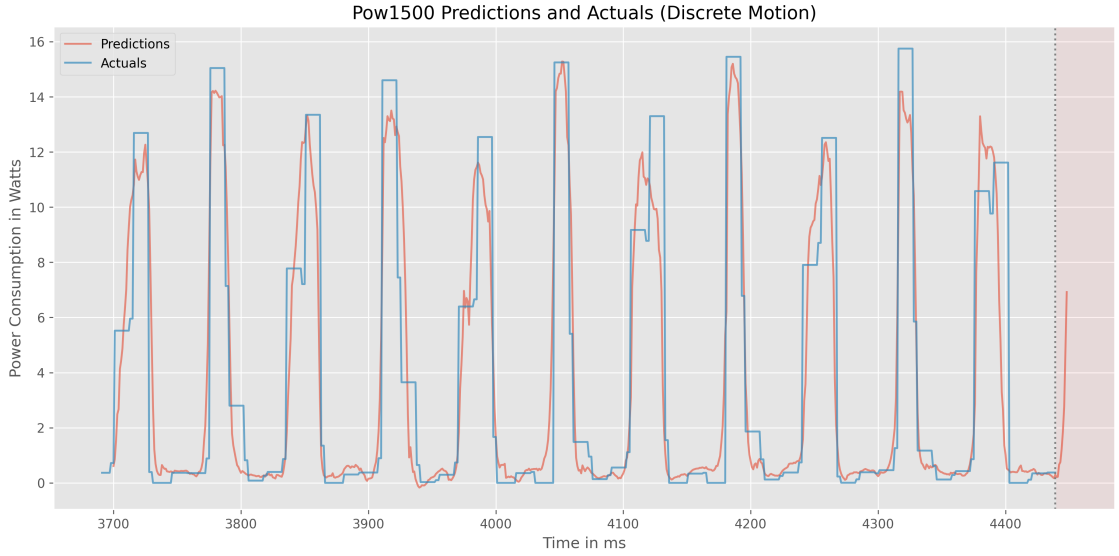

Pow1000 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

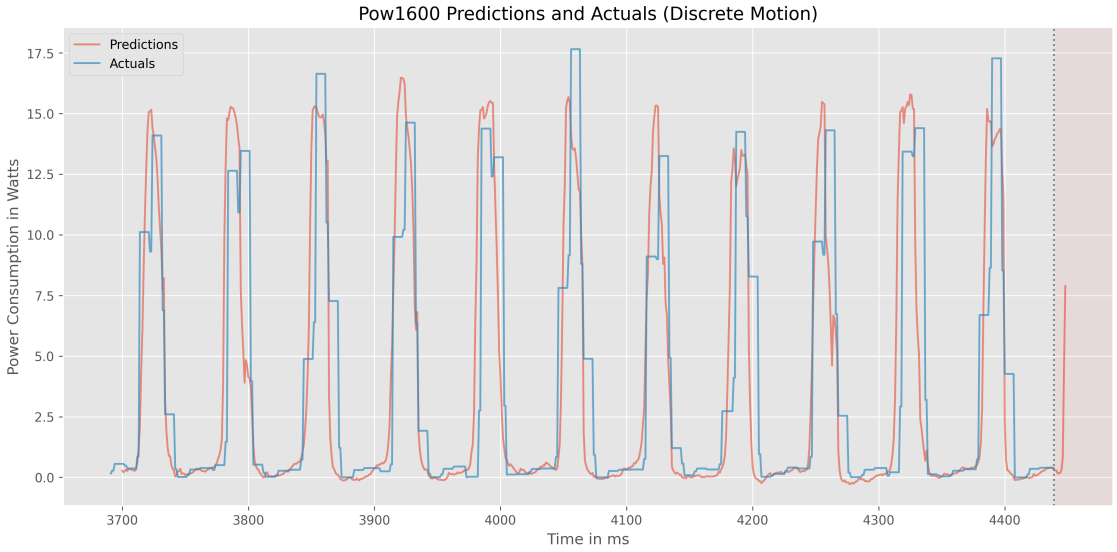

Pow1100 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

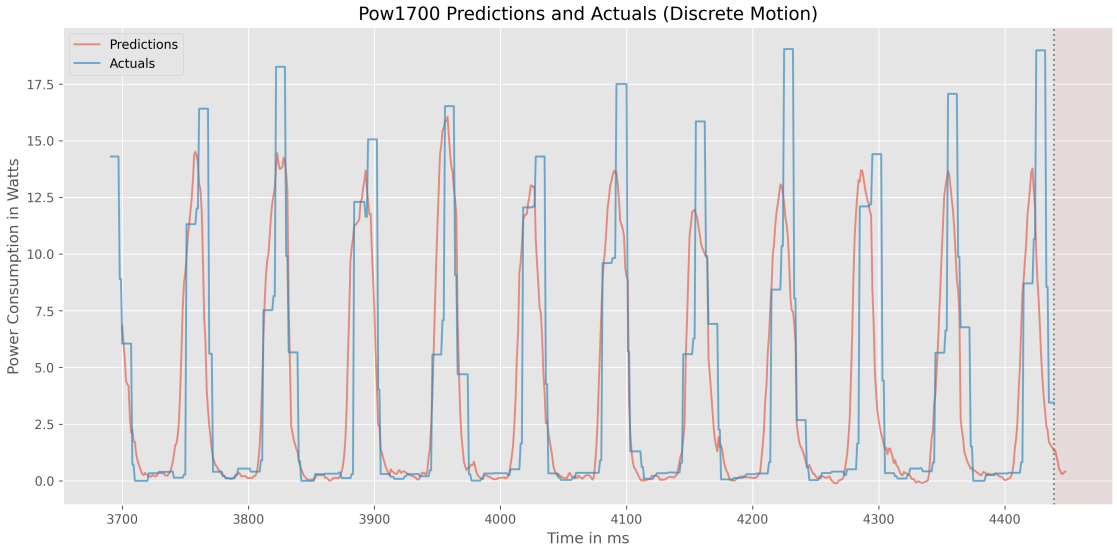

Pow1200 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
12
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

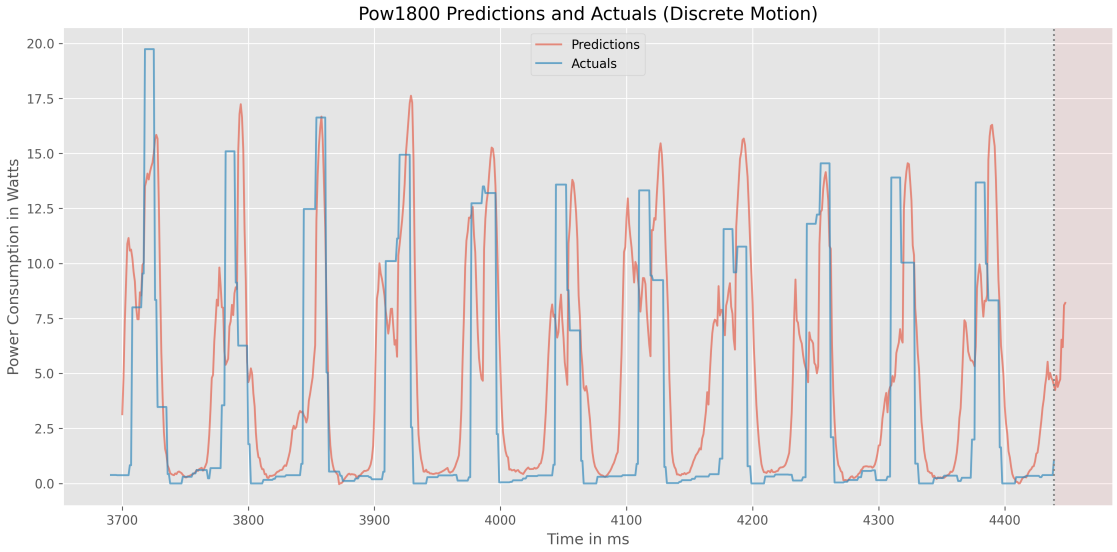

Pow1300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
12
14
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms

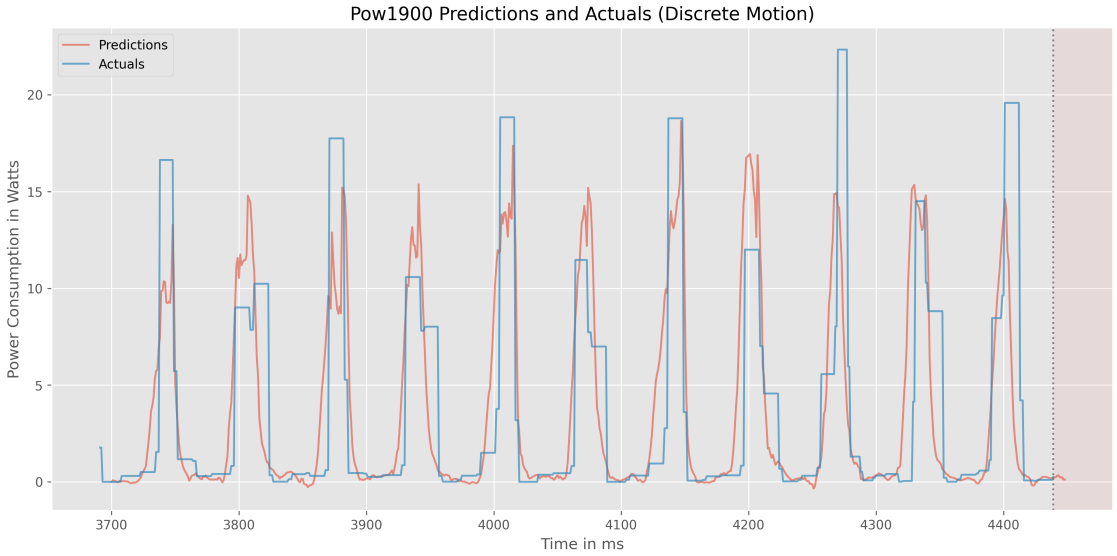

Pow1400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
3700
3800
3900
4000
4100
4200
4300
4400

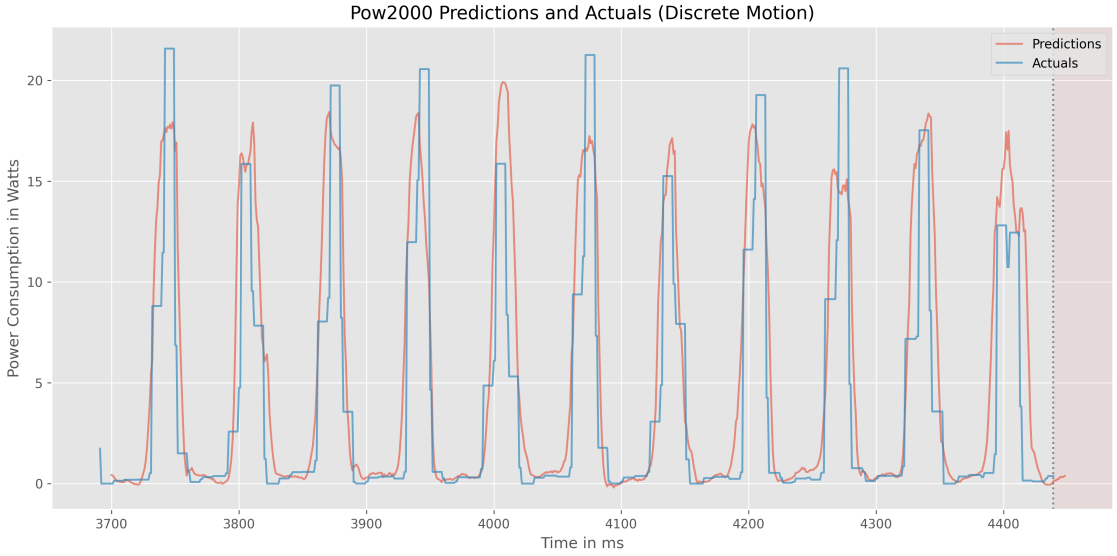

Pow1500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
16
3700
3800
3900
4000
4100
4200
4300
4400


Pow1600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms


Pow1700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3700
3800
3900
4000
4100
4200
4300
4400
Time in ms


Pow1800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
3700
3800
3900
4000
4100
4200
4300
4400


Pow1900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
20
15
10
5
0
3700
3800
3900
4000
4100
4200
4300
4400

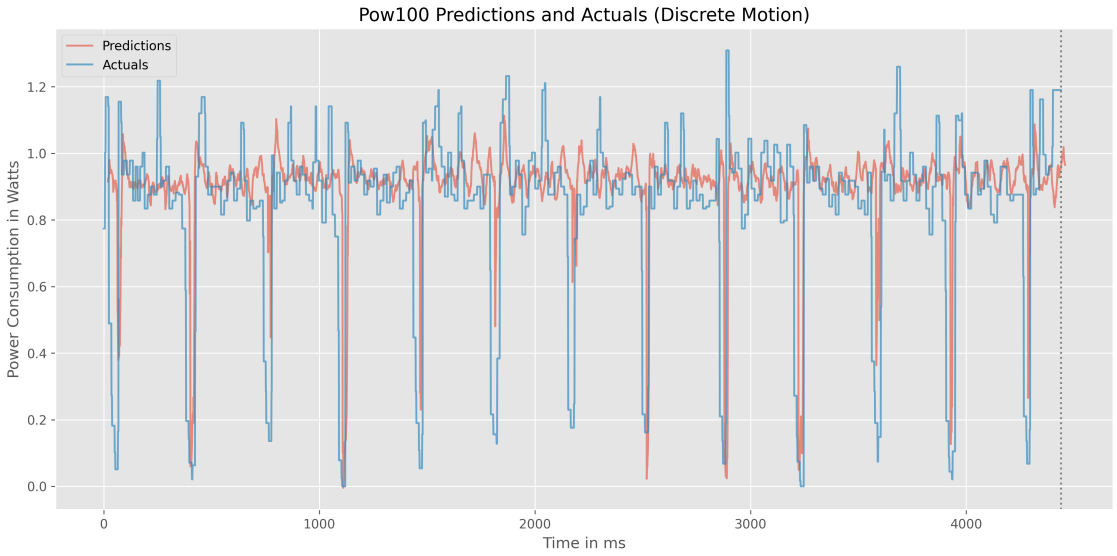


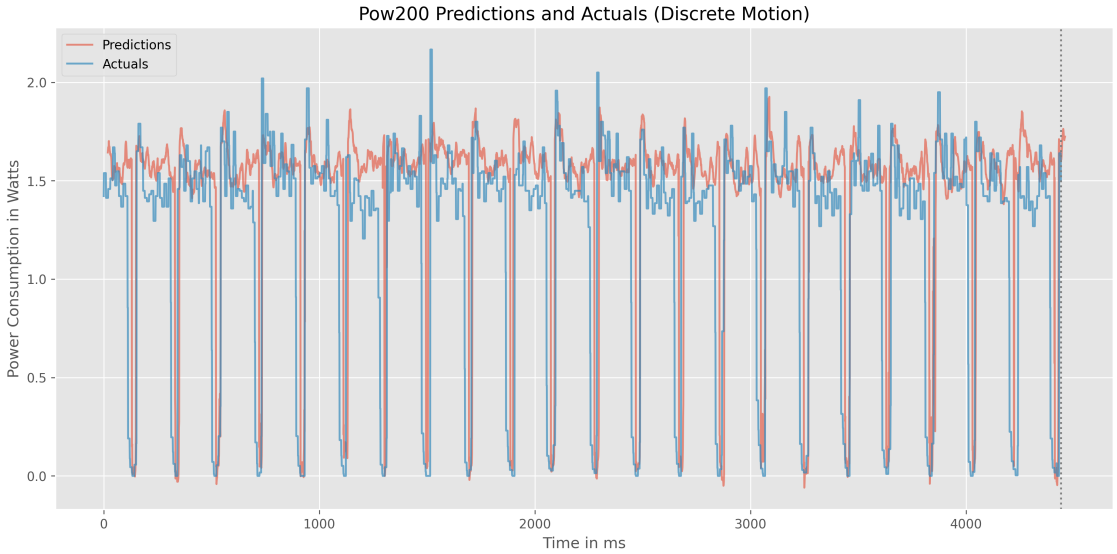
2.2.3 With Forecast Horizon of t+20 [Range: All Timestamps]

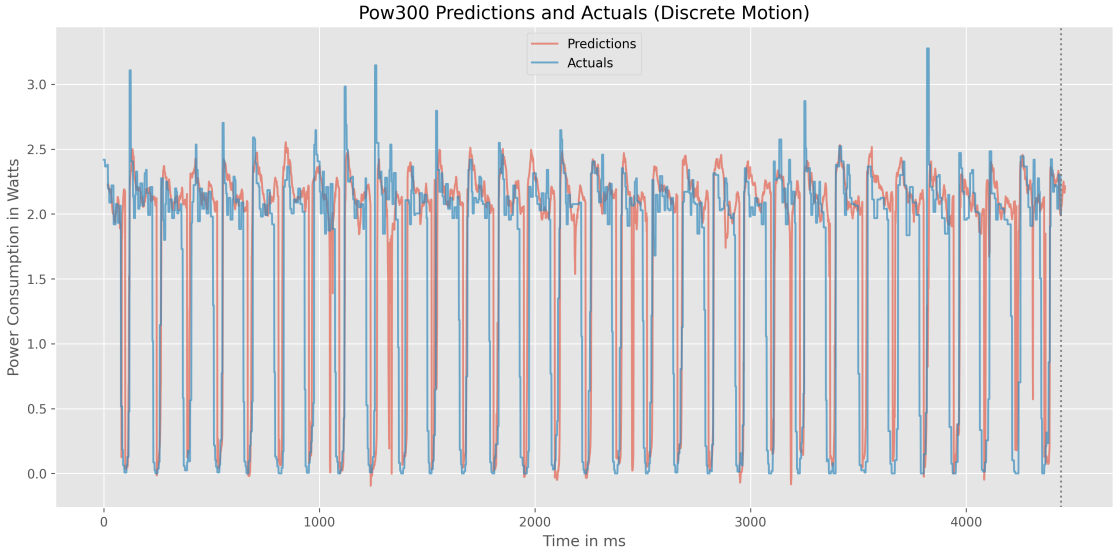
[ ]:
```
from sklearn.metrics import r2_score
def print_act_and_pred_tables_all(yhat,ytest,forecast_horizon,␣
 ↪start_graph,end_graph):
  pow_preds = []
  pow_actuals = []
  for i in range(20):
    pow_preds.append(yhat[:, forecast_horizon-1, i]) # 1 refers to the forecast␣
 ↪horizon --> t+1; shape of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

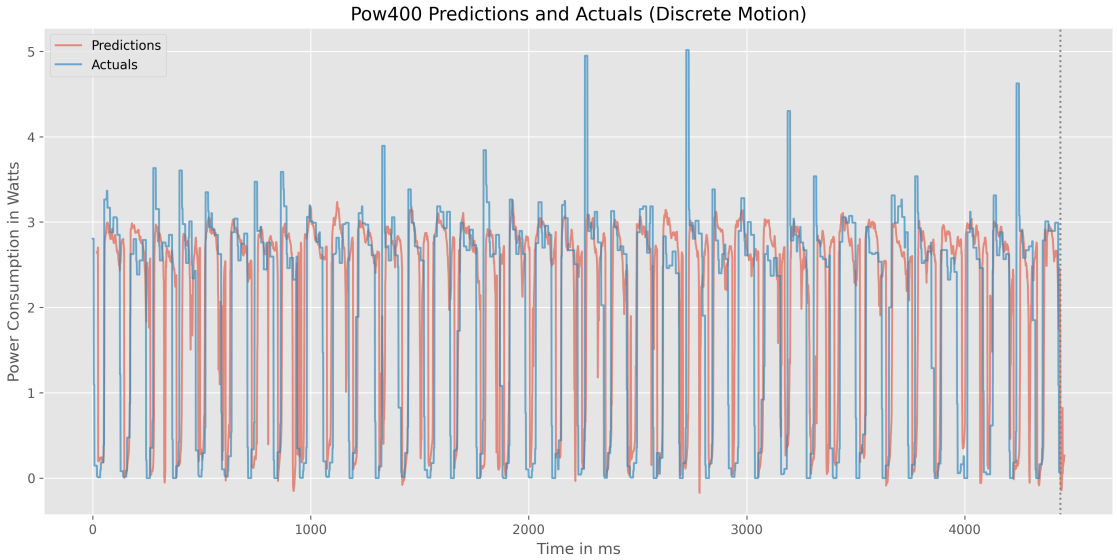
```
    p_temp = predictions.to_frame()
    p_temp['time_index_shift'] = predictions.index+forecast_horizon-1
    p_temp.set_index('time_index_shift', inplace=True)
    # Plot the data
    plt.plot(p_temp, alpha=0.6)
    plt.plot(actuals, alpha=0.7)
    # Set the plot title and axis labels
    plt.title(f'Pow{100*(i+1)} 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()
```

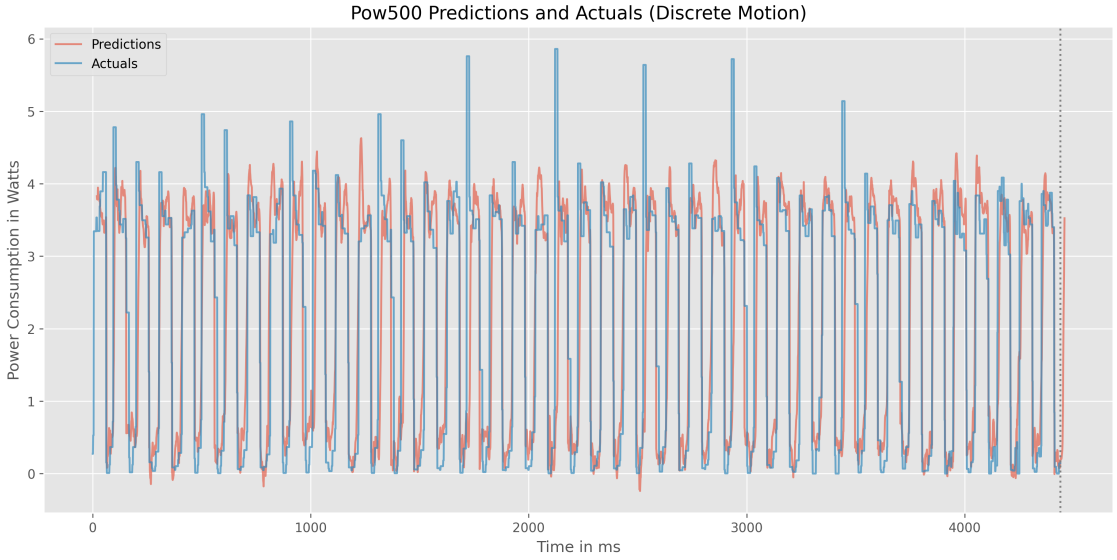
[ ]:
```
print_act_and_pred_tables_all(re_yhat2,re_y_test2,20,0,-1)
```

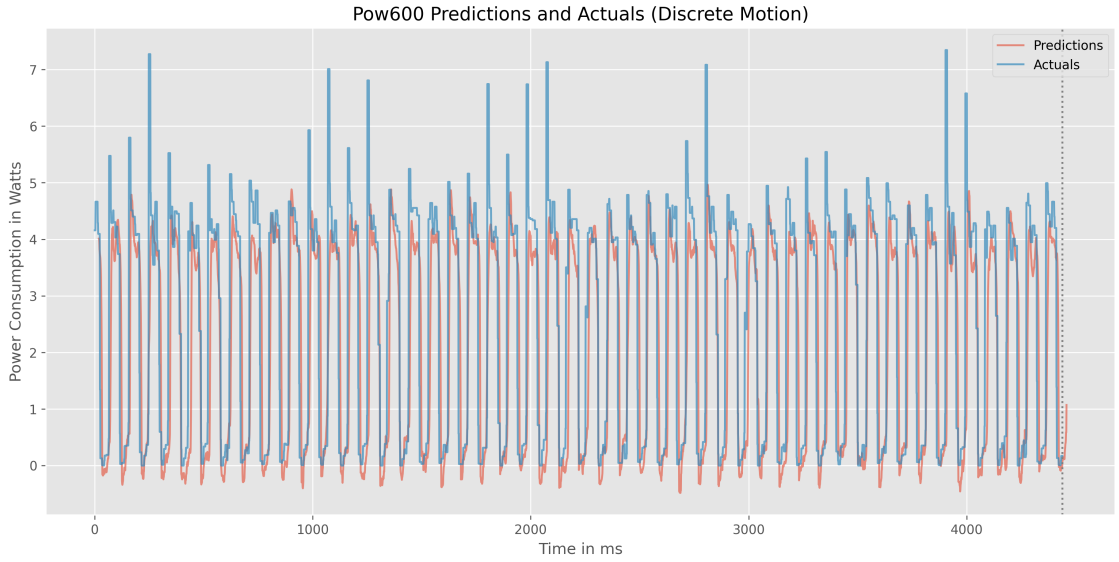


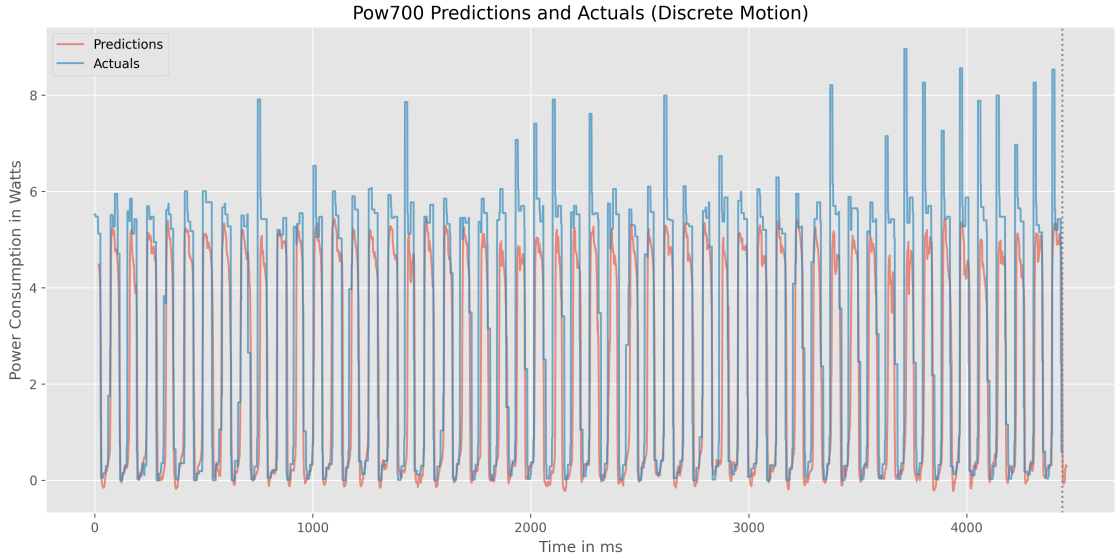

Pow200 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0.0
0.5
1.0
1.5
2.0
0
1000
2000
3000
4000
Time in ms

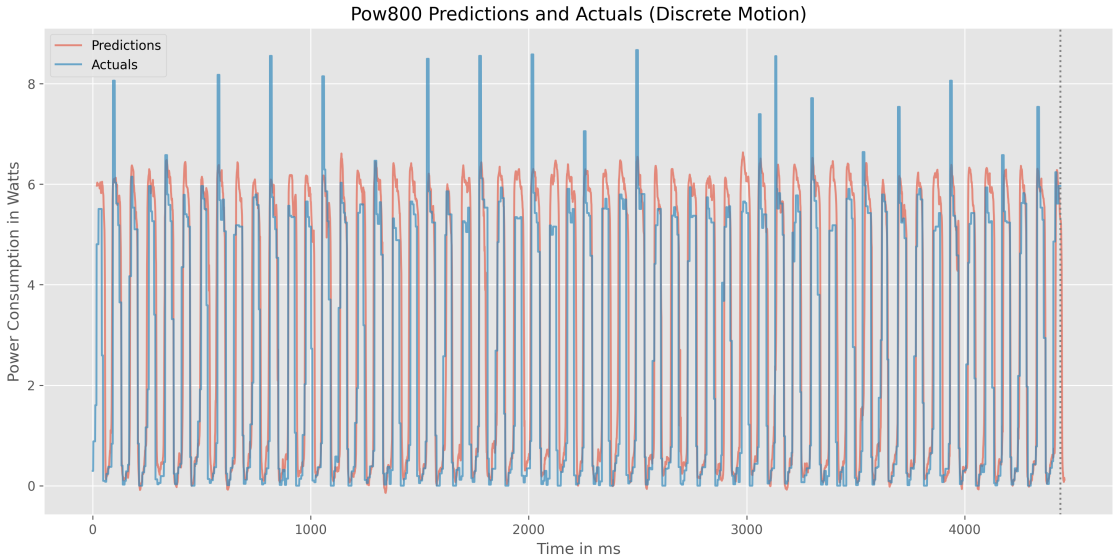

Pow300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
1000
2000
3000
4000
Time in ms

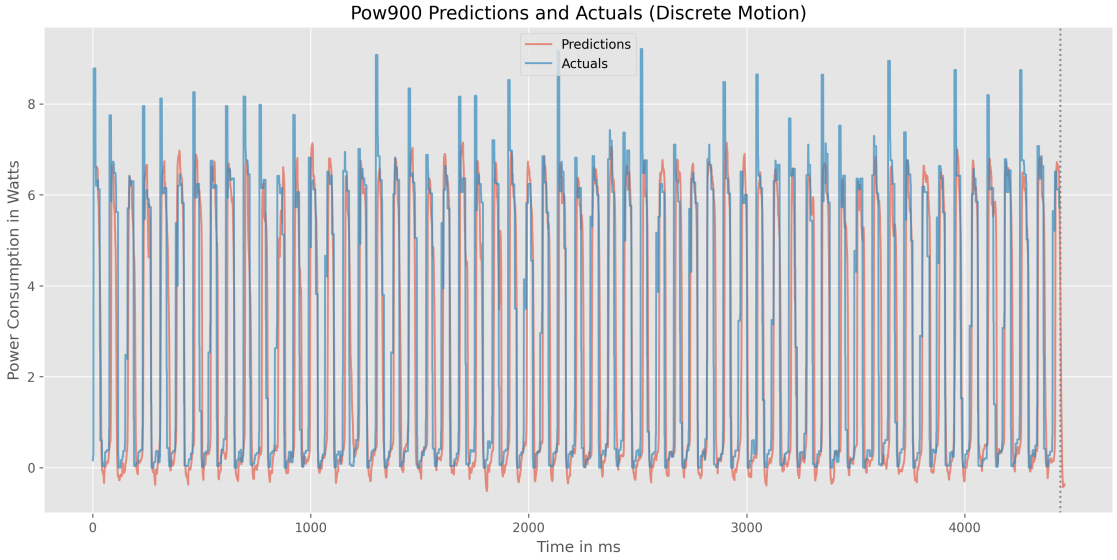

Pow400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
0
1000
2000
3000
4000

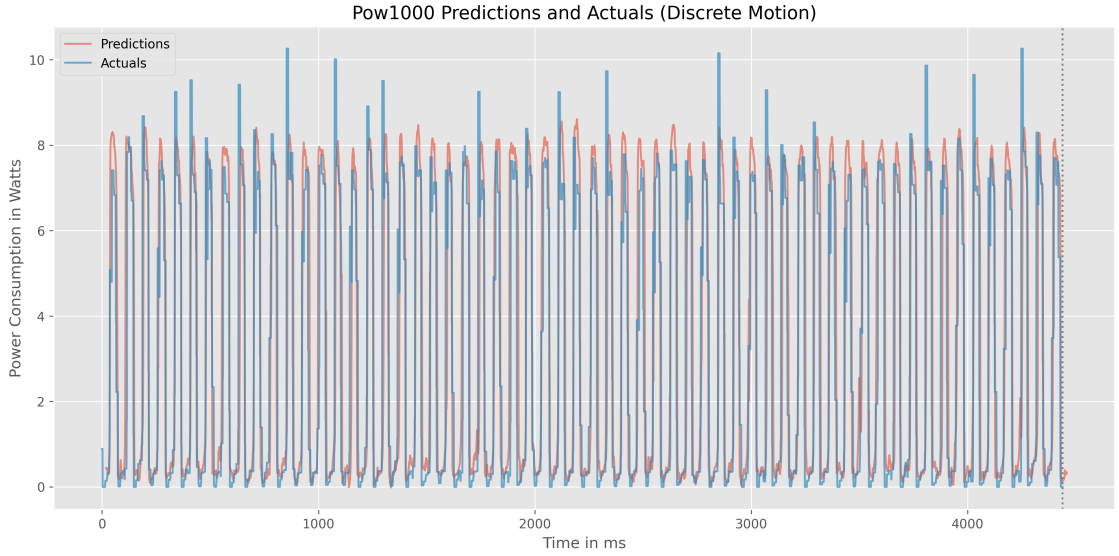

Pow500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
0
1000
2000
3000
4000

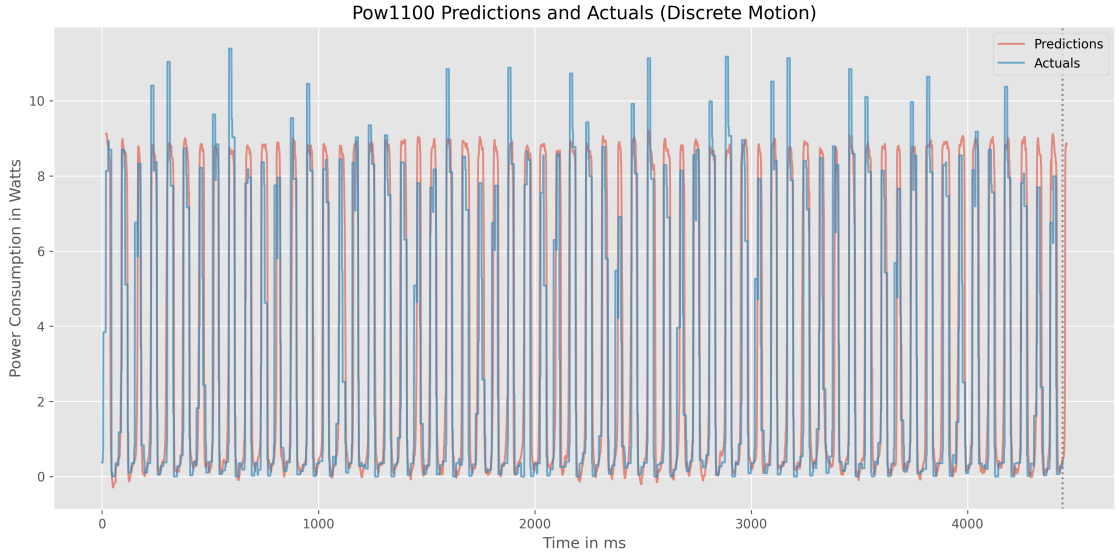

Pow600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
7
0
1000
2000
3000
4000

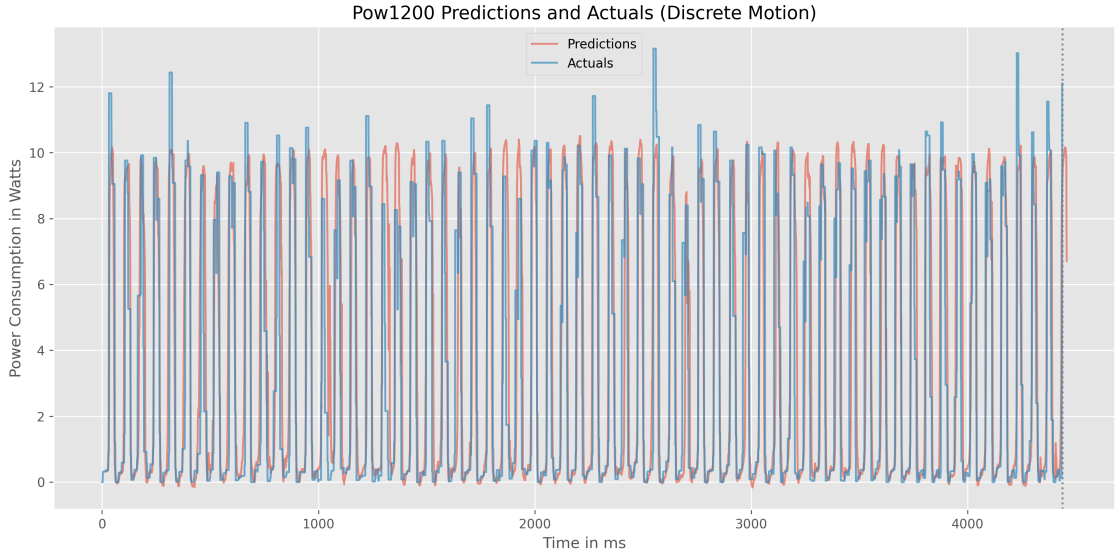

Pow700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
0
1000
2000
3000
4000

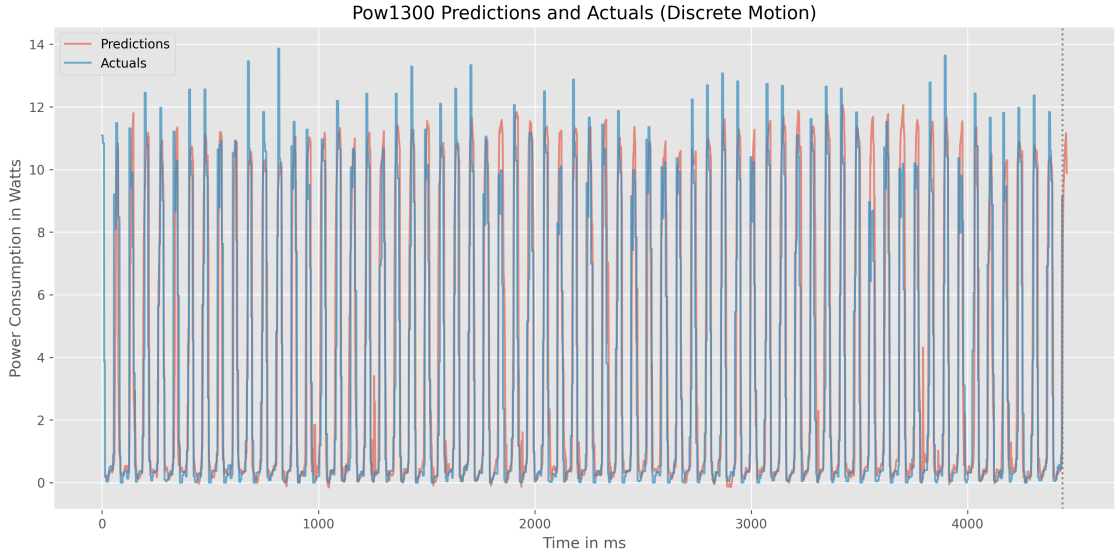

Pow800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
0
1000
2000
3000
4000
Time in ms

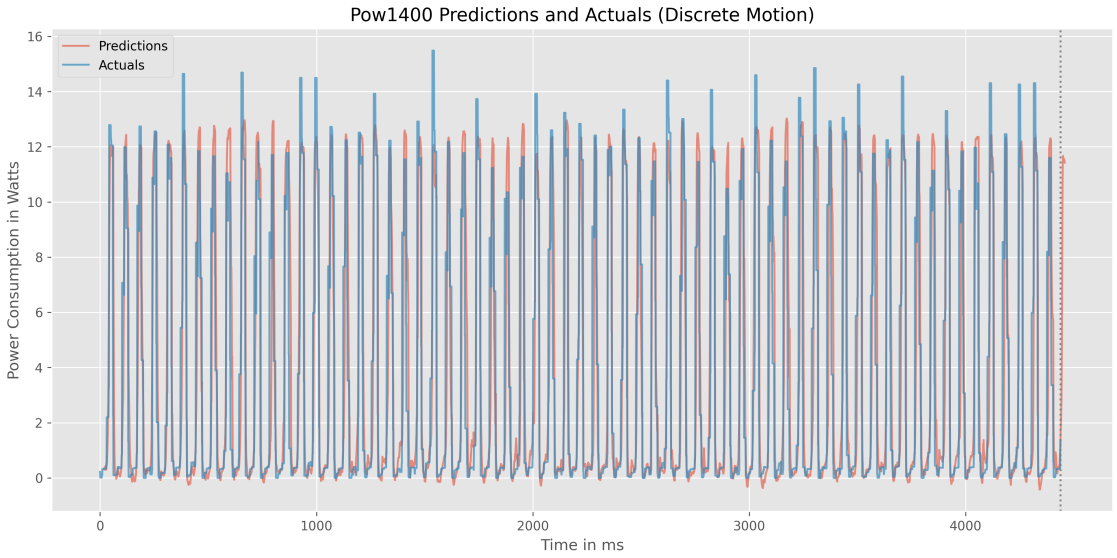

Pow900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
0
1000
2000
3000
4000
Time in ms

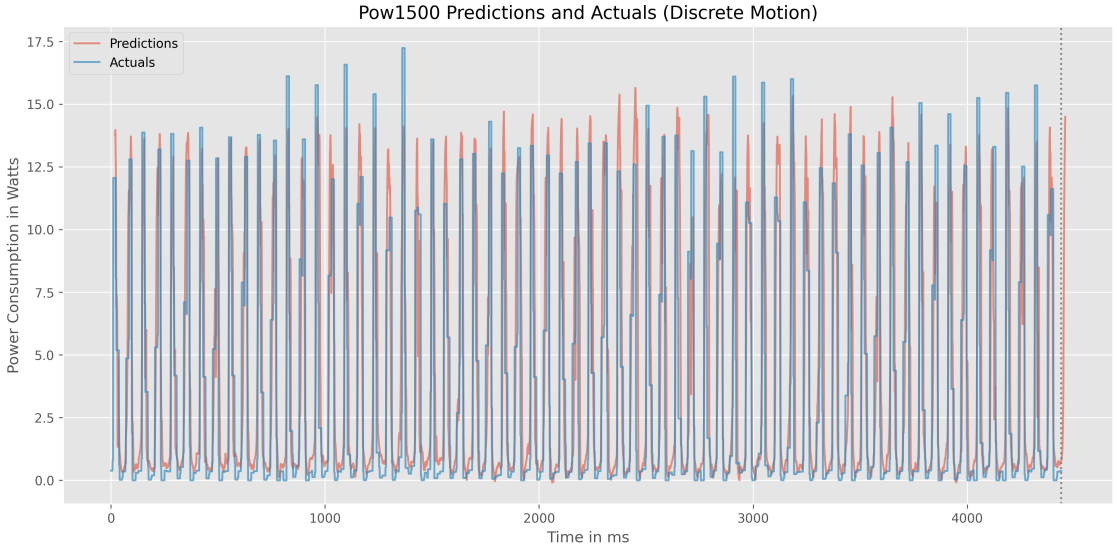

Pow1000 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1000
2000
3000
4000
0
2
4
6
8
10

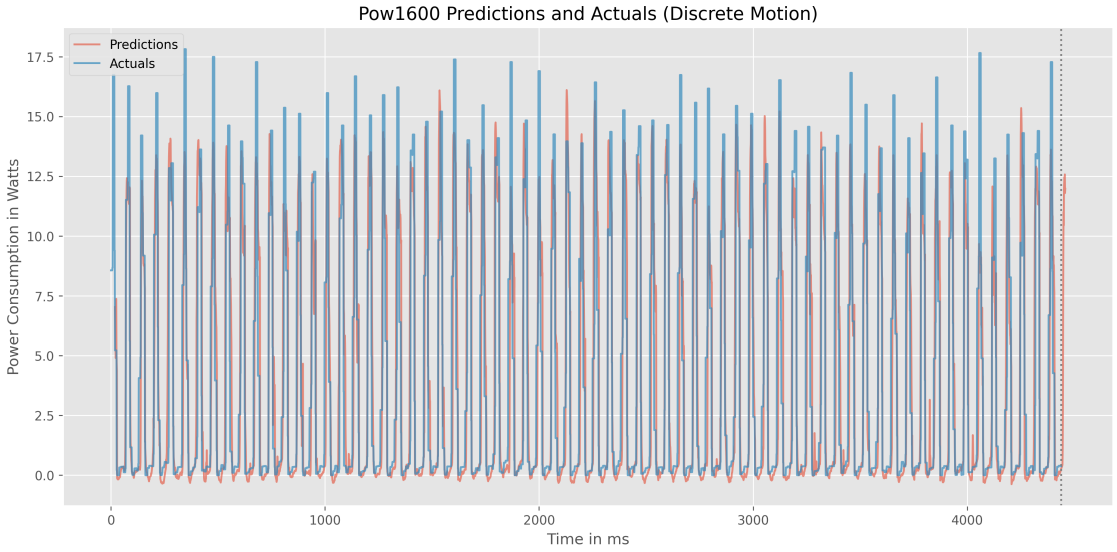

Pow1100 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1000
2000
3000
4000
0
2
4
6
8
10

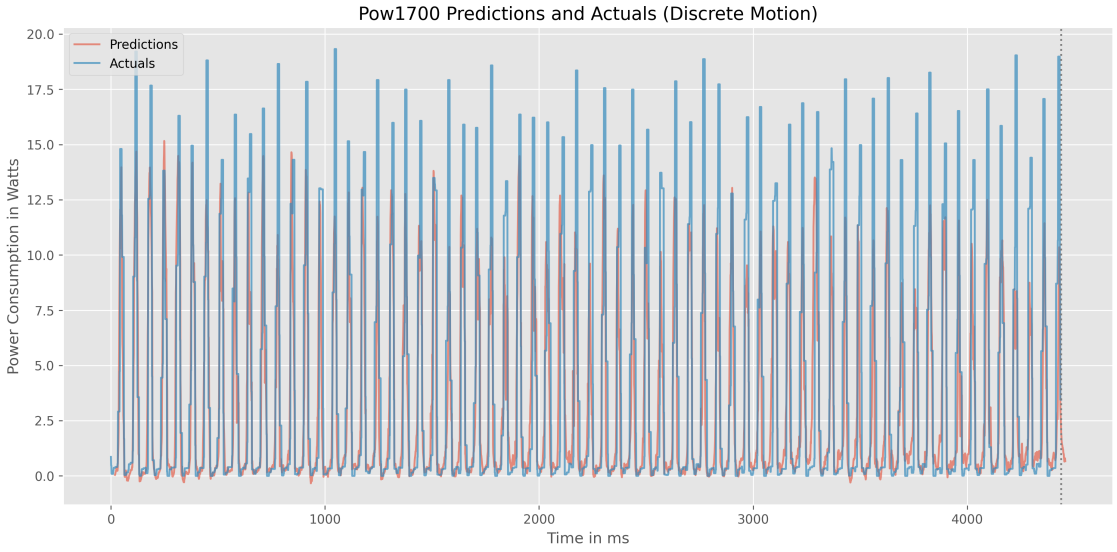

Pow1200 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
0
1000
2000
3000
4000

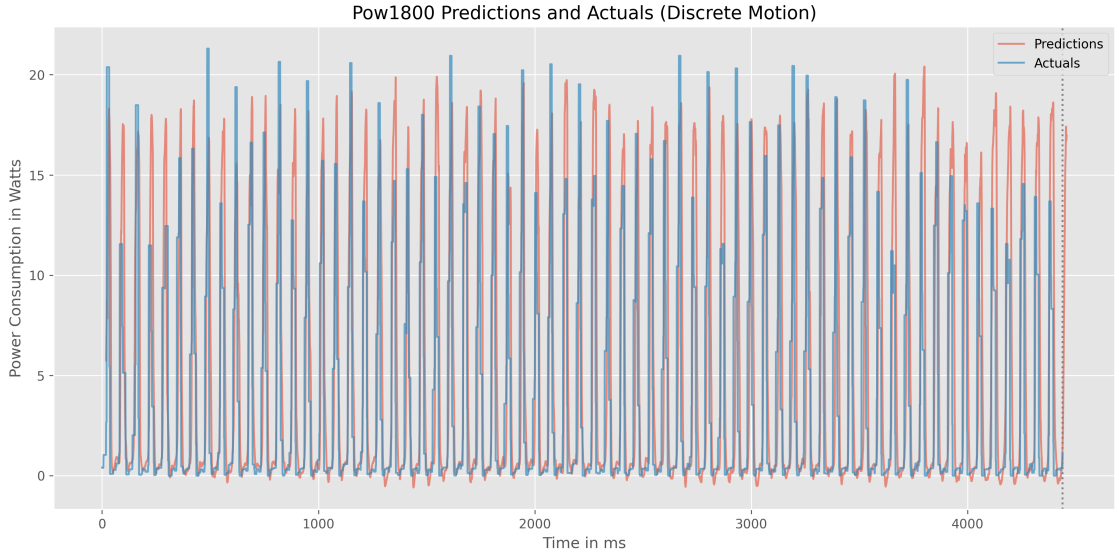

Pow1300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
0
1000
2000
3000
4000

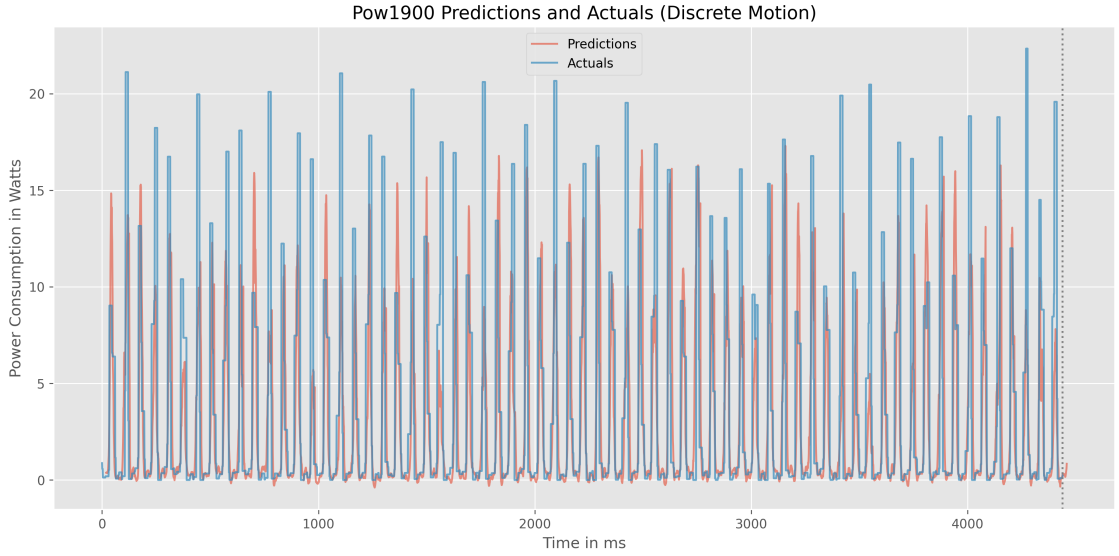

Pow1400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
16
0
1000
2000
3000
4000

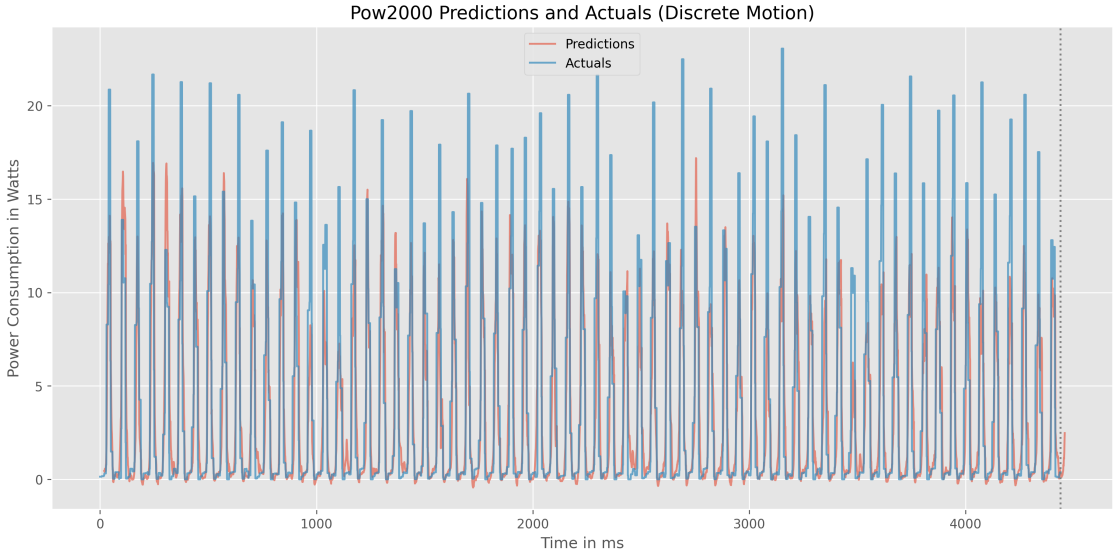

Pow1500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
1000
2000
3000
4000


Pow1600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
1000
2000
3000
4000


Pow1700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0
1000
2000
3000
4000


Pow1800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
5
10
15
20
0
1000
2000
3000
4000
Time in ms


Pow1900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Power Consumption in Watts
0
5
10
15
20
0
1000
2000
3000
4000
Time in ms

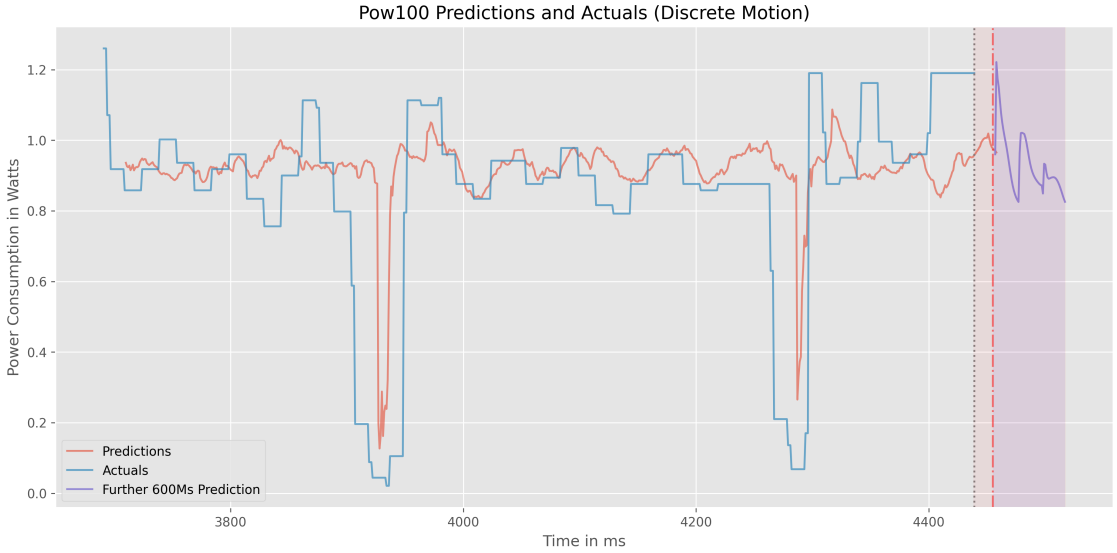


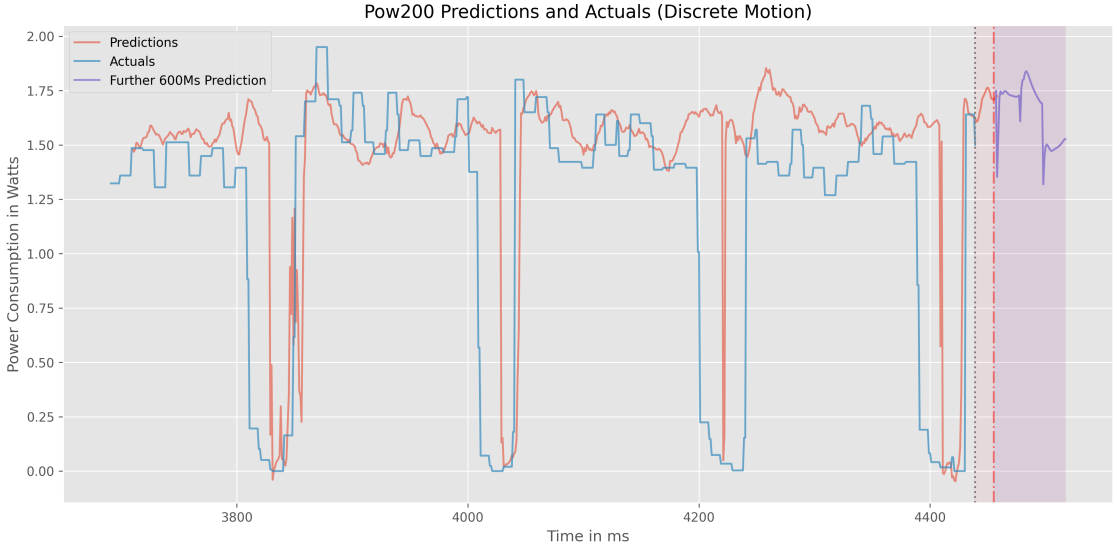
2.2.4 With Forecast Horizon of t+20 [Range: last 750 Timestamps] & *Further 600ms Prediction (Addon – Recursive Method)*

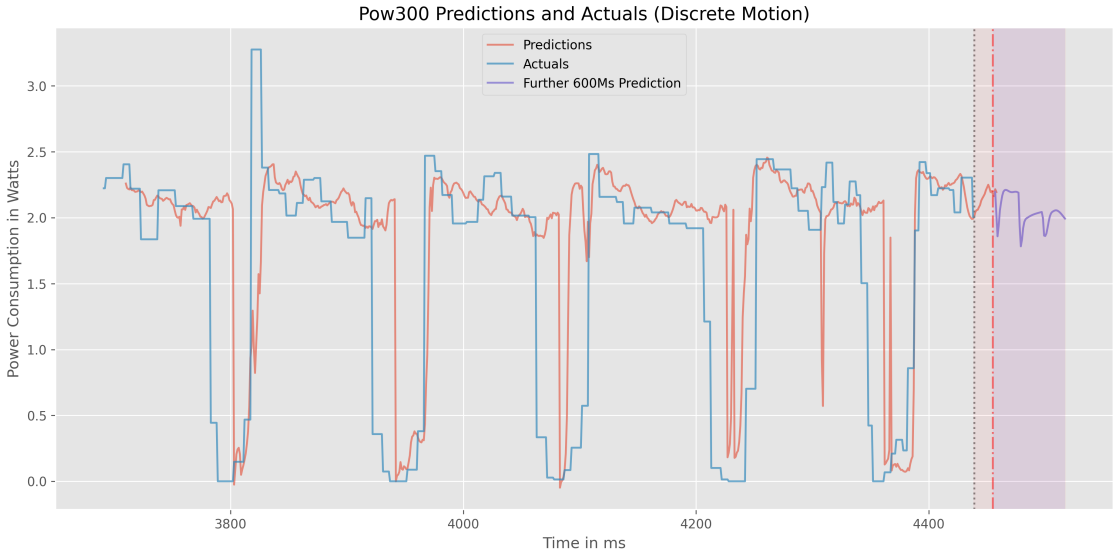
[ ]:
```
def print_act_and_pred_tables_withFurther(yhat, ytest, forecast_horizon,␣
↪start_graph, end_graph):
  pow_preds = [yhat[:, forecast_horizon-1, i] for i in range(20)]
  pow_actuals = [ytest[:, 0, i] for i in range(20)]
  data = {}
  for i in range(20):
    data[f"Pow{100*(i+1)} Predictions"] = 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)
```

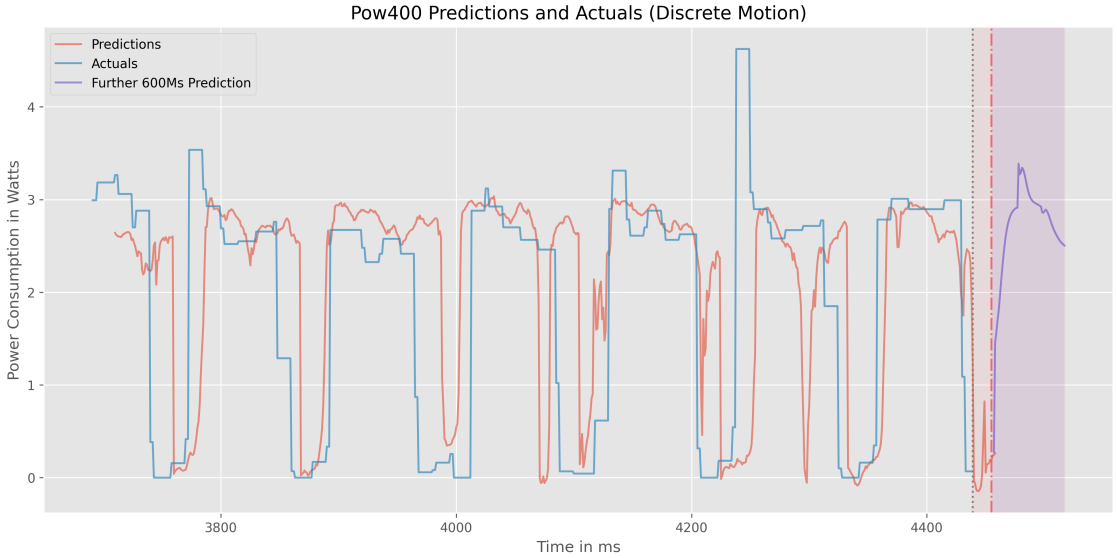
```
    plt.axvline(x=4455, linestyle="-.", color="red", alpha=0.5)
    plt.legend(['Predictions', 'Actuals', "Further 600Ms Prediction"])
    plt.show()
```

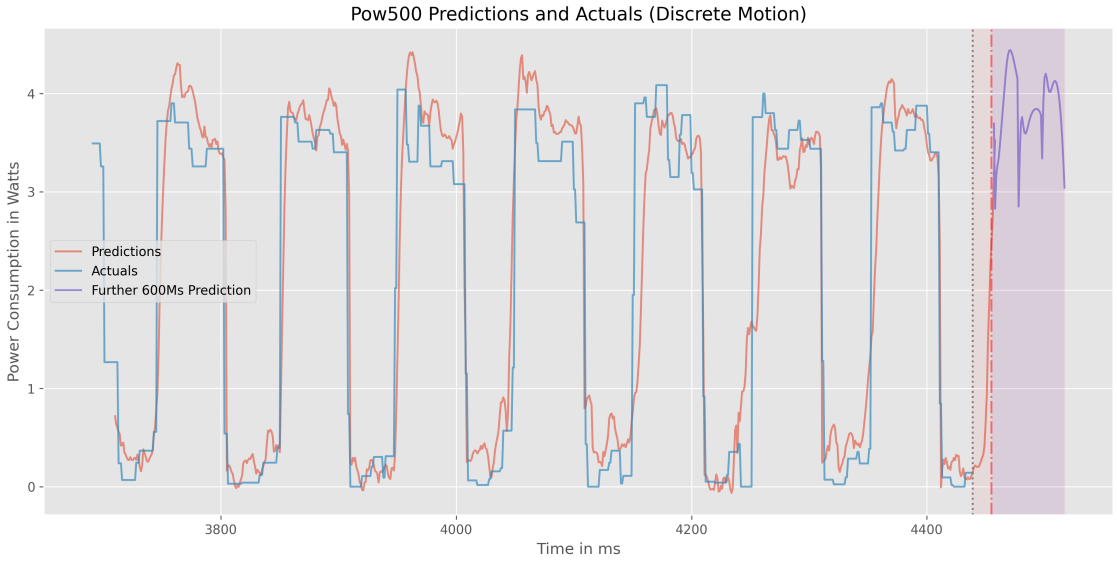
[ ]:
```
print_act_and_pred_tables_withFurther(re_yhat2,re_y_test2,20, -750,-1)
```

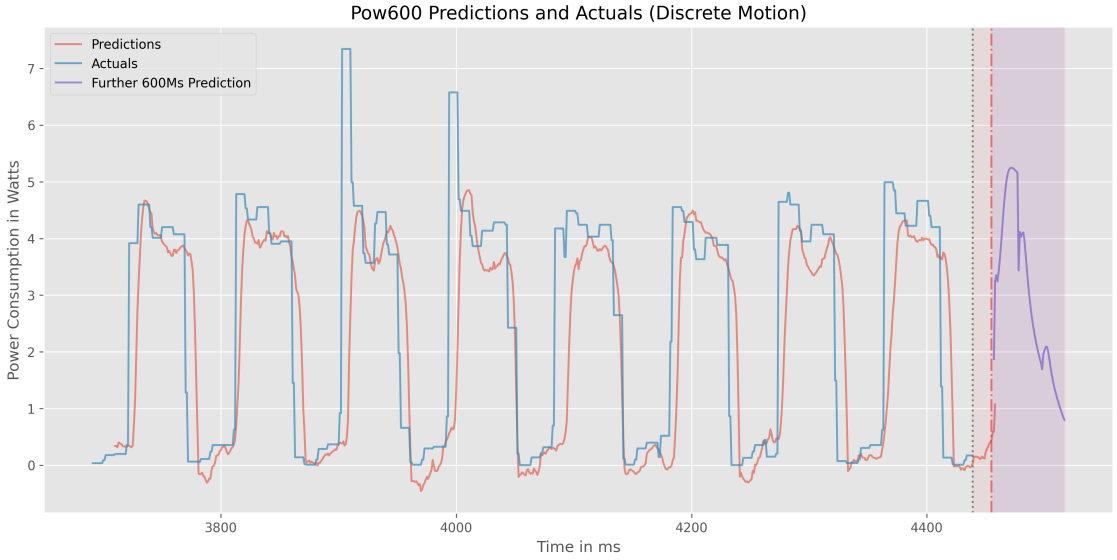


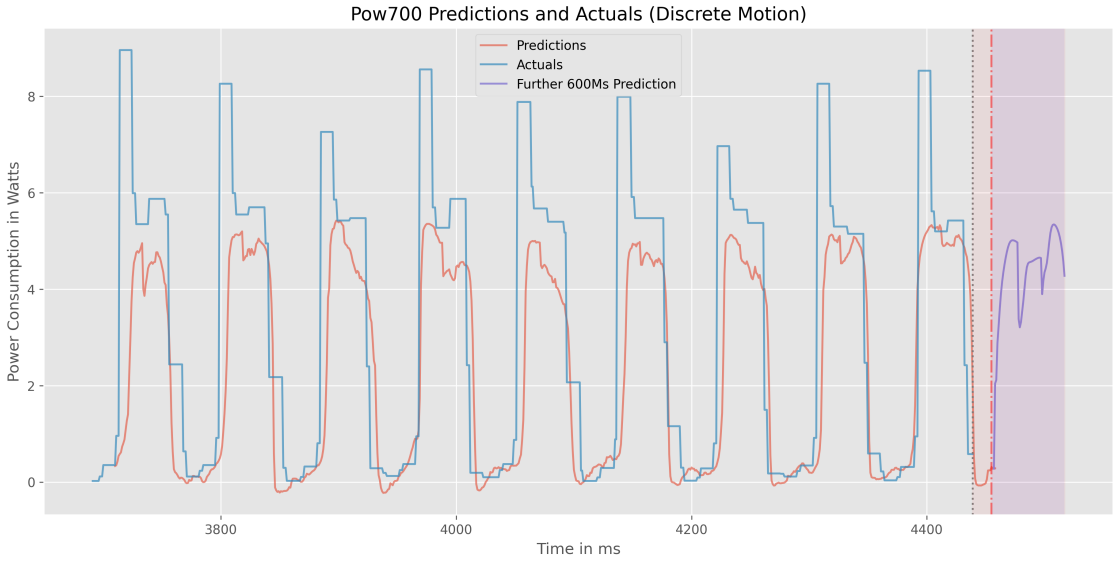


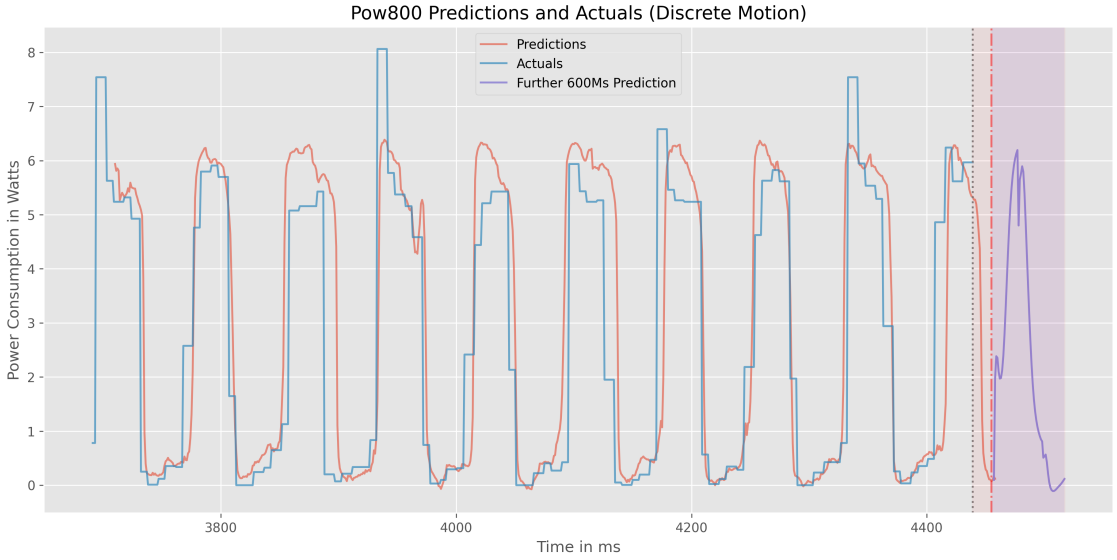

Pow300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3800
4000
4200
4400

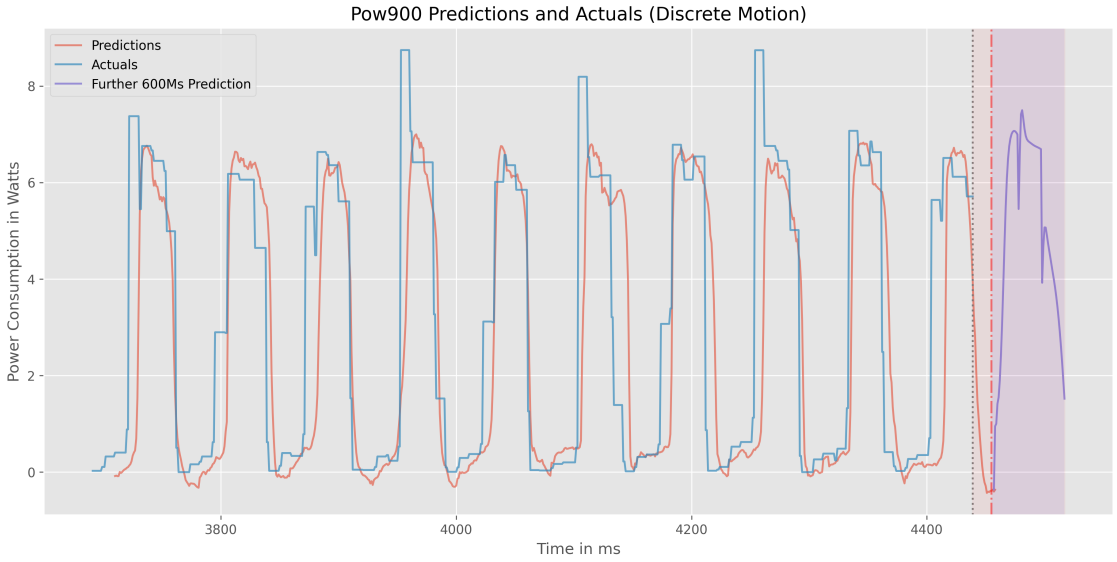

Pow400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
4
3
2
1
0
3800
4000
4200
4400

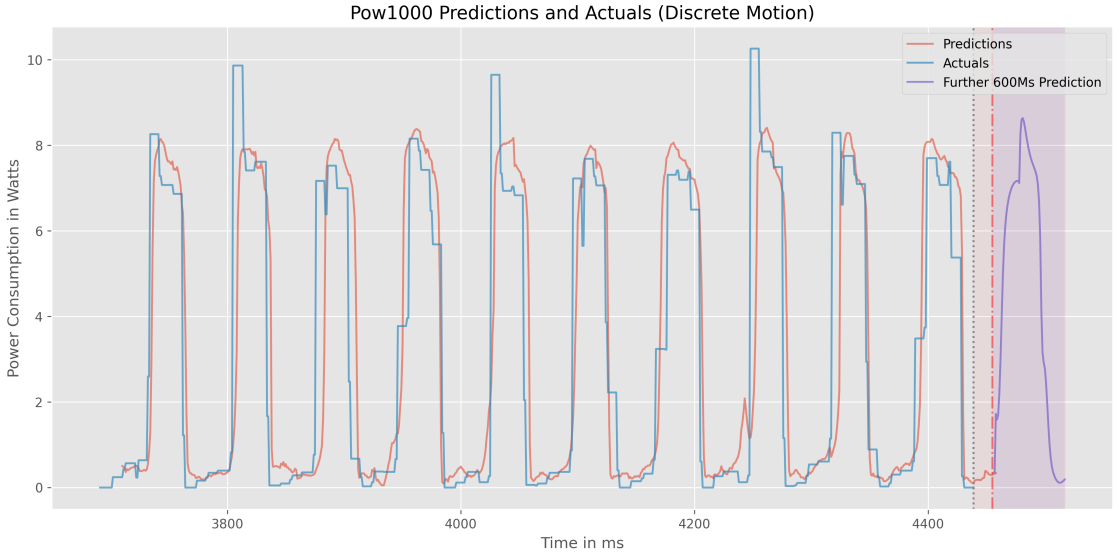

Pow500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0
1
2
3
4
3800
4000
4200
4400

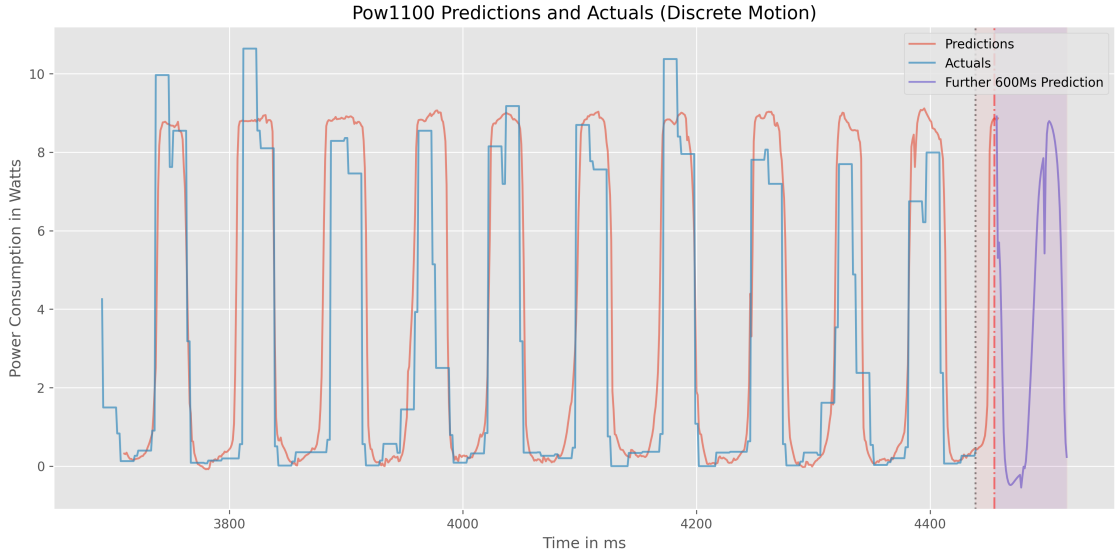

Pow600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
7
3800
4000
4200
4400

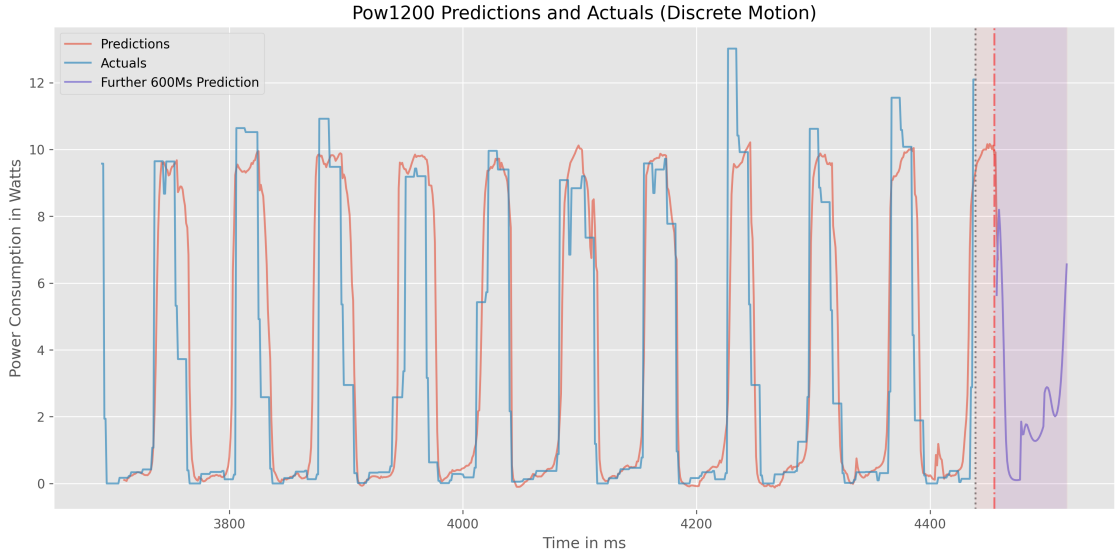

Pow700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0
2
4
6
8
3800
4000
4200
4400
Time in ms

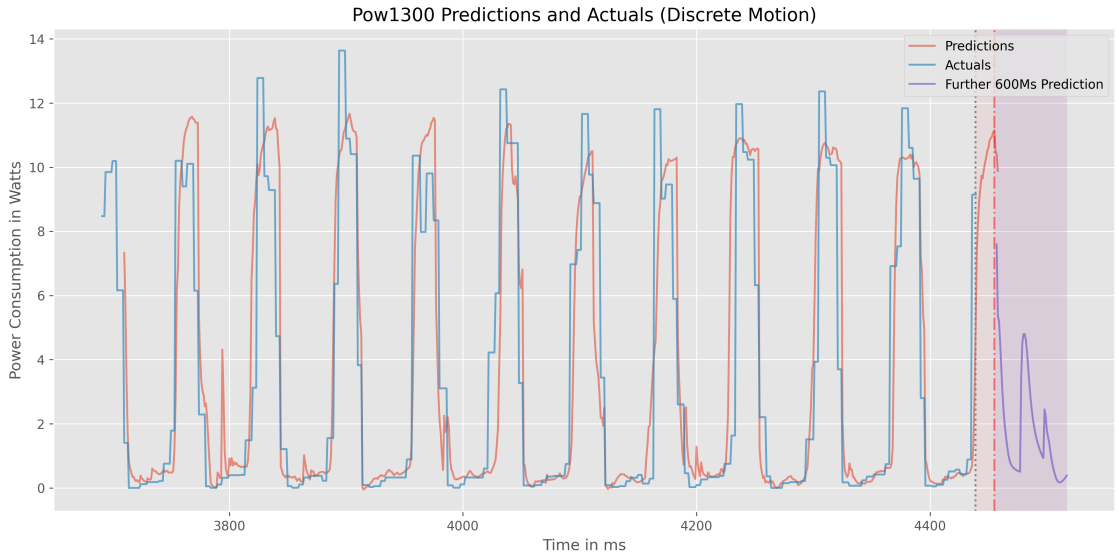

Pow800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0
1
2
3
4
5
6
7
8
3800
4000
4200
4400
Time in ms

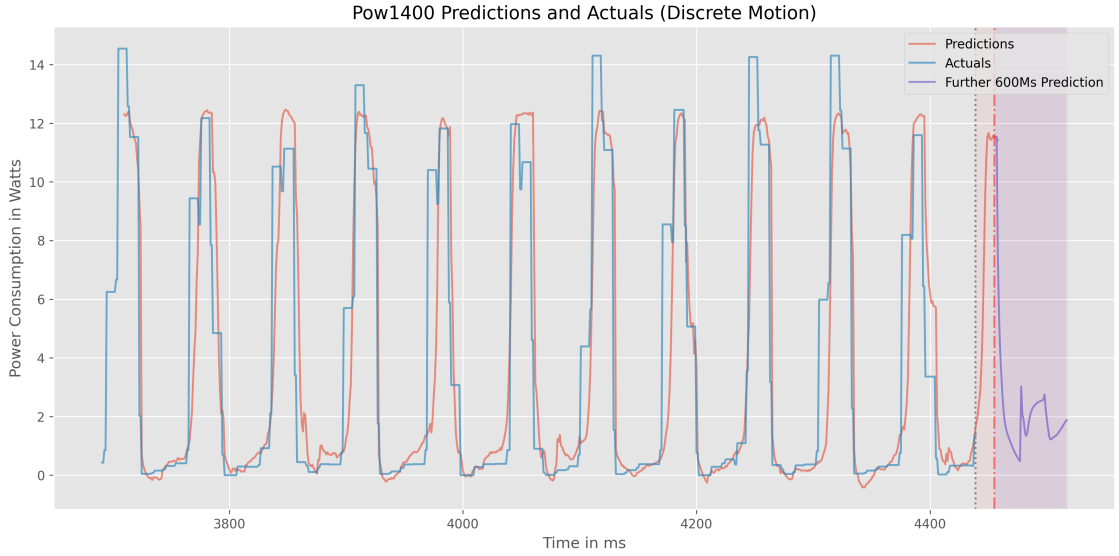

Pow900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0
2
4
6
8
3800
4000
4200
4400
Time in ms

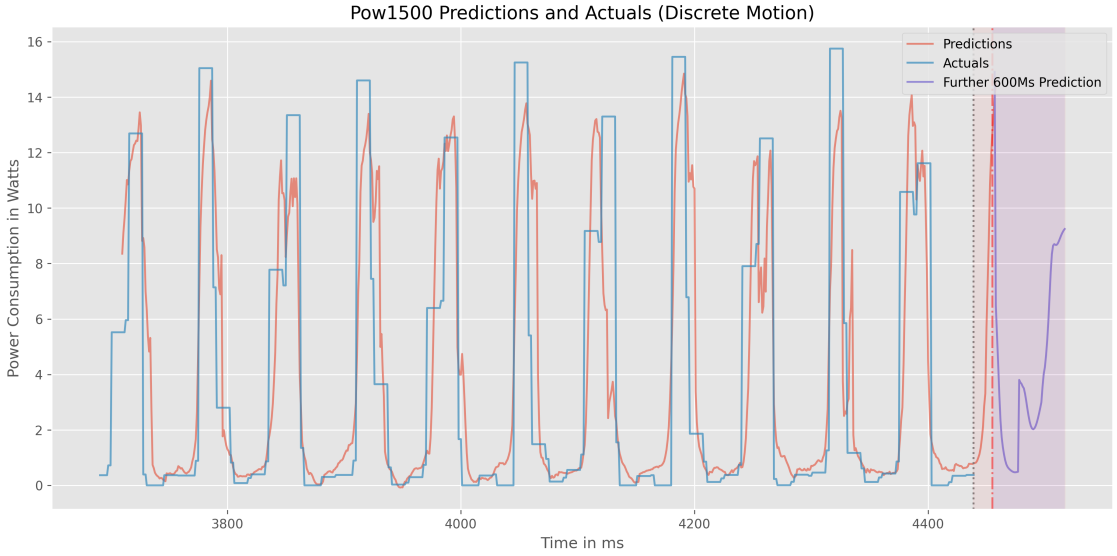

Pow1000 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0
2
4
6
8
10
3800
4000
4200
4400
Time in ms

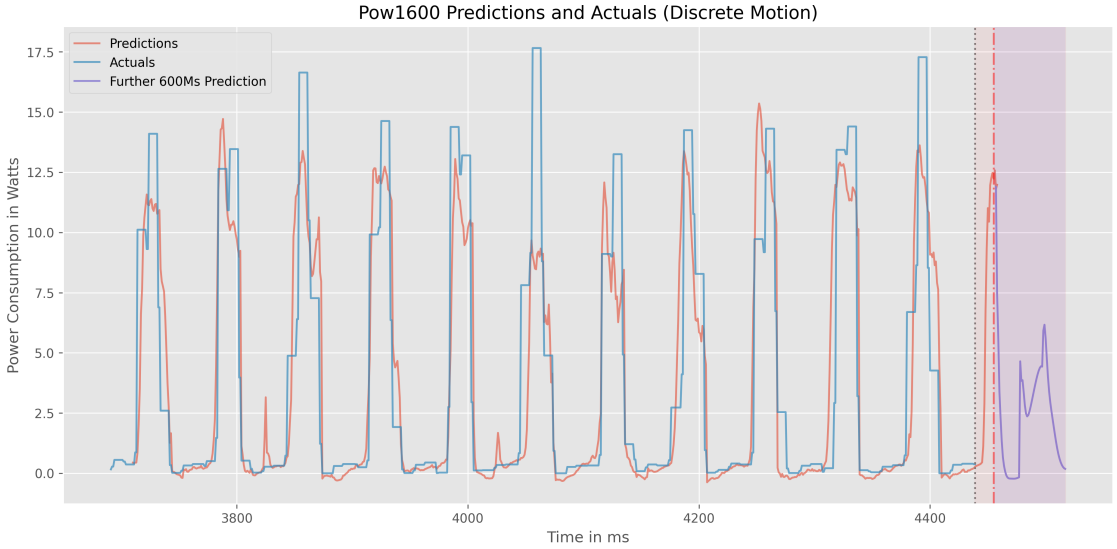

Pow1100 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
3800
4000
4200
4400

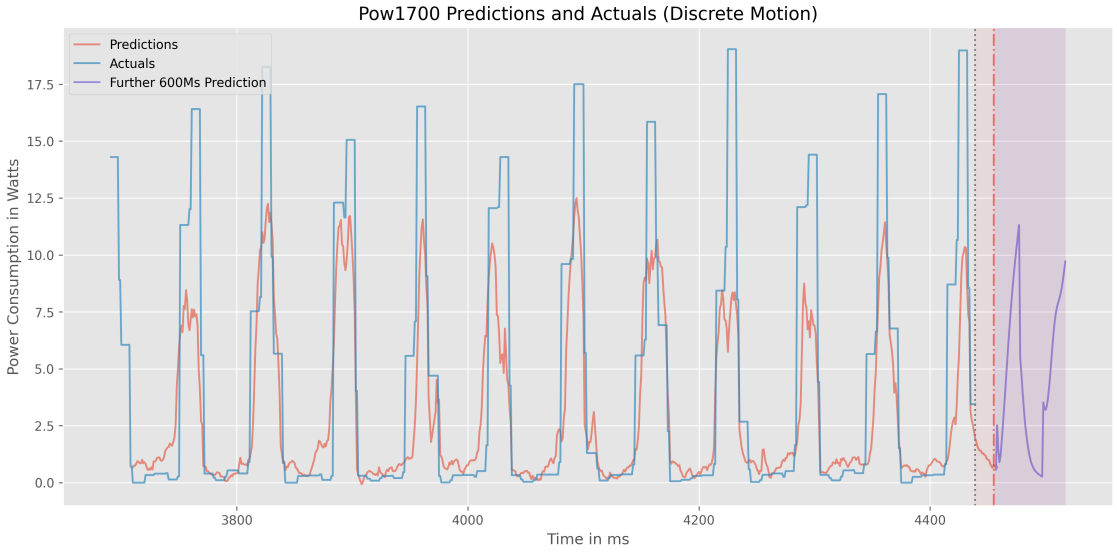

Pow1200 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
3800
4000
4200
4400

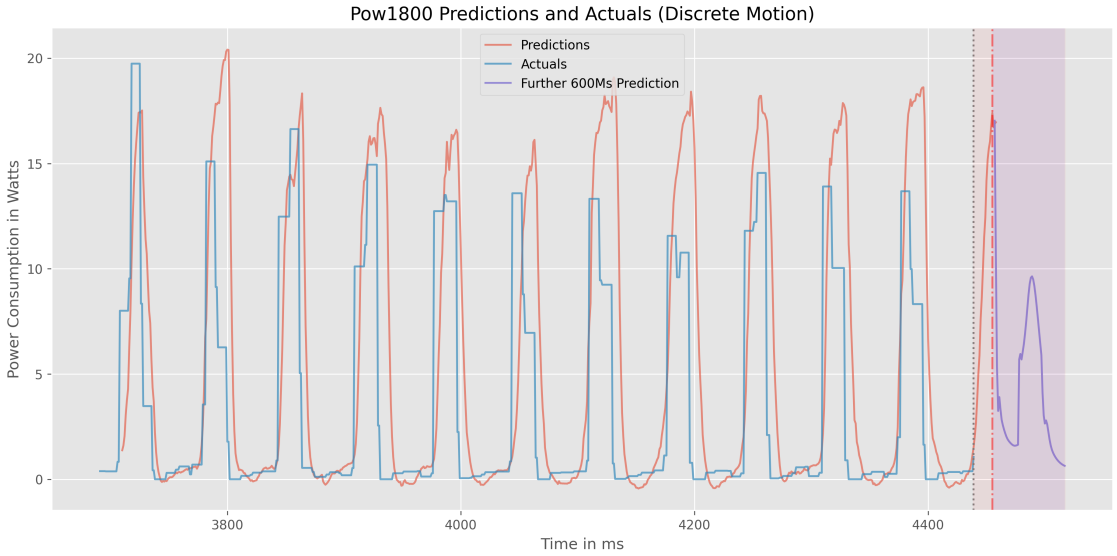

Pow1300 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
3800
4000
4200
4400

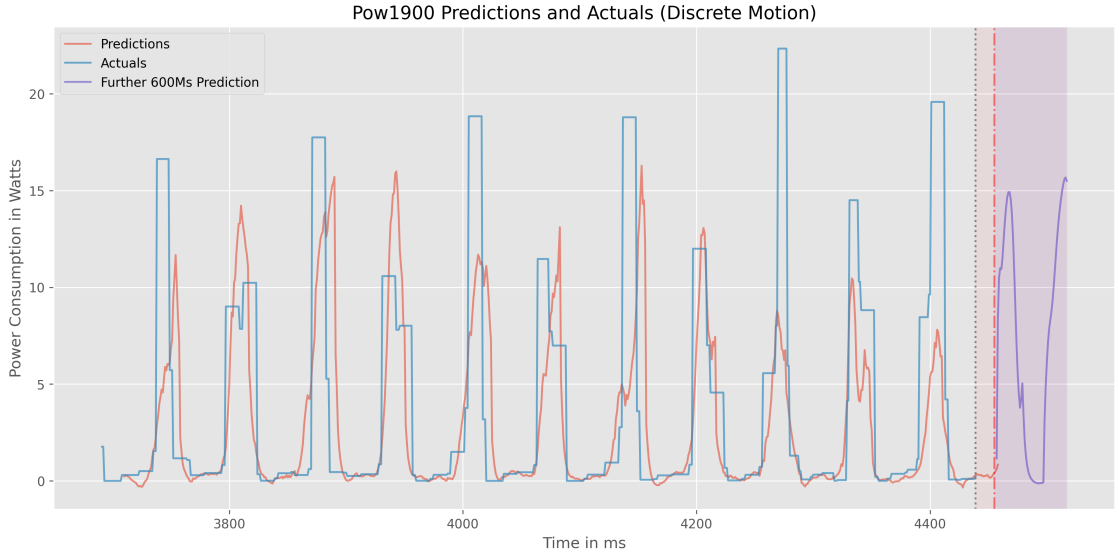

Pow1400 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
3800
4000
4200
4400

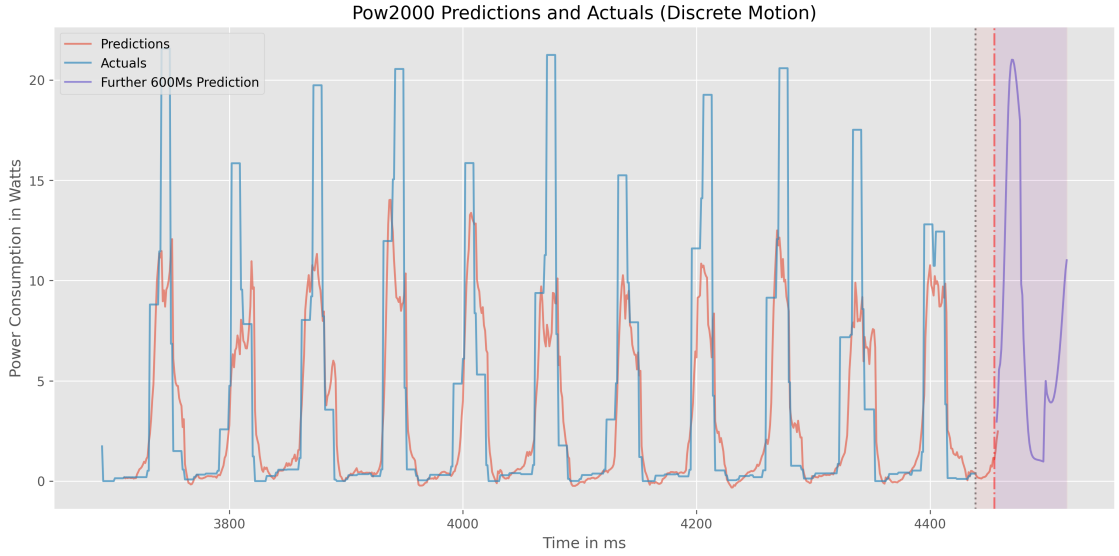

Pow1500 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
16
3800
4000
4200
4400


Pow1600 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
Time in ms
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3800
4000
4200
4400


Pow1700 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3800
4000
4200
4400
Time in ms


Pow1800 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0
5
10
15
20
3800
4000
4200
4400
Time in ms


Pow1900 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0
5
10
15
20
3800
4000
4200
4400
Time in ms


Pow2000 Predictions and Actuals (Discrete Motion)
Predictions
Actuals
Further 600Ms Prediction
Power Consumption in Watts
0
5
10
15
20
3800
4000
4200
4400
Time in ms

[ ]: