FINAL_gru_continuous_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 Resulution

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

Read In the different Datasets

[3]:
```
df = pd.read_excel('ContinuousMotion_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	0.914997	1.489418	2.085885	2.704043	3.316586
std	0.115249	0.203847	0.334017	0.514336	0.724688
min	0.000000	0.054000	0.000000	0.000000	0.000000
25%	0.858000	1.422000	2.004000	2.610000	3.258000
50%	0.900000	1.485000	2.100000	2.760000	3.420000
75%	0.960000	1.575000	2.236000	2.925000	3.618000
max	1.316000	1.960000	2.990000	4.335000	5.520000

	Pow_600	Pow_700	Pow_800	Pow_900	Pow_1000
count	45000.000000	45000.000000	45000.000000	45000.000000	45000.000000
mean	3.971865	4.581573	5.268829	5.965631	6.633125
std	0.941374	1.204082	1.495026	1.817265	2.229889
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	3.948000	4.608000	5.346000	6.120000	6.962999
50%	4.137000	4.824000	5.616000	6.450000	7.326000
75%	4.347000	5.064000	5.908000	6.789001	7.718000
max	6.648000	7.398000	8.460000	9.537000	10.512000

	Pow_1100	Pow_1200	Pow_1300	Pow_1400	Pow_1500
count	45000.000000	45000.000000	45000.000000	45000.000000	45000.000000
mean	7.357132	8.101215	8.847017	9.569282	10.302015
std	2.616157	3.043377	3.524884	4.005067	4.572351
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	7.743999	8.502001	9.245000	9.917999	9.486000
50%	8.288000	9.240000	10.248001	11.340000	12.495000
75%	8.695000	9.760000	10.836000	11.914000	13.082999
max	11.271000	12.474001	14.344000	15.745001	16.650000

	Pow_1600	Pow_1700	Pow_1800	Pow_1900	Pow_2000
count	45000.000000	45000.000000	45000.000000	45000.000000	45000.000000
mean	10.961324	11.728856	12.532663	13.285962	13.883382
std	5.205278	5.829242	6.454288	7.140127	7.816016
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	8.320000	7.708000	7.134000	6.931000	6.480000
50%	13.624001	15.007998	16.298000	17.732000	18.910000
75%	14.310000	15.679999	17.109999	18.538000	19.997999
max	17.712000	19.207998	20.940001	22.528000	23.517000

1.2 Defining Functions

Split the data accoring 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 betweet 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

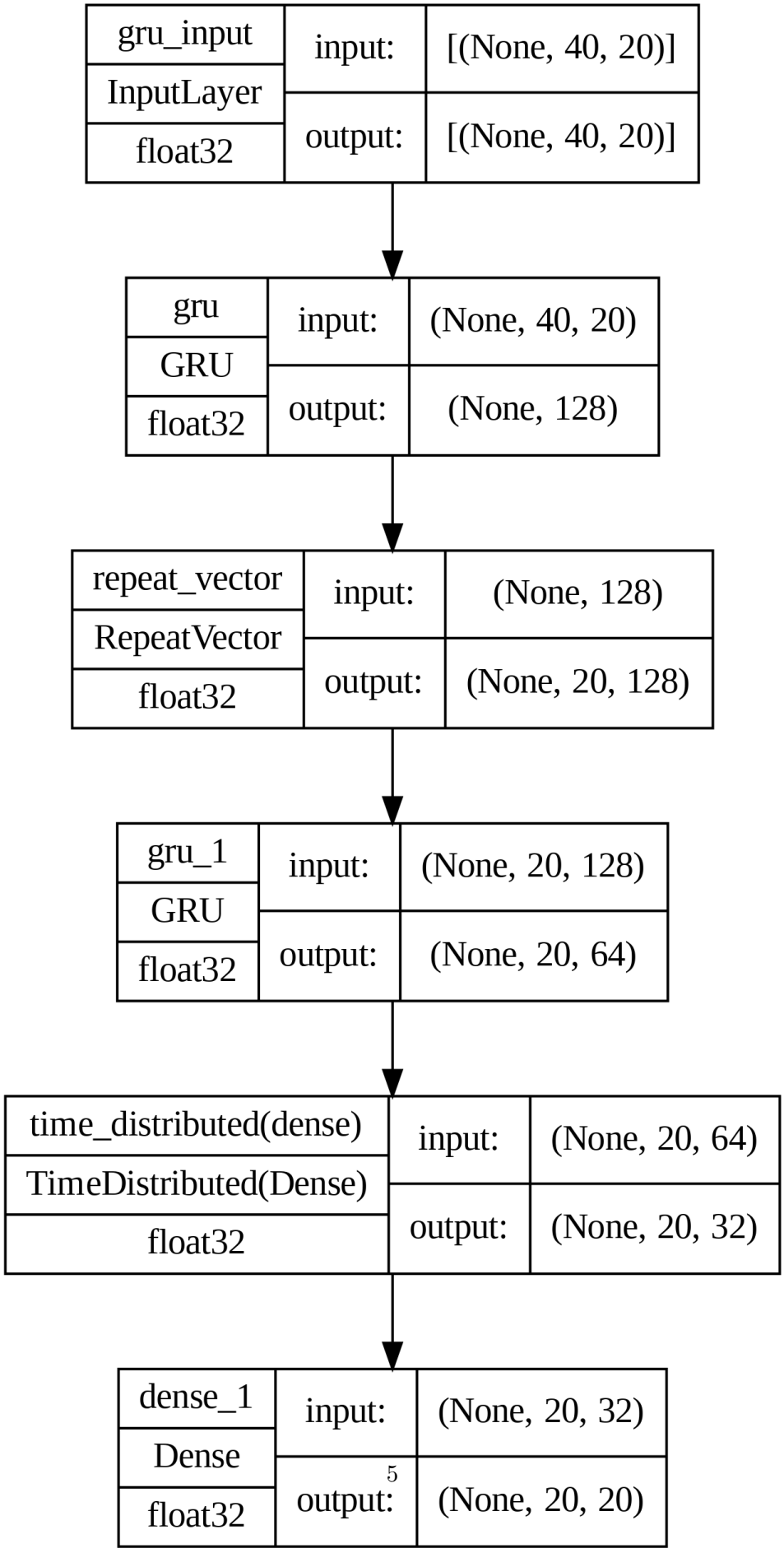
1.4.1 Model Summary

[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
InputLayer
float32
input:
[(None, 40, 20)]
output:
[(None, 40, 20)]
gru
GRU
float32
input:
(None, 40, 20)
output:
(None, 128)
repeat_vector
RepeatVector
float32
input:
(None, 128)
output:
(None, 20, 128)
gru_1
GRU
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 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])
```

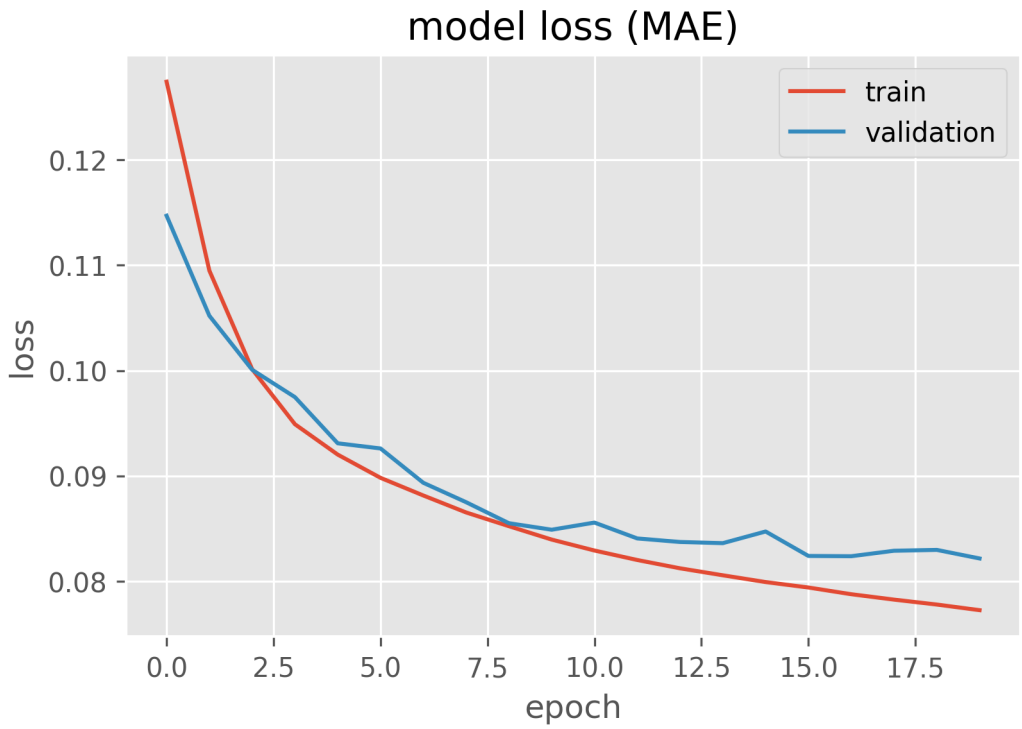
```
Epoch 1/20
1124/1124 [==============================] - 58s 48ms/step - loss: 0.1274 - mse:
0.0524 - val_loss: 0.1147 - val_mse: 0.0427
Epoch 2/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.1095 - mse:
0.0395 - val_loss: 0.1052 - val_mse: 0.0365
Epoch 3/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.1001 - mse:
0.0344 - val_loss: 0.1001 - val_mse: 0.0334
Epoch 4/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.0949 - mse:
0.0318 - val_loss: 0.0975 - val_mse: 0.0320
Epoch 5/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0920 - mse:
0.0305 - val_loss: 0.0931 - val_mse: 0.0295
Epoch 6/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.0898 - mse:
0.0295 - val_loss: 0.0926 - val_mse: 0.0291
Epoch 7/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.0881 - mse:
0.0287 - val_loss: 0.0893 - val_mse: 0.0285
Epoch 8/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0865 - mse:
0.0281 - val_loss: 0.0875 - val_mse: 0.0280
Epoch 9/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0852 - mse:
0.0276 - val_loss: 0.0855 - val_mse: 0.0270
Epoch 10/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0839 - mse:
0.0271 - val_loss: 0.0849 - val_mse: 0.0264
Epoch 11/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.0829 - mse:
0.0267 - val_loss: 0.0856 - val_mse: 0.0272
Epoch 12/20
```

```
1124/1124 [==============================] - 53s 47ms/step - loss: 0.0820 - mse:
0.0263 - val_loss: 0.0841 - val_mse: 0.0266
Epoch 13/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.0812 - mse:
0.0260 - val_loss: 0.0837 - val_mse: 0.0263
Epoch 14/20
1124/1124 [==============================] - 55s 49ms/step - loss: 0.0806 - mse:
0.0258 - val_loss: 0.0836 - val_mse: 0.0266
Epoch 15/20
1124/1124 [==============================] - 53s 47ms/step - loss: 0.0799 - mse:
0.0255 - val_loss: 0.0847 - val_mse: 0.0268
Epoch 16/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0794 - mse:
0.0252 - val_loss: 0.0824 - val_mse: 0.0259
Epoch 17/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0788 - mse:
0.0249 - val_loss: 0.0824 - val_mse: 0.0262
Epoch 18/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0782 - mse:
0.0246 - val_loss: 0.0829 - val_mse: 0.0262
Epoch 19/20
1124/1124 [==============================] - 55s 49ms/step - loss: 0.0778 - mse:
0.0244 - val_loss: 0.0830 - val_mse: 0.0268
Epoch 20/20
1124/1124 [==============================] - 54s 48ms/step - loss: 0.0772 - mse:
0.0242 - val_loss: 0.0822 - val_mse: 0.0262
```

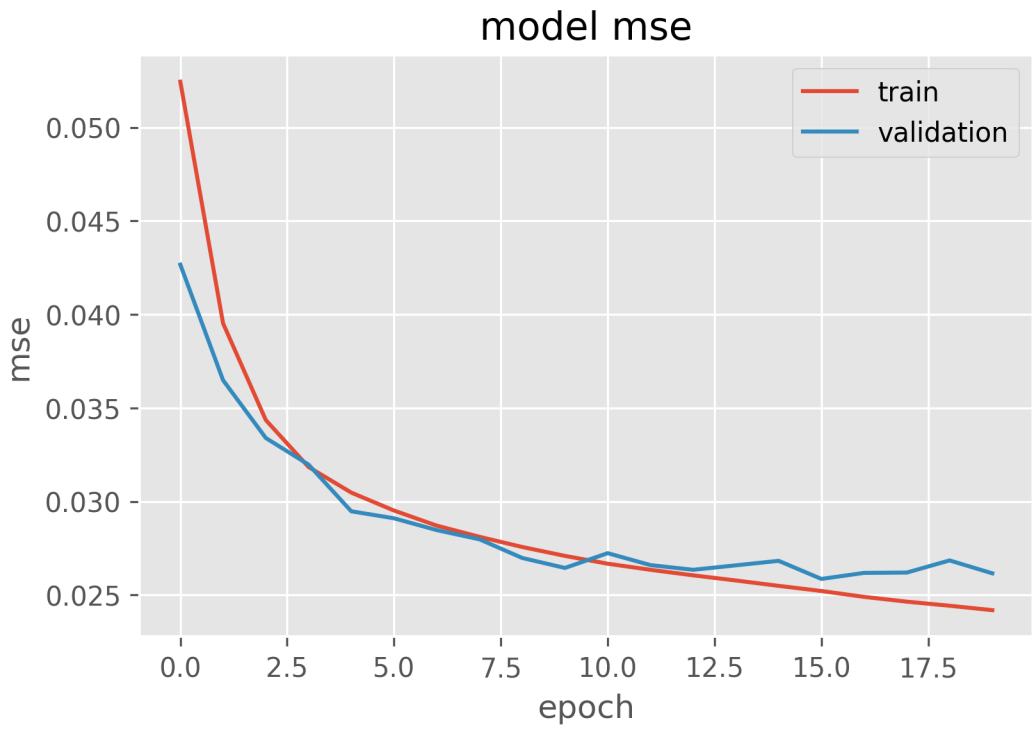
1.4.3 Model History

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

[21]:
```
model.evaluate(X_test, y_test)
```

```
139/139 [==============================] - 2s 14ms/step - loss: 0.0808 - mse:
0.0254
```

[21]:
```
[0.08079581707715988, 0.02538587711751461]
```

1.4.4 Predicting GRU

Predict GRU Model

[22]:
```
yhat = model.predict(X_test)
```

```
139/139 [==============================] - 2s 14ms/step
```

Saving the shape of y_test

[23]:
```
test_shape = y_test.shape
```

Print RMSE Plots from t+1 to t+20

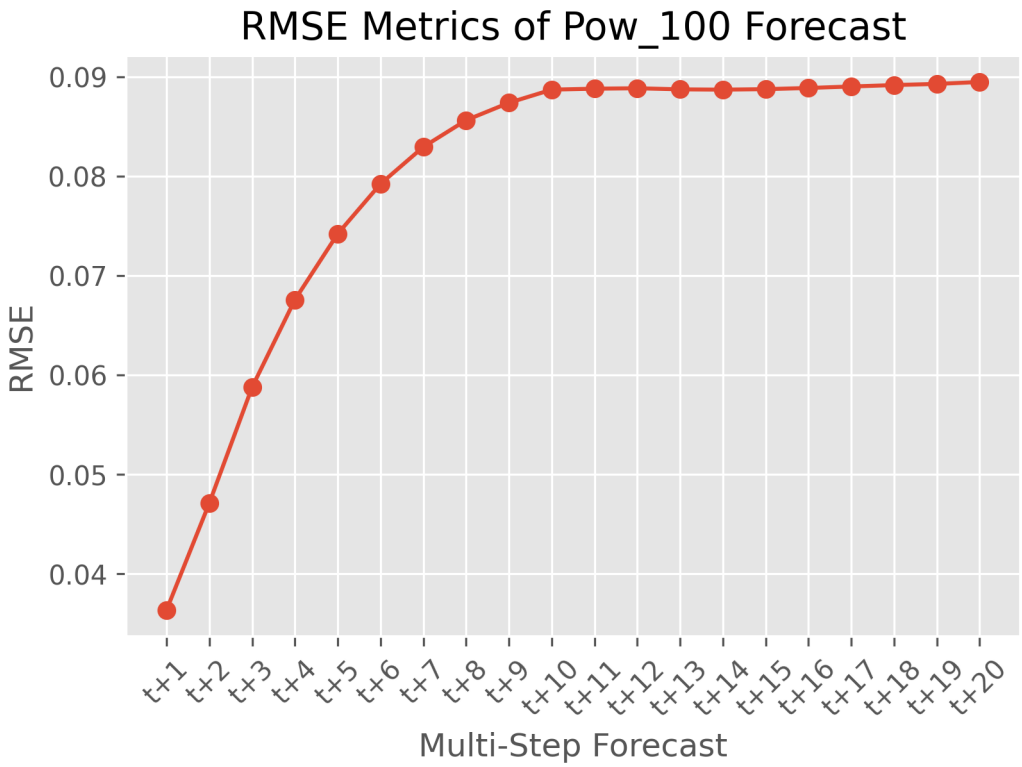
[24]:
```
from math import sqrt
# evaluate forecasts against expected values
```

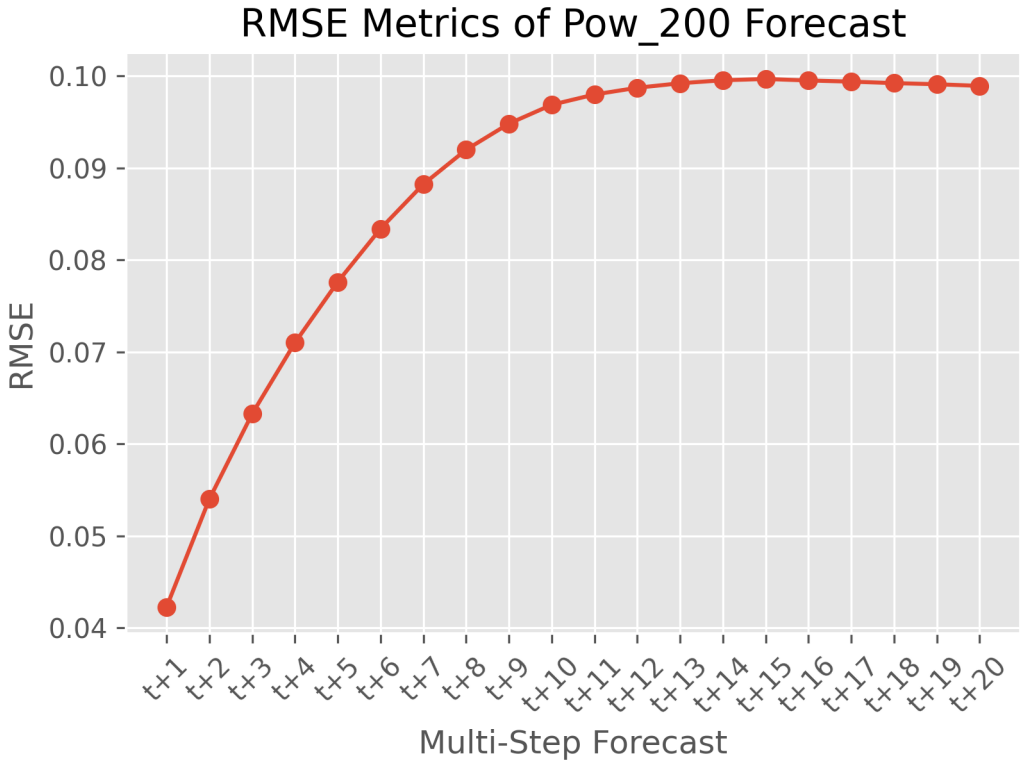
```
def evaluate_forecasts(actual, predicted):
    scores_pow = results = np.zeros((20, 20))
    scores = []
 # calculate an RMSE score for each prediction period
    for i in range(actual.shape[1]):
      for t in range(actual.shape[2]):
       # calculate mse
       scores_pow[i, t] = sqrt(mean_squared_error(actual[:,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

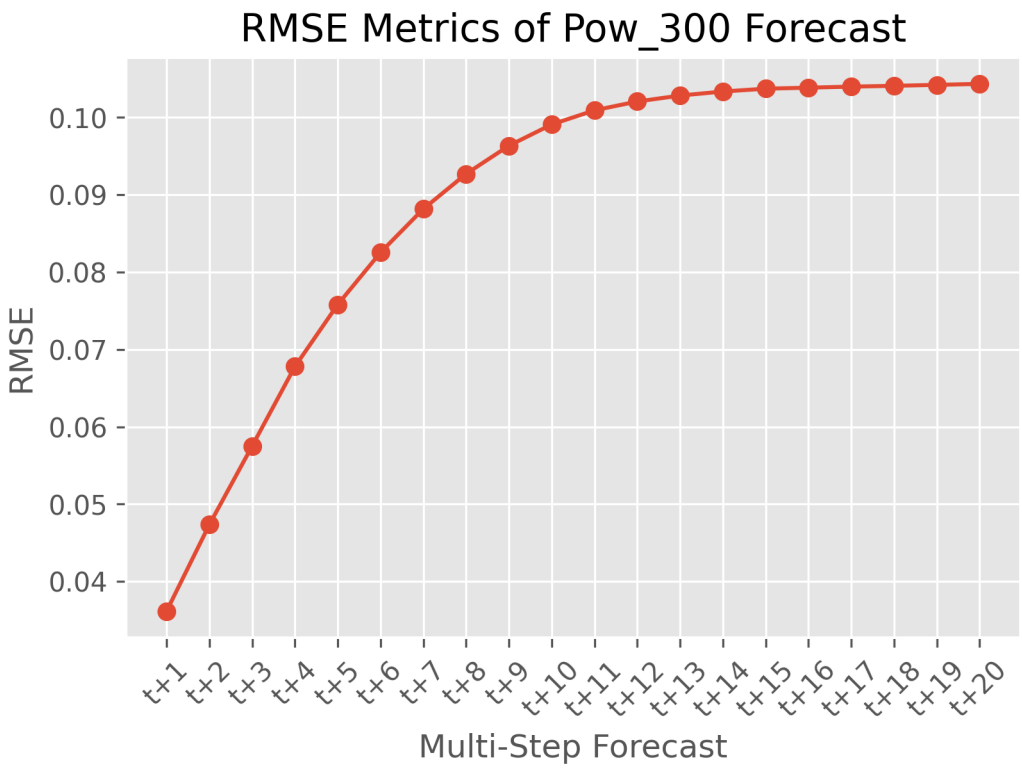
[25]:
```
def summarize_scores(name, score, scores):
    s_scores = ', '.join(['%.3f'% s for s in scores])
    print('%s: [%.3f] %s' % (name, score, s_scores))
```

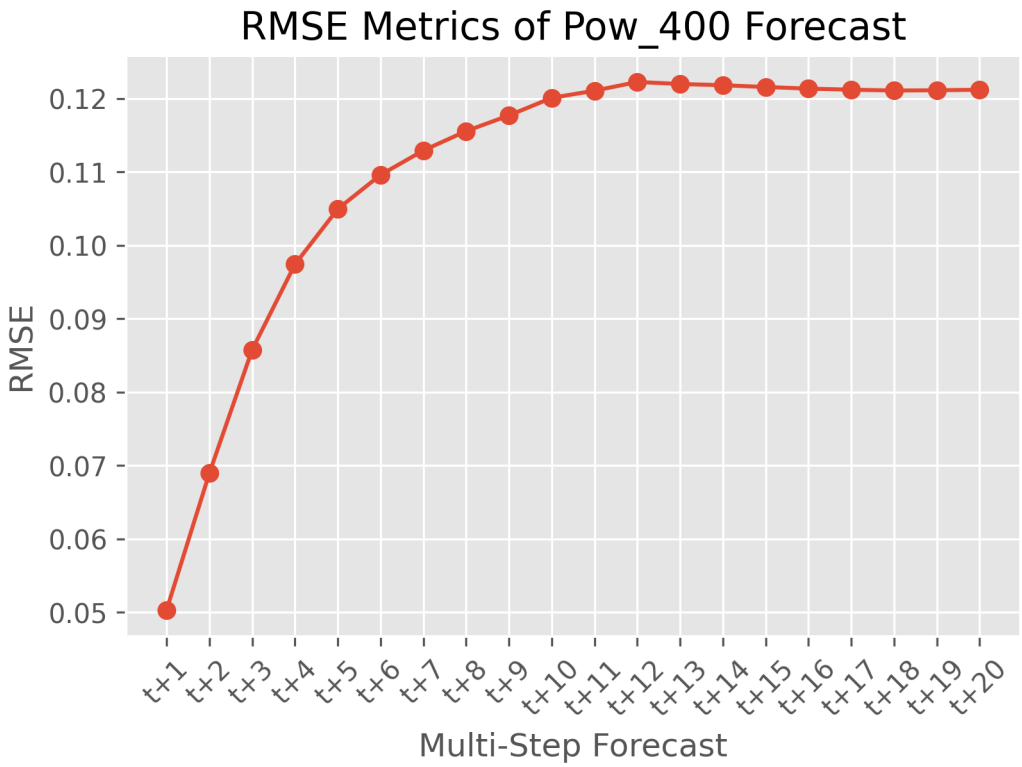
[26]:
```
overall_RMSE, all_RMSEs = evaluate_forecasts(y_test, yhat)
```

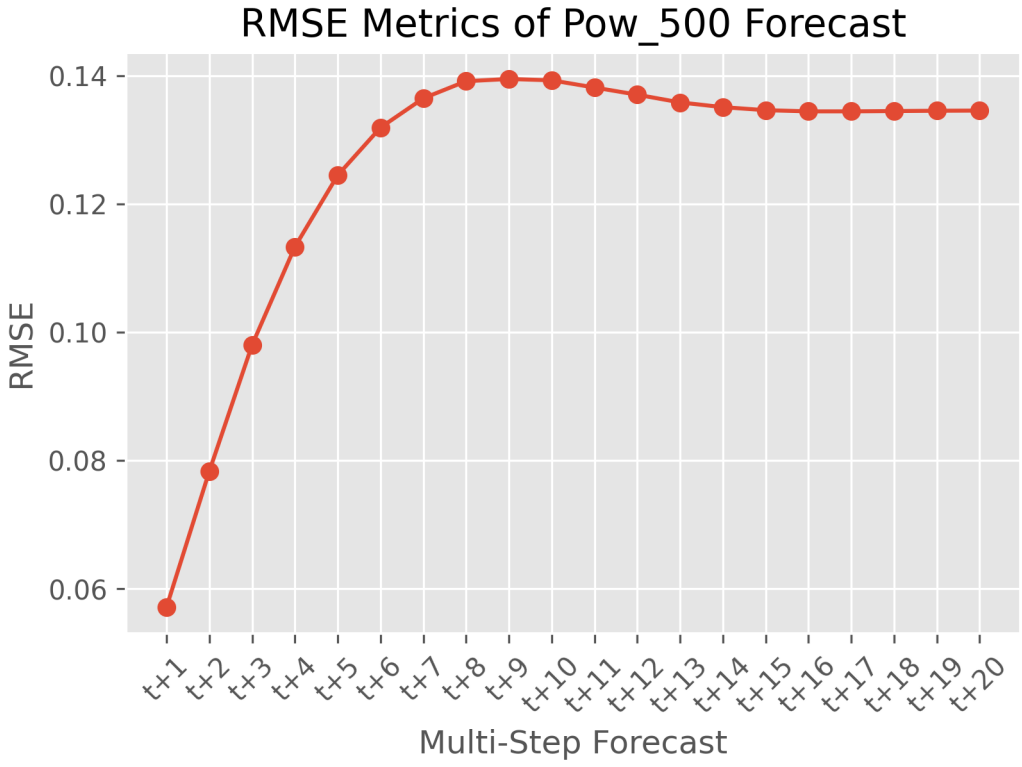
[27]:
```
forecast = []
for i in range(1,21):
    title = f't+{i}'
    forecast.append(title)
for i in range(20):
  plt.style.use("ggplot")
  plt.plot(forecast, all_RMSEs[i], marker='o', label='gru')
  plt.xticks(rotation=45)
  plt.title('RMSE Metrics of Pow_' +str(100*(i+1))+ ' Forecast')
  plt.xlabel('Multi-Step Forecast')
  plt.ylabel('RMSE')
  plt.show()
```

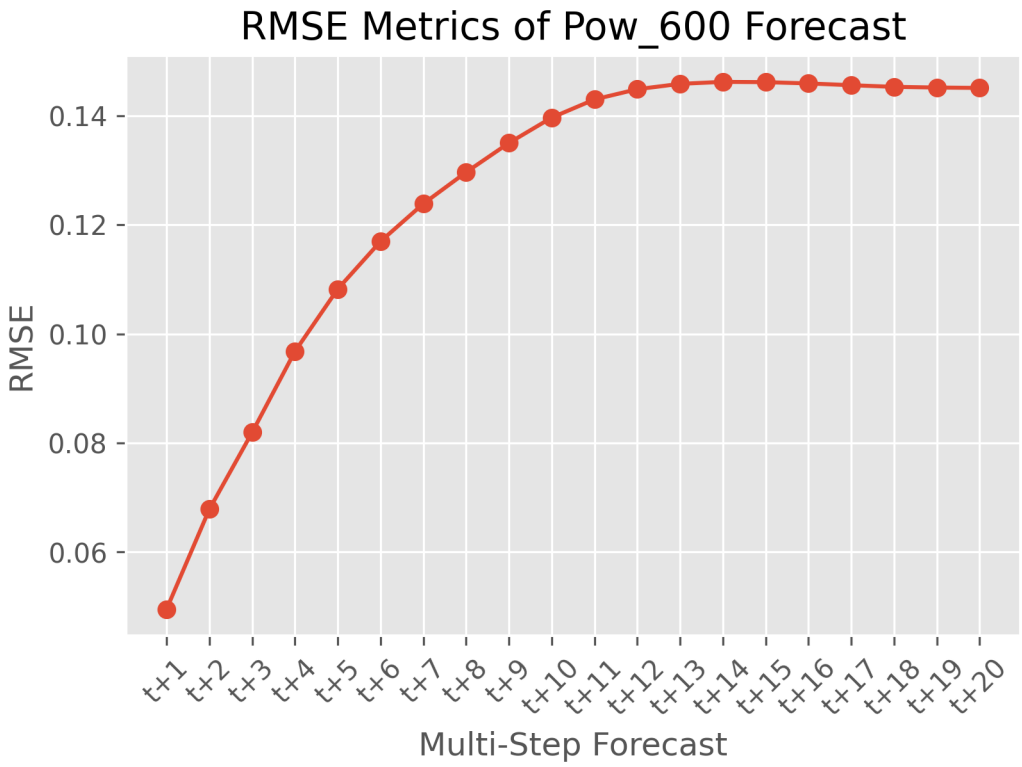

RMSE Metrics of Pow_100 Forecast
RMSE
0.09
0.08
0.07
0.06
0.05
0.04
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

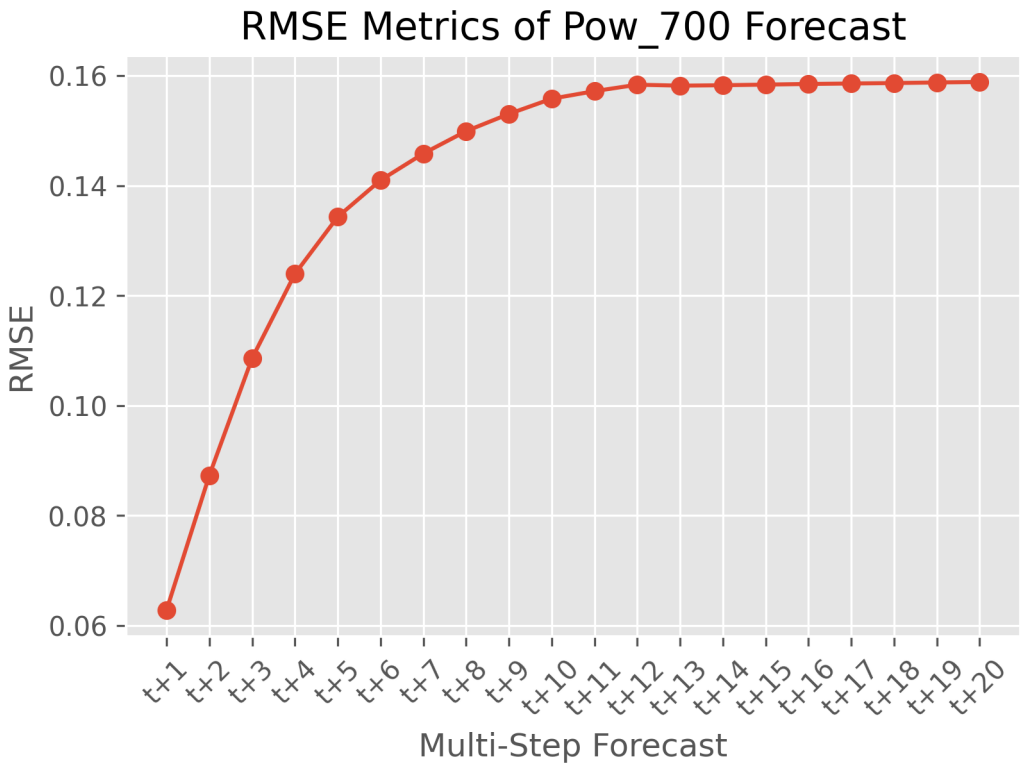

RMSE Metrics of Pow_200 Forecast
RMSE
0.10
0.09
0.08
0.07
0.06
0.05
0.04
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

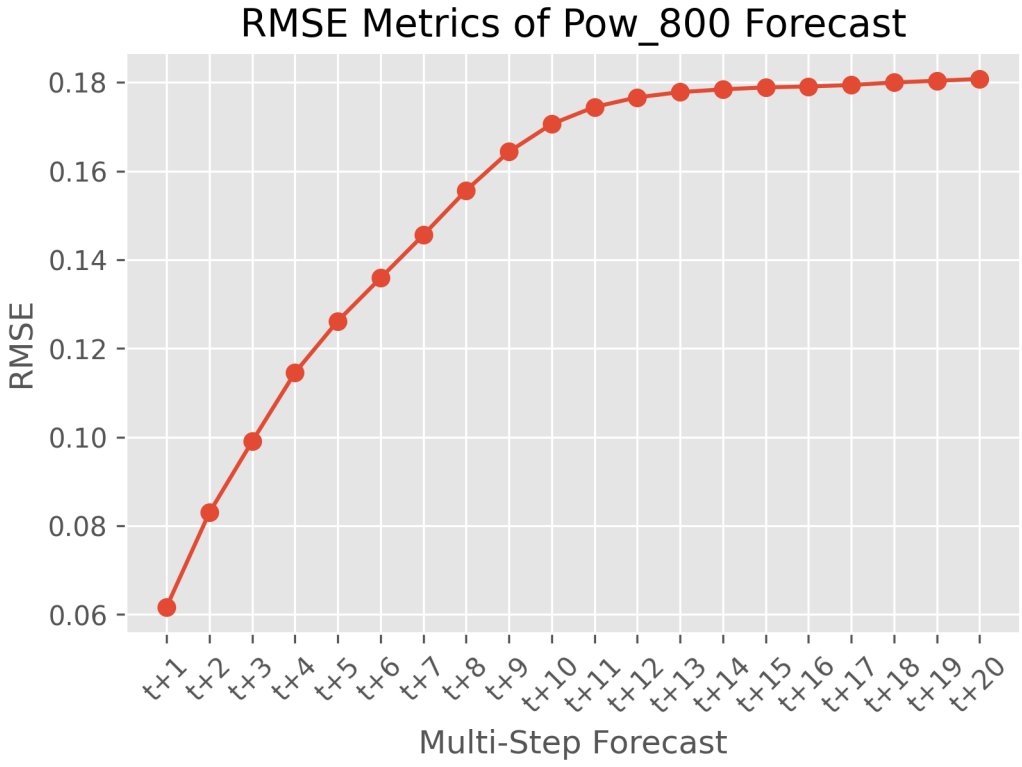

RMSE Metrics of Pow_300 Forecast
RMSE
0.10
0.09
0.08
0.07
0.06
0.05
0.04
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

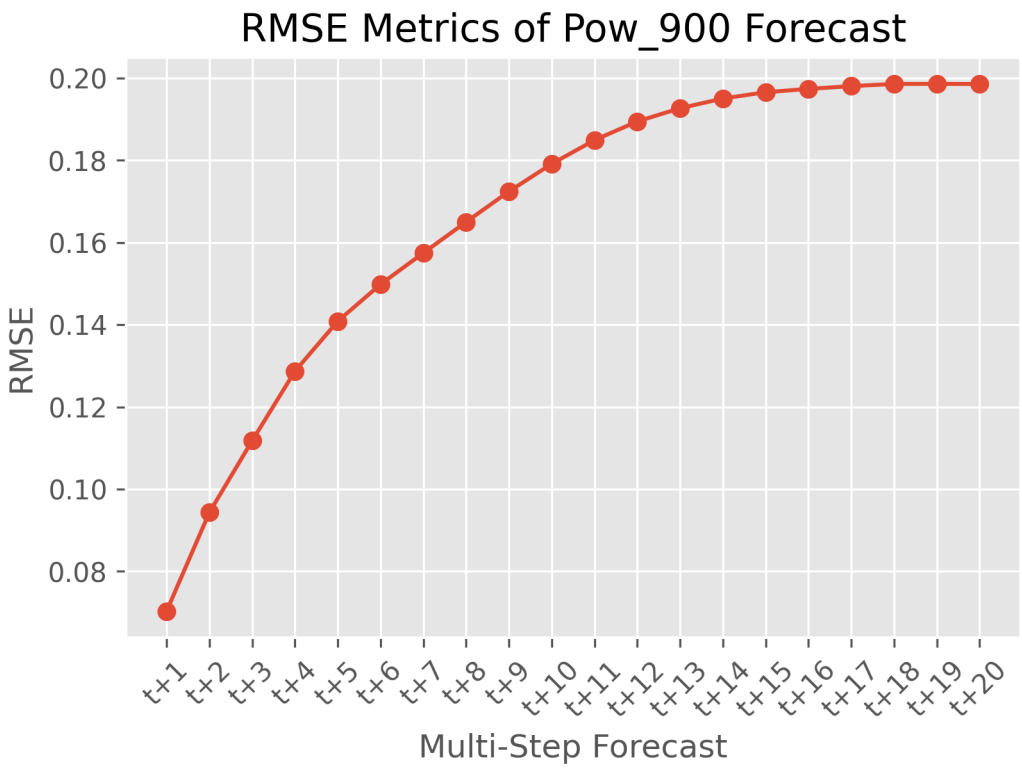

RMSE Metrics of Pow_400 Forecast
RMSE
0.12
0.11
0.10
0.09
0.08
0.07
0.06
0.05
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

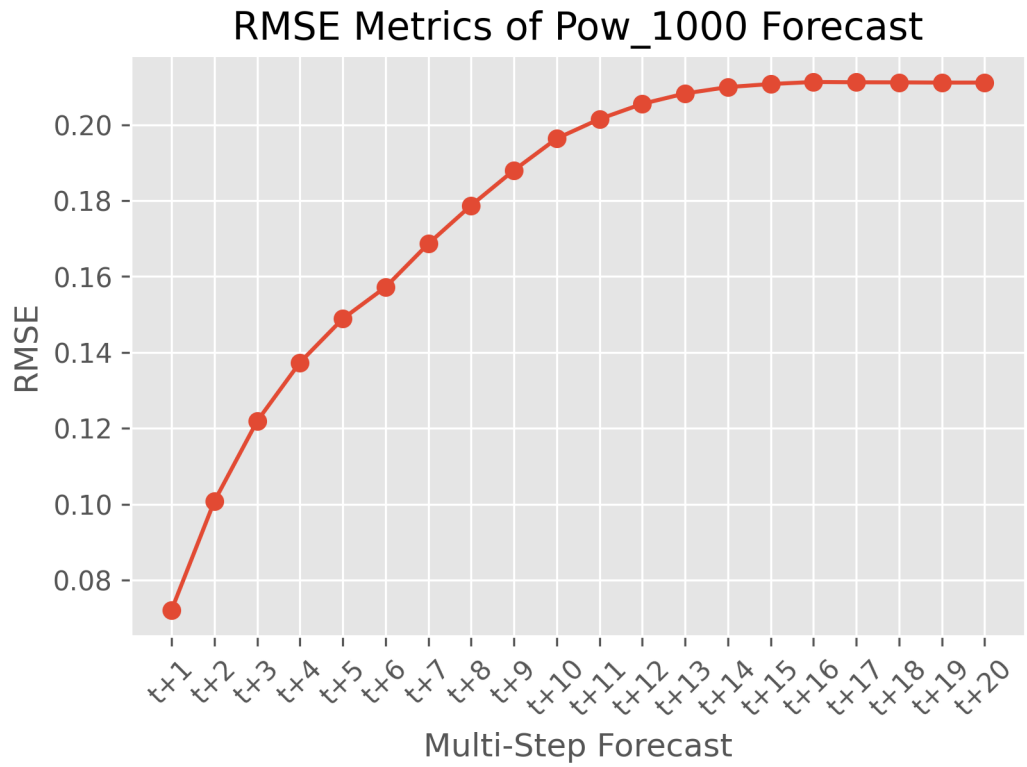

RMSE Metrics of Pow_500 Forecast
RMSE
0.14
0.12
0.10
0.08
0.06
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

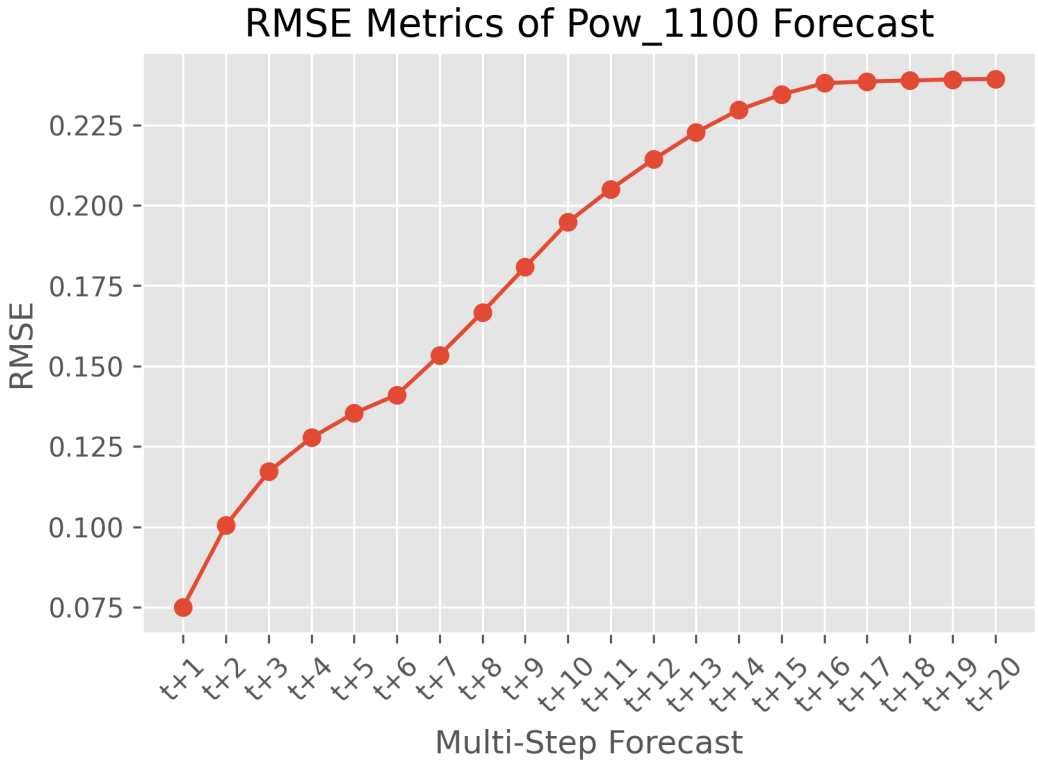

RMSE Metrics of Pow_600 Forecast
RMSE
0.14
0.12
0.10
0.08
0.06
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

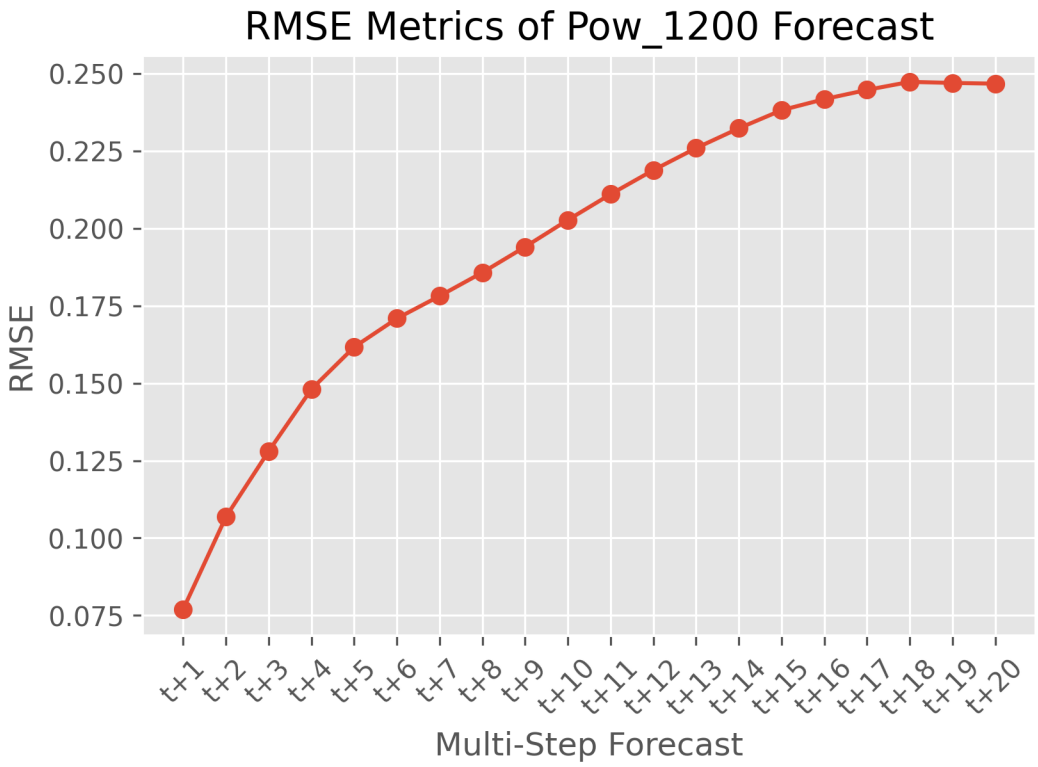

RMSE Metrics of Pow_700 Forecast
RMSE
0.16
0.14
0.12
0.10
0.08
0.06
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

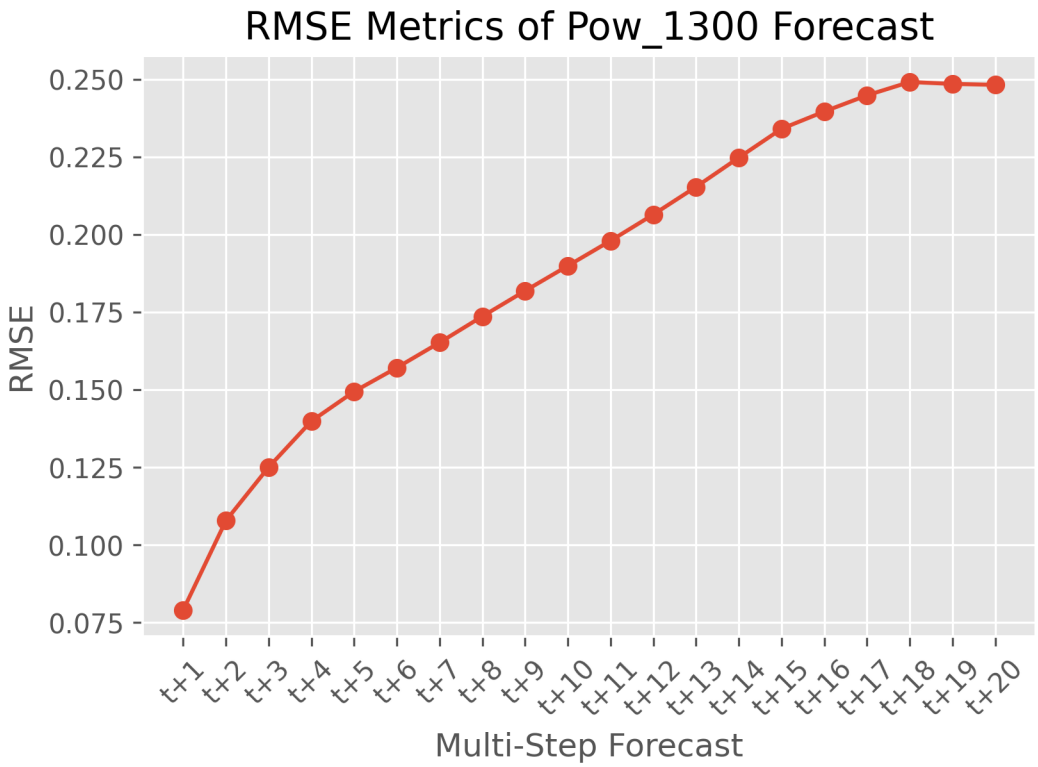

RMSE Metrics of Pow_800 Forecast
RMSE
0.18
0.16
0.14
0.12
0.10
0.08
0.06
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

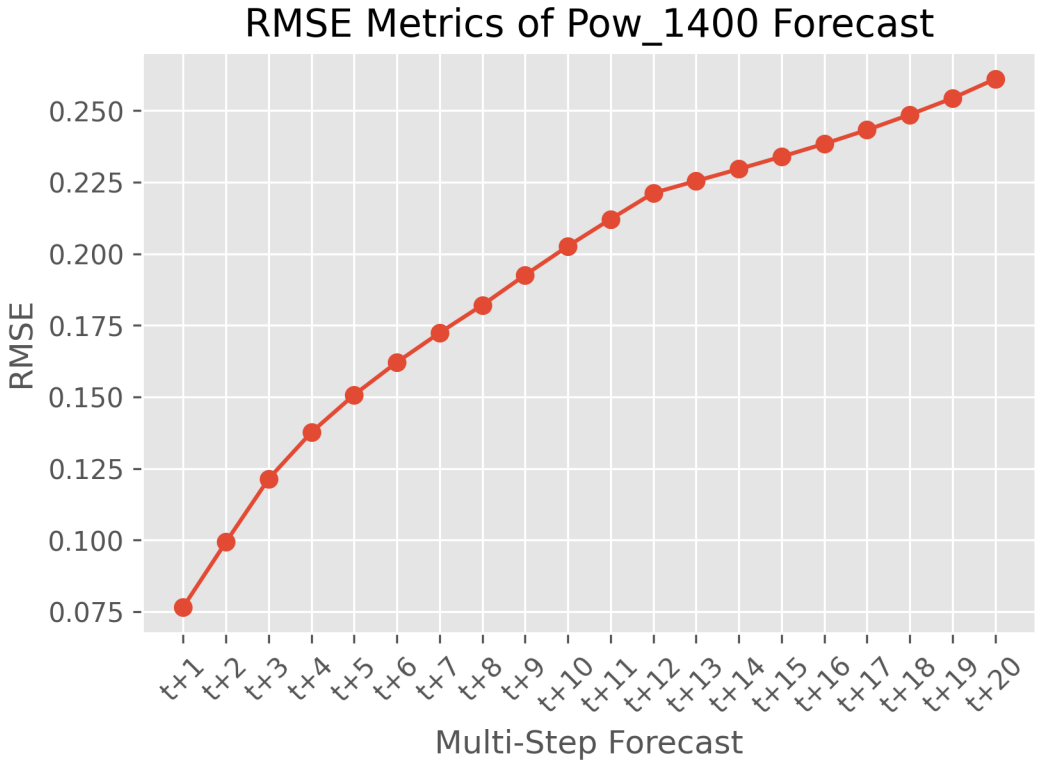

RMSE Metrics of Pow_900 Forecast
RMSE
0.20
0.18
0.16
0.14
0.12
0.10
0.08
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

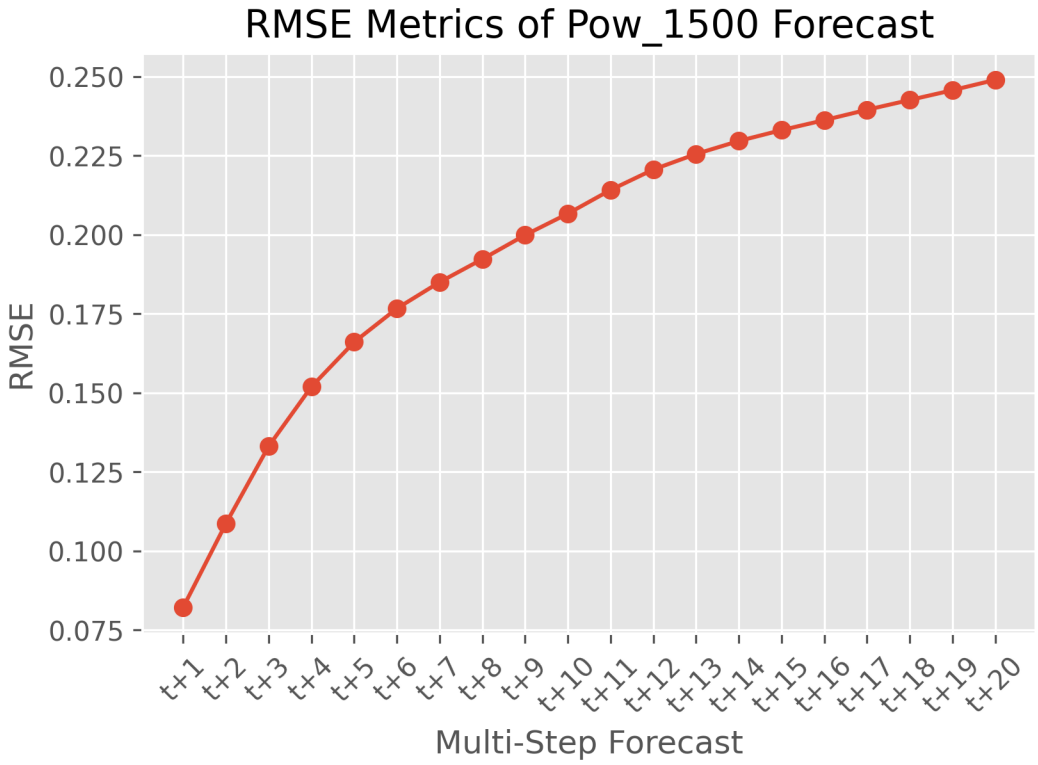

RMSE Metrics of Pow_1000 Forecast
RMSE
0.20
0.18
0.16
0.14
0.12
0.10
0.08
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

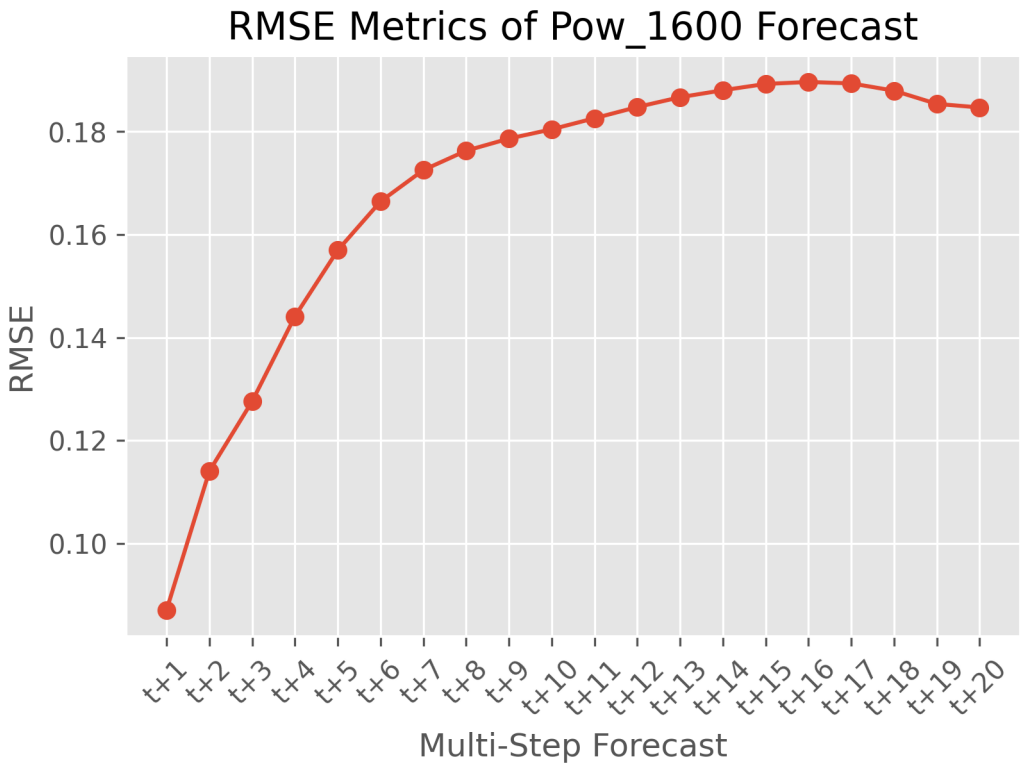

RMSE Metrics of Pow_1100 Forecast
RMSE
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1 t+2 t+3 t+4 t+5 t+6 t+7 t+8 t+9 t+10 t+11 t+12 t+13 t+14 t+15 t+16 t+17 t+18 t+19 t+20
Multi-Step Forecast

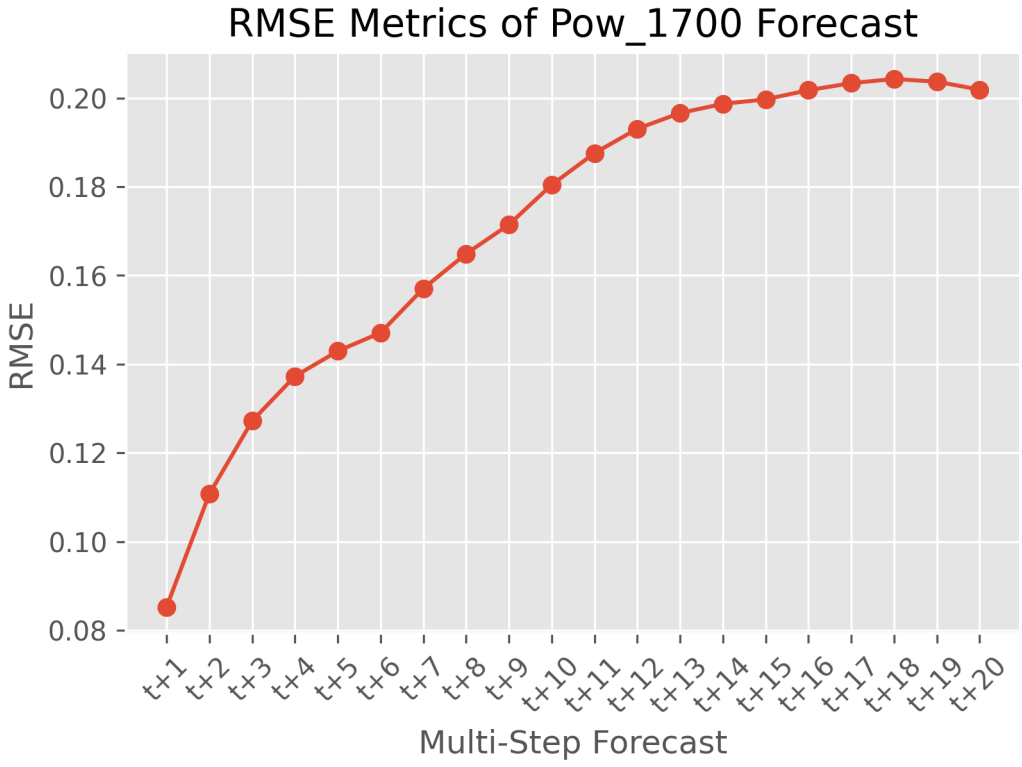

RMSE Metrics of Pow_1200 Forecast
RMSE
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

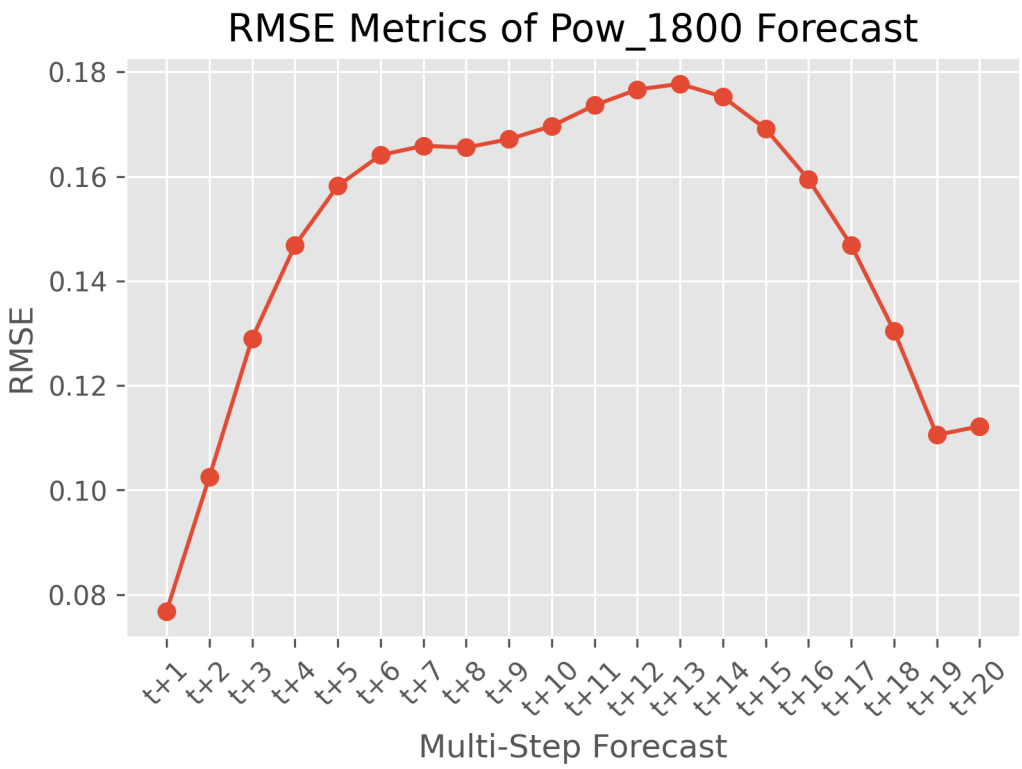

RMSE Metrics of Pow_1300 Forecast
RMSE
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

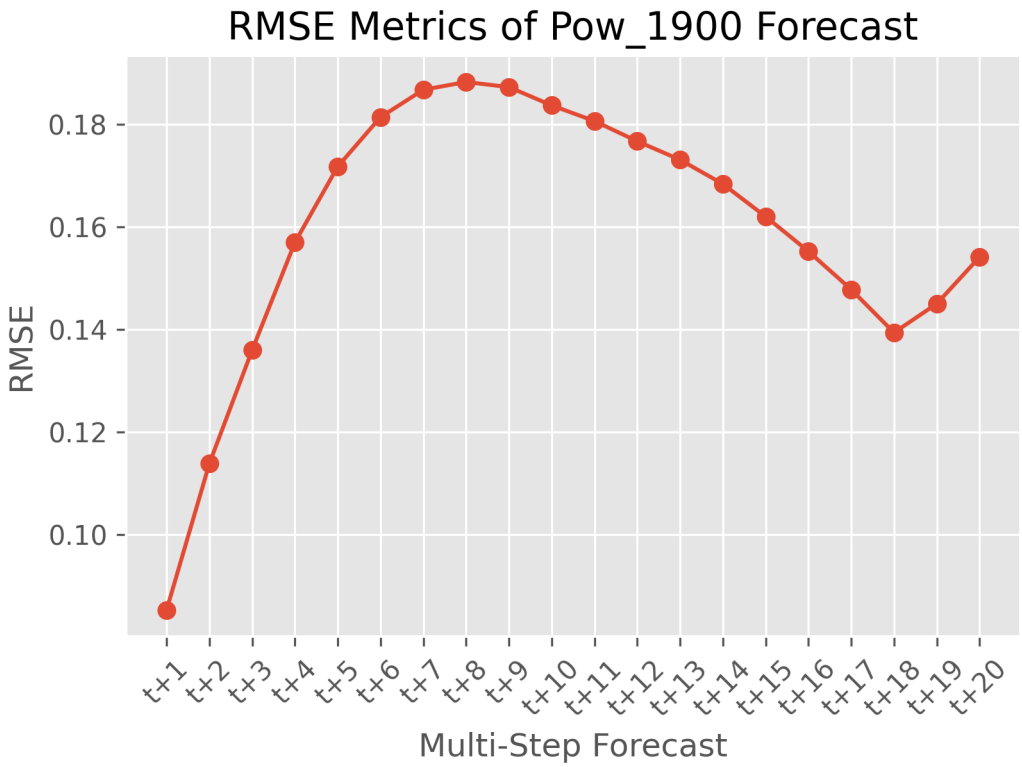

RMSE Metrics of Pow_1400 Forecast
RMSE
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast

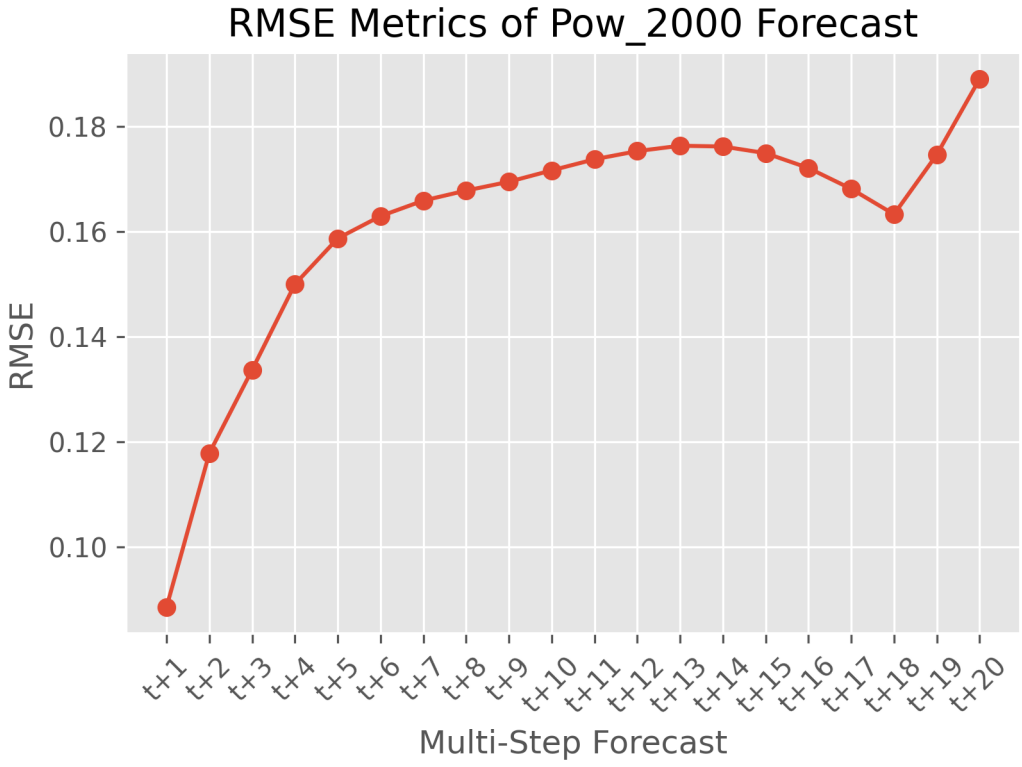

RMSE Metrics of Pow_1500 Forecast
RMSE
0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
t+1
t+2
t+3
t+4
t+5
t+6
t+7
t+8
t+9
t+10
t+11
t+12
t+13
t+14
t+15
t+16
t+17
t+18
t+19
t+20
Multi-Step Forecast


RMSE Metrics of Pow_1600 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_1700 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_1800 Forecast
RMSE
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_1900 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_2000 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

Backing up the data

[28]:
```
yhat2 = yhat
y_test2 = y_test
```

New Unseen Predictions

[29]:
```
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 515ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
```

Rescale

[30]:
```
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
```

[31]:
```
re_yhat, re_y_test = inverse_scale_data(yhat2, y_test2)
```

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

[32]:
```
y_test_3 = y_test
y_test_4 = y_test
y_test_5 = y_test
```

[33]:
```
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)
```

[34]:
```
# 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)
```

[35]:
```
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)

[36]:
```
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

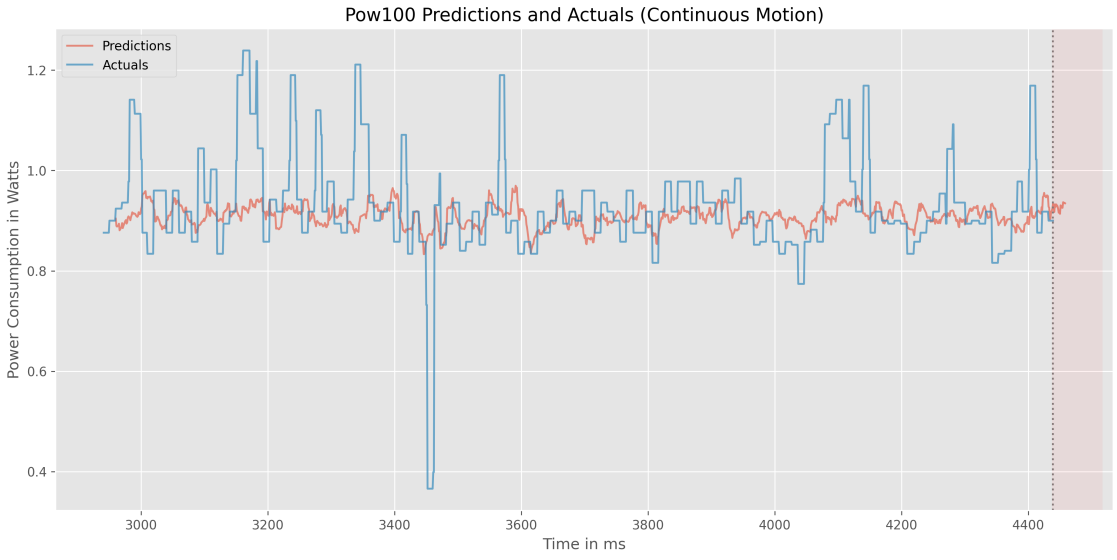
[37]:
```
from sklearn.metrics import r2_score
def print_act_and_pred_tables(yhat,ytest,forecast_horizon,␣
 ↪start_graph,end_graph):
  pow_preds = []
  pow_actuals = []
  for i in range(20):
    pow_preds.append(yhat[:, forecast_horizon-1, i]) # 1 refers to the forecast␣
 ↪horizon --> t+1; shape of yhat: [(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
```

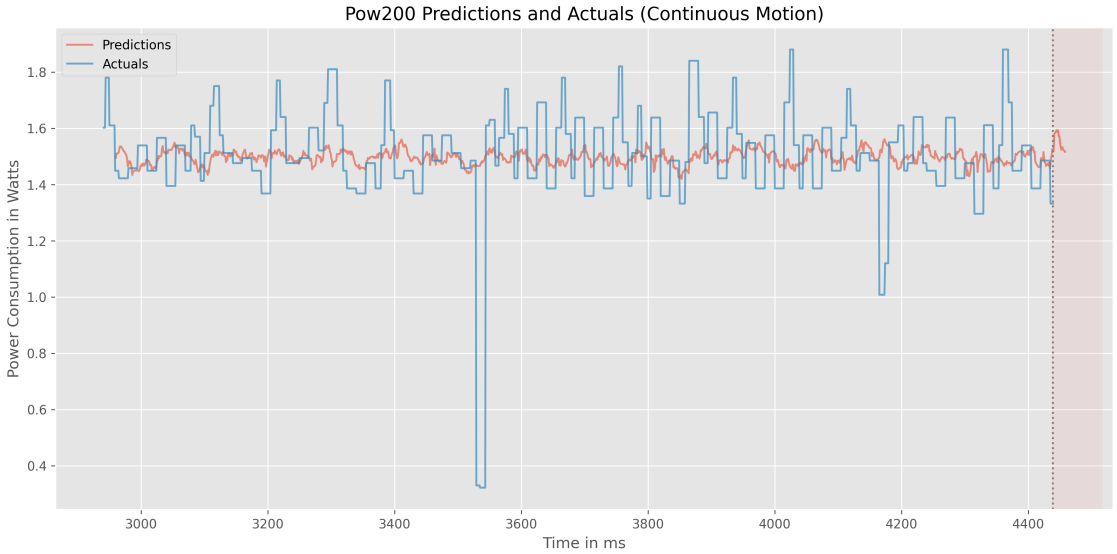
```
    actuals = df_new[f"Pow{100*(i+1)} Actuals"][start_graph:end_graph]
    predictions = df_new[f"Pow{100*(i+1)} Predictions"][start_graph:end_graph]
    # shift the t+20 prediction 20 (relating to the wished forecast horizon) to␣
↪the right
    p_temp = predictions.to_frame()
    p_temp['time_index_shift'] = 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 (Continuous 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 = 40, color="red", alpha=0.05)
    # Set the legend
    plt.legend(['Predictions', 'Actuals'])
    # Show the plot
    plt.show()
```

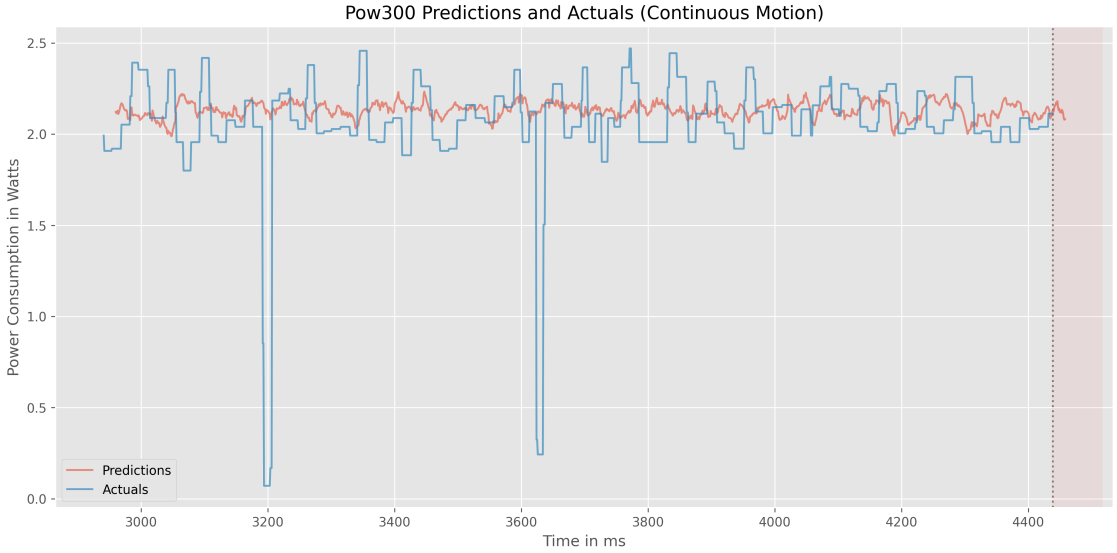
2.2 Print Actuals and Predictions

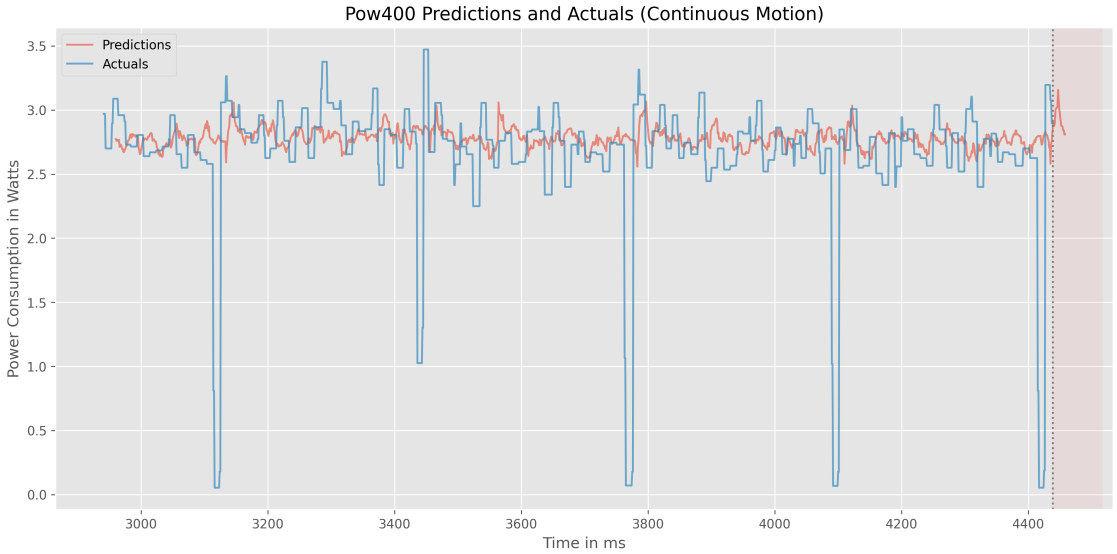
2.2.1 With Forecast Horizon of t+20 [Range: last 1500 Timestamps]

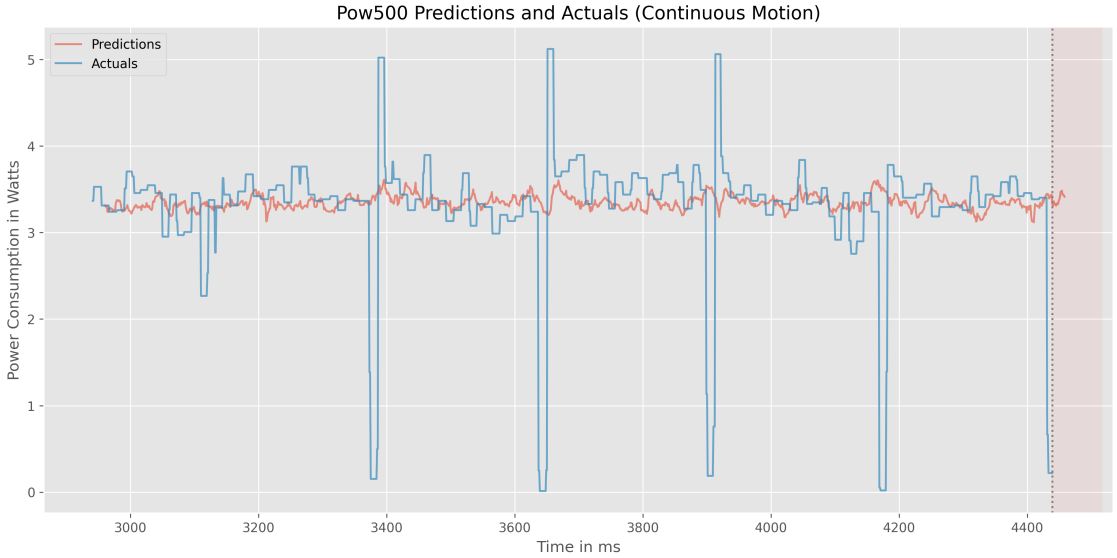
[38]:
```
print_act_and_pred_tables(re_yhat2,re_y_test2,20, -1500,-1)
```

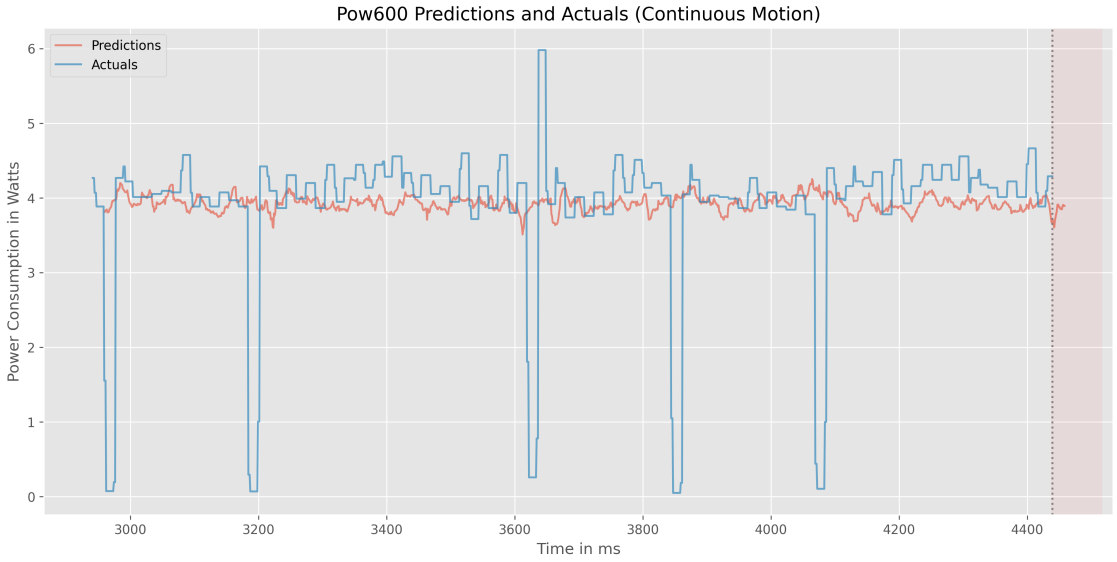


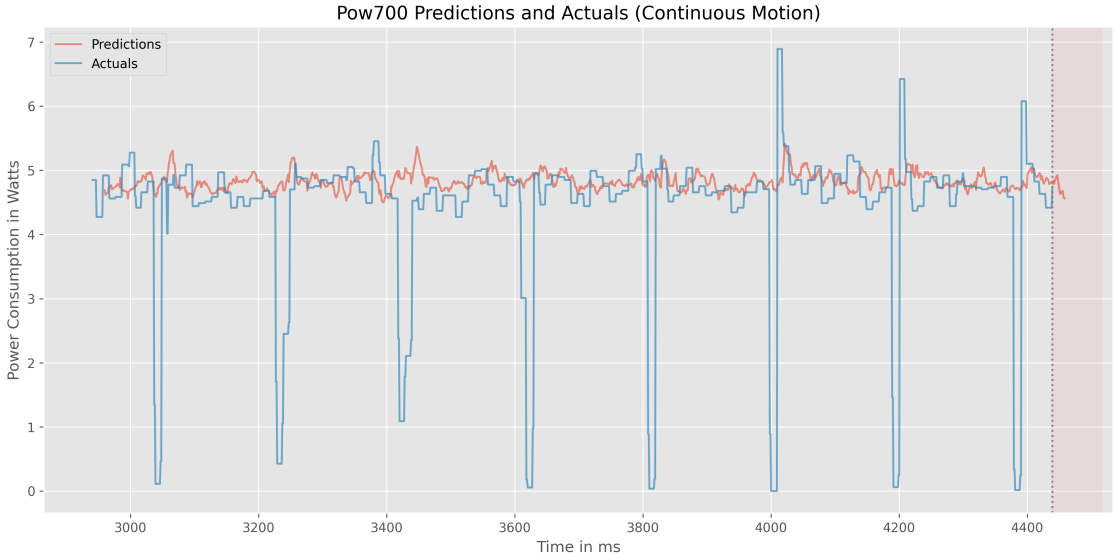

Pow200 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

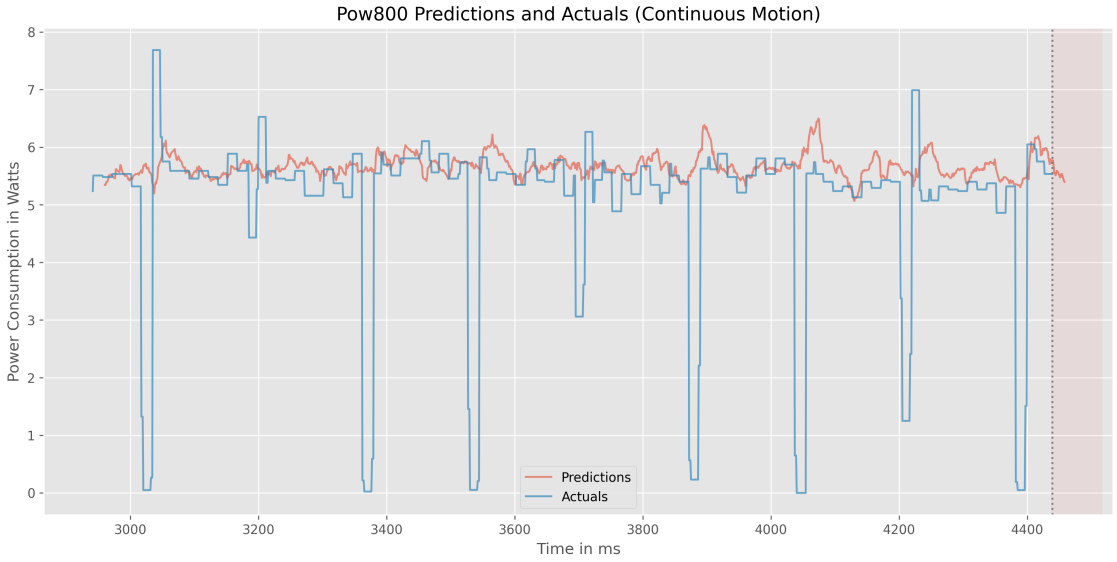

Pow300 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

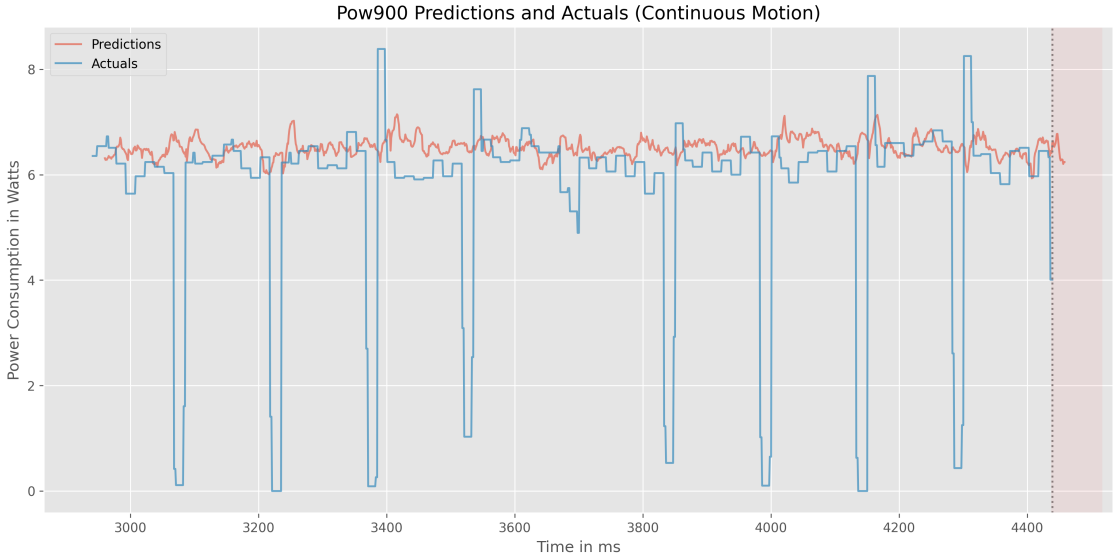

Pow400 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
3000
3200
3400
3600
3800
4000
4200
4400

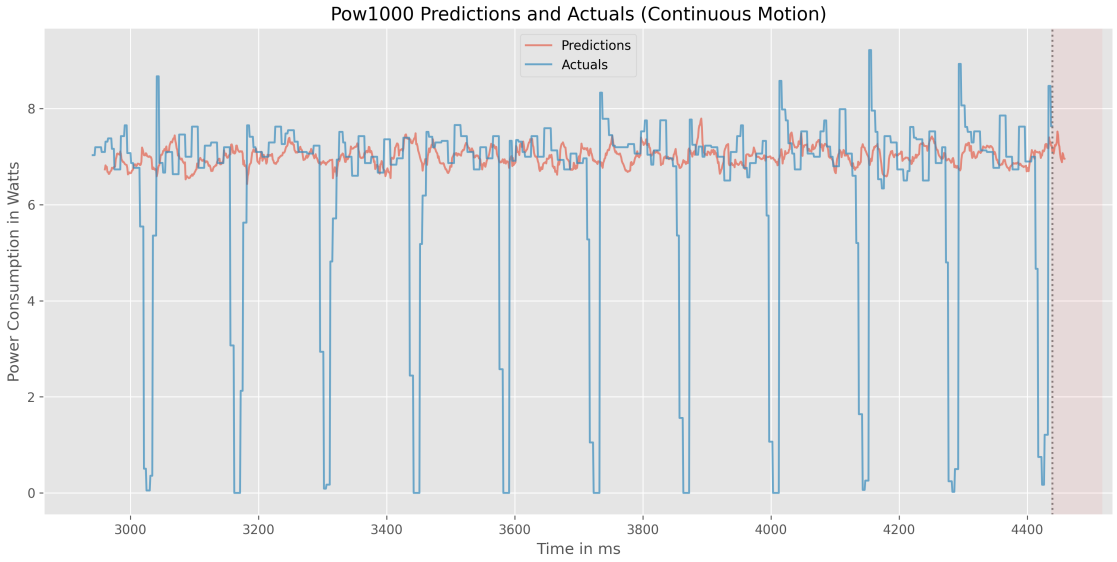

Pow500 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
3000
3200
3400
3600
3800
4000
4200
4400

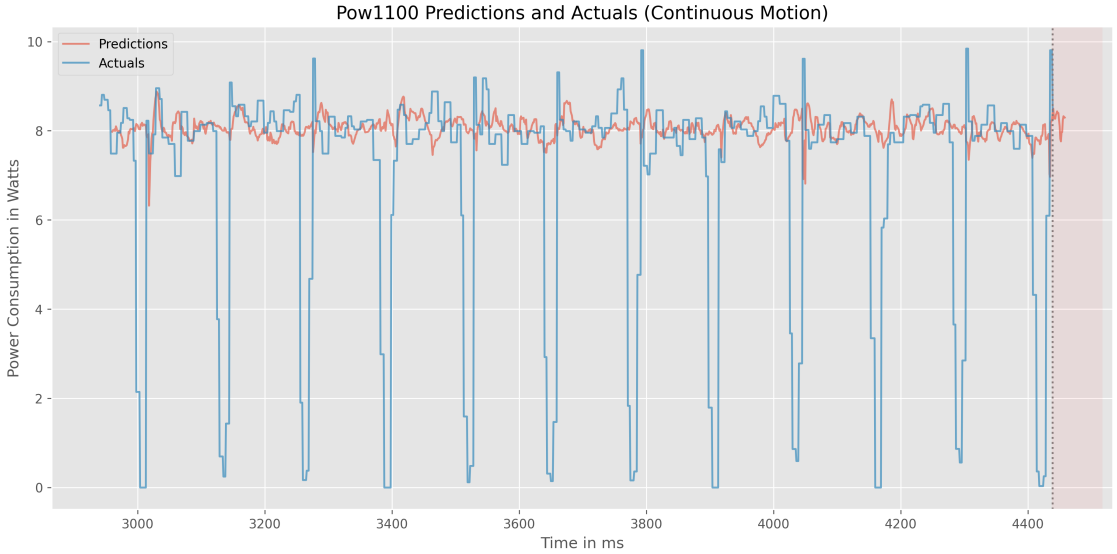

Pow600 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
3000
3200
3400
3600
3800
4000
4200
4400

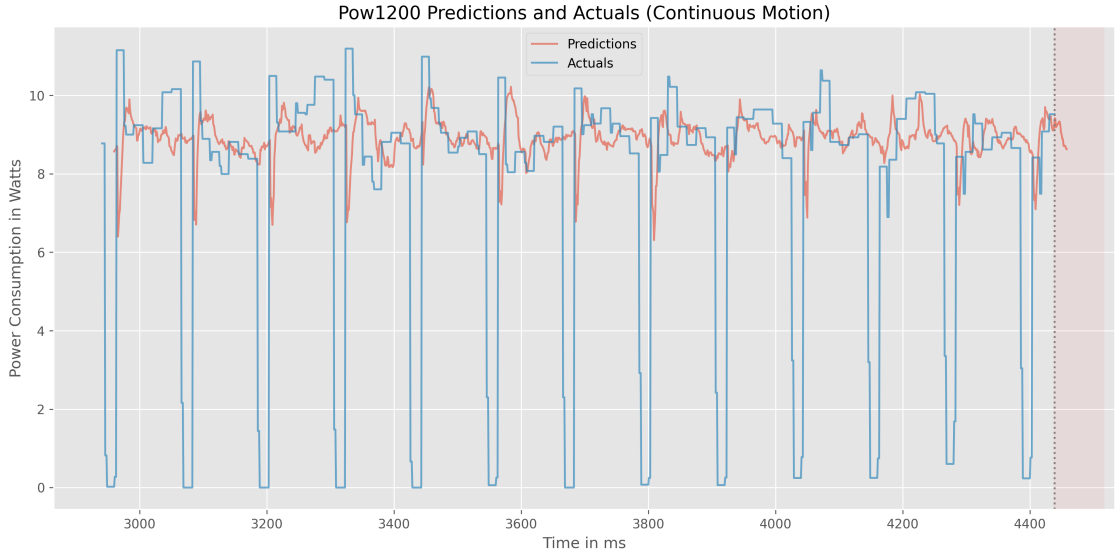

Pow700 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
7
3000
3200
3400
3600
3800
4000
4200
4400

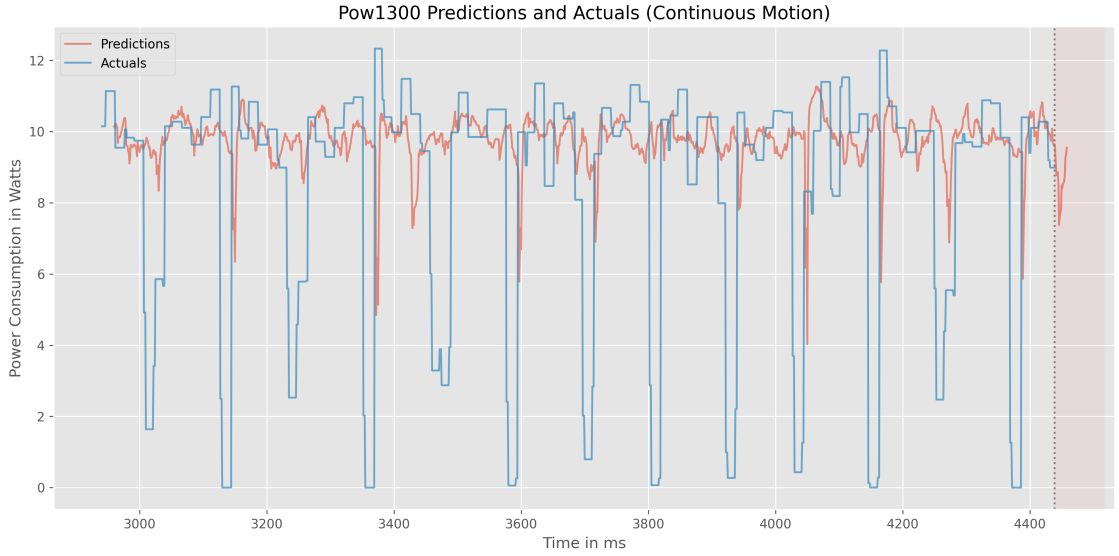

Pow800 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
Time in ms
Predictions
Actuals
0
1
2
3
4
5
6
7
8
3000
3200
3400
3600
3800
4000
4200
4400

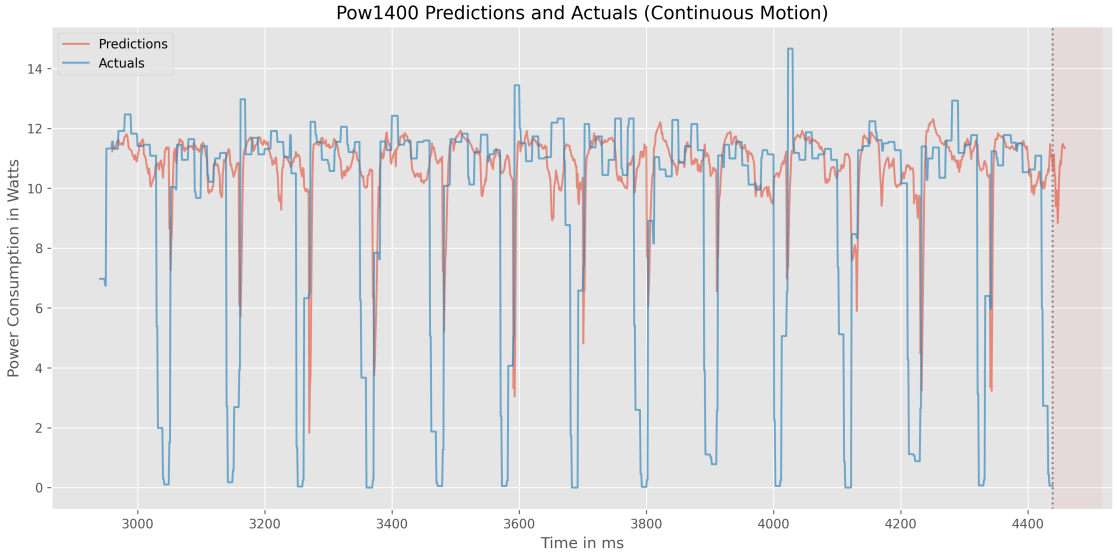

Pow900 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
3000
3200
3400
3600
3800
4000
4200
4400

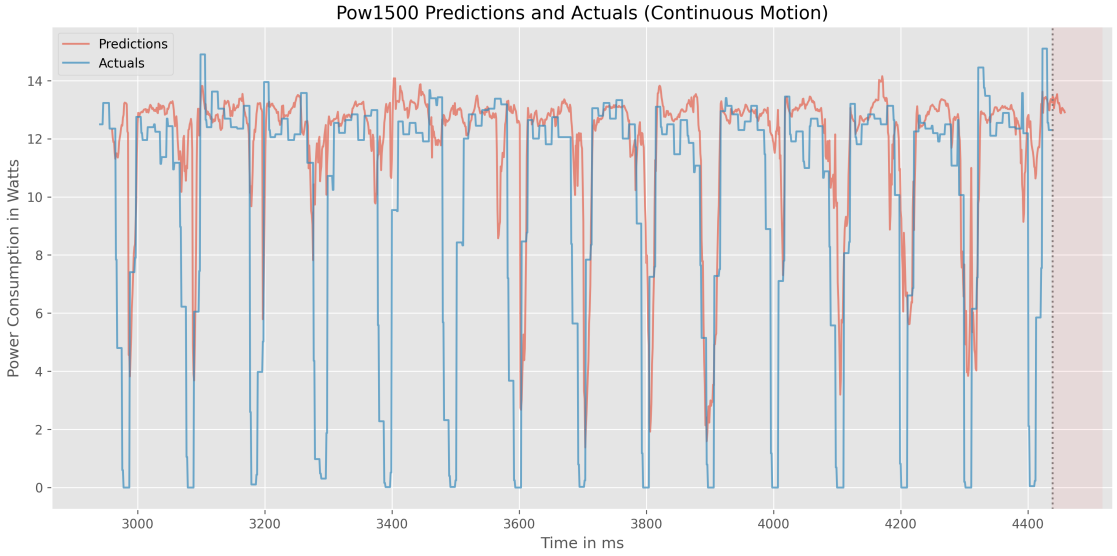

Pow1000 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
3000
3200
3400
3600
3800
4000
4200
4400

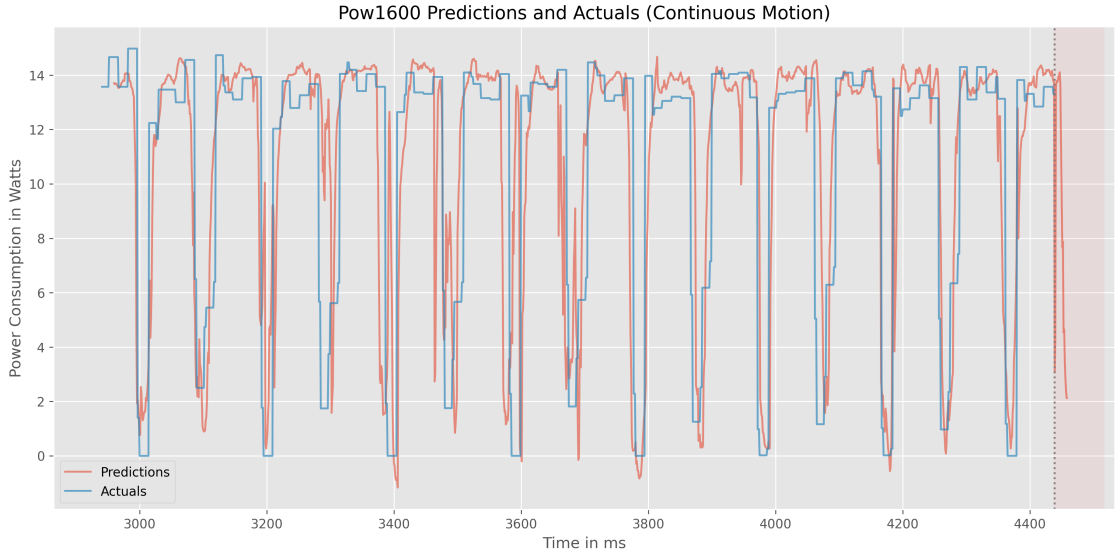

Pow1100 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
3000
3200
3400
3600
3800
4000
4200
4400

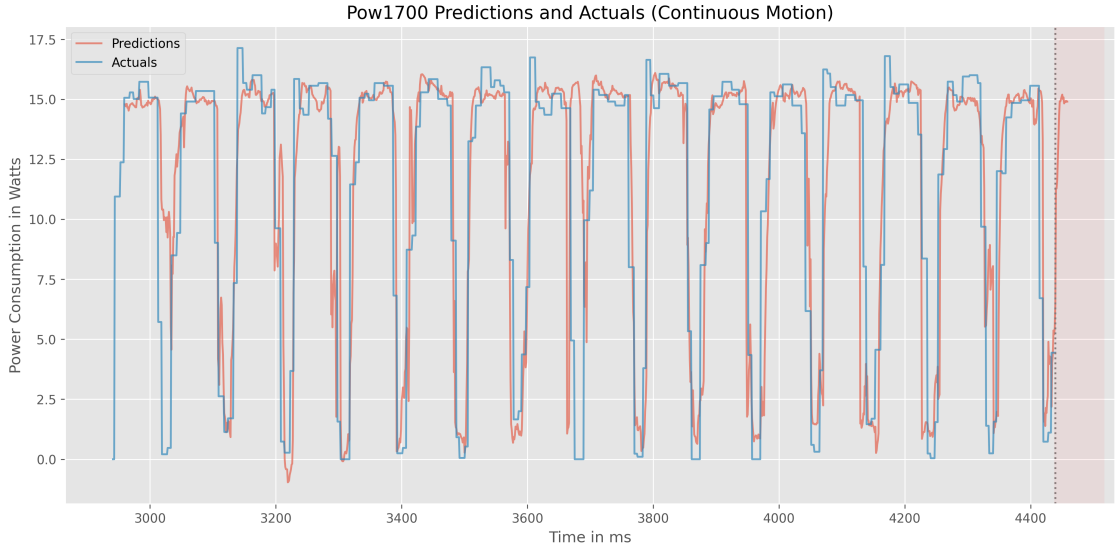

Pow1200 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
3000
3200
3400
3600
3800
4000
4200
4400

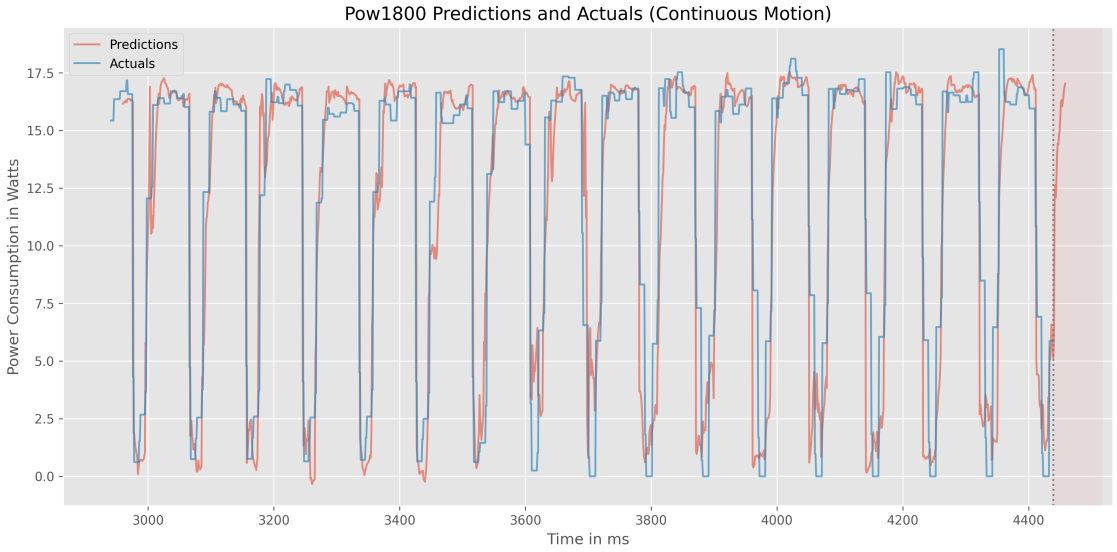

Pow1300 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
3000
3200
3400
3600
3800
4000
4200
4400

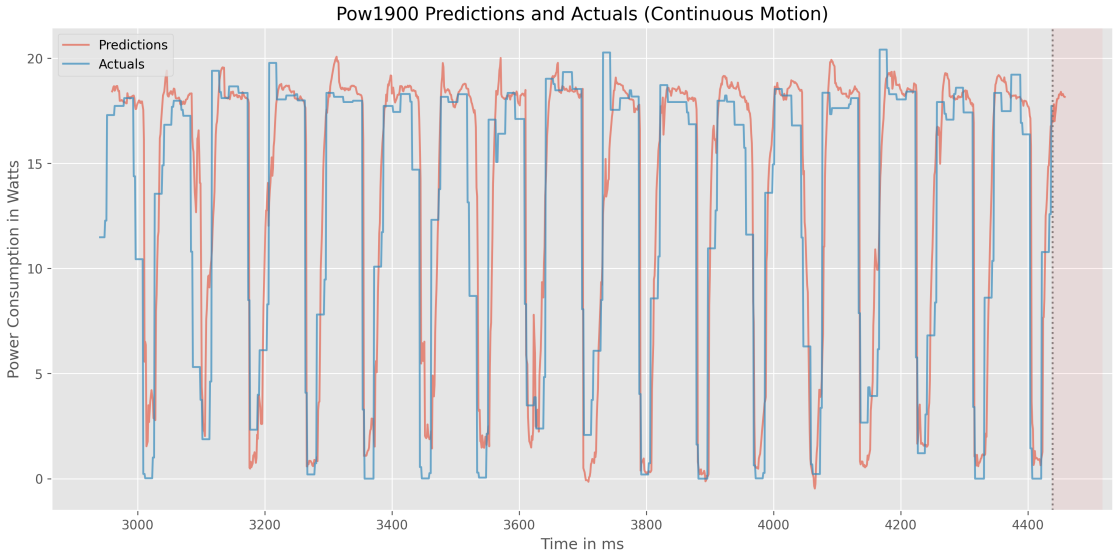

Pow1400 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
3000
3200
3400
3600
3800
4000
4200
4400

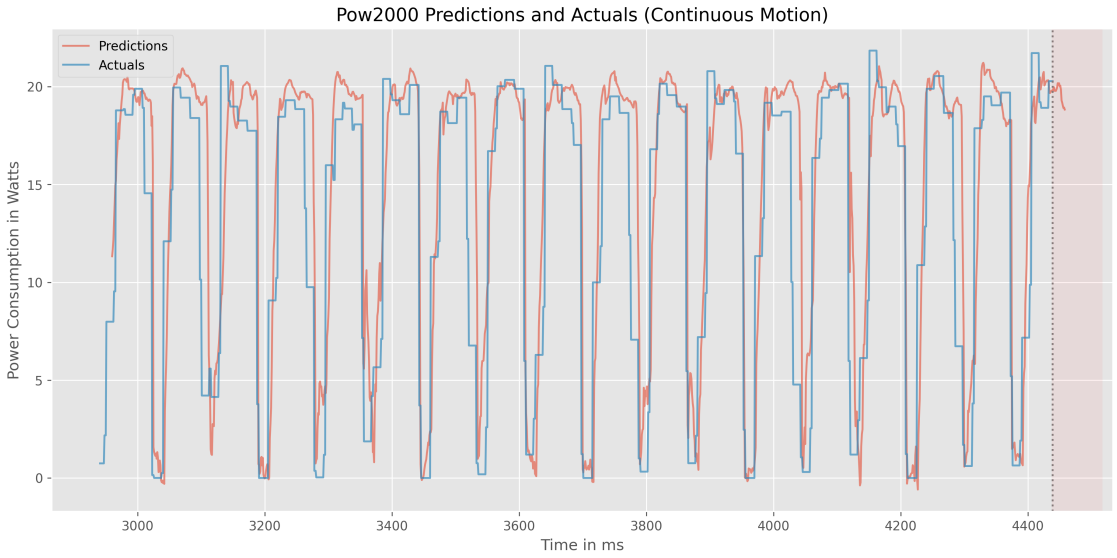

Pow1500 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
3000
3200
3400
3600
3800
4000
4200
4400


Pow1600 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
Time in ms
Predictions
Actuals
0
2
4
6
8
10
12
14
3000
3200
3400
3600
3800
4000
4200
4400


Pow1700 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
Time in ms
Predictions
Actuals
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3000
3200
3400
3600
3800
4000
4200
4400

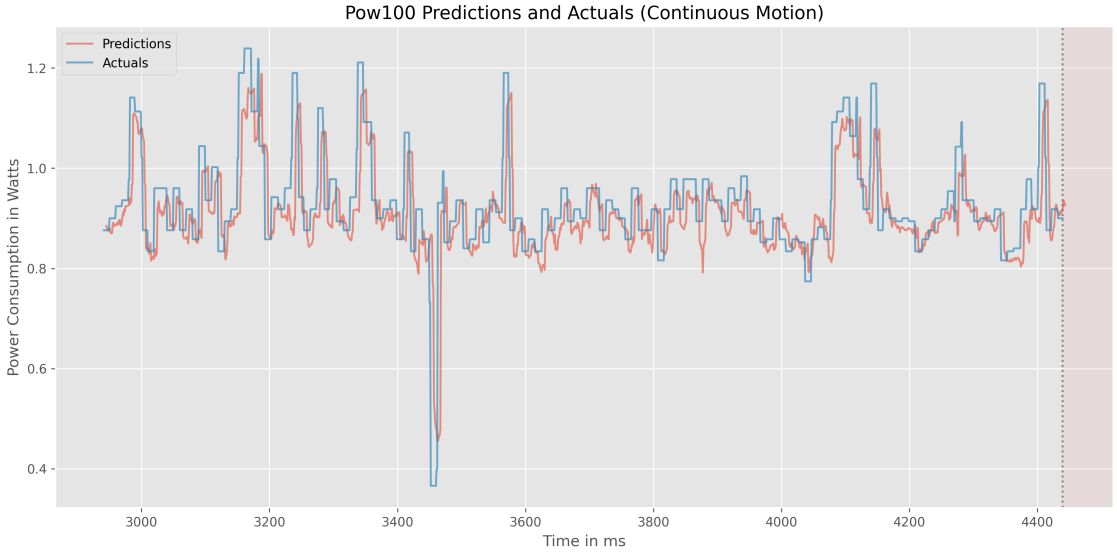

Pow1800 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
3000
3200
3400
3600
3800
4000
4200
4400

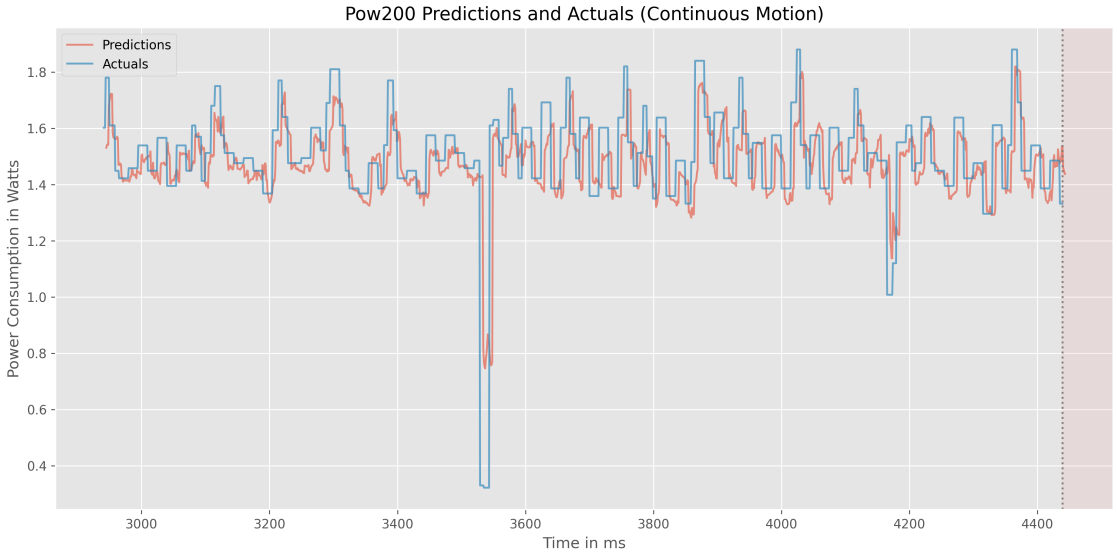

Pow1900 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
5
10
15
20
3000
3200
3400
3600
3800
4000
4200
4400

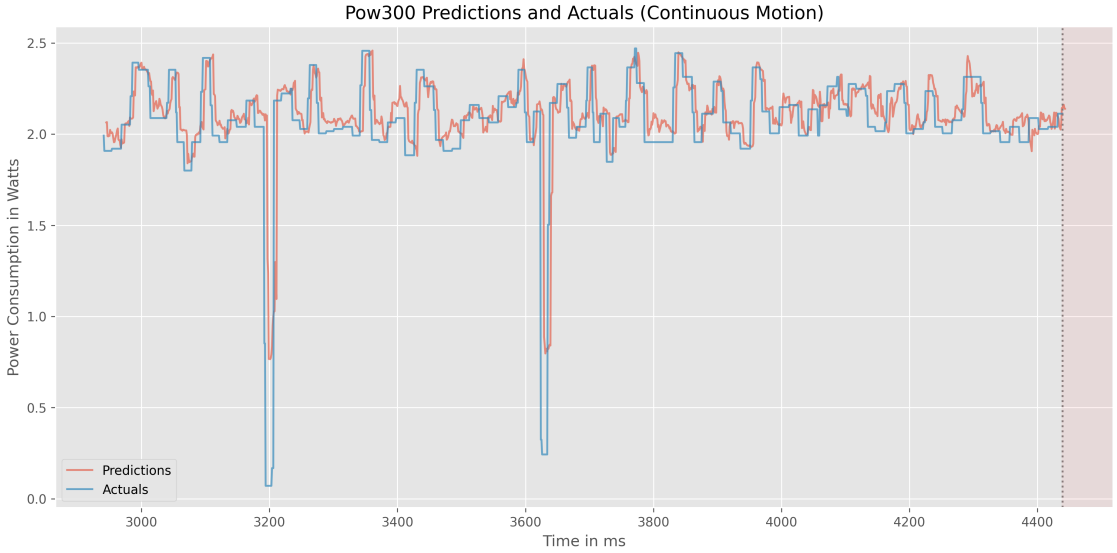


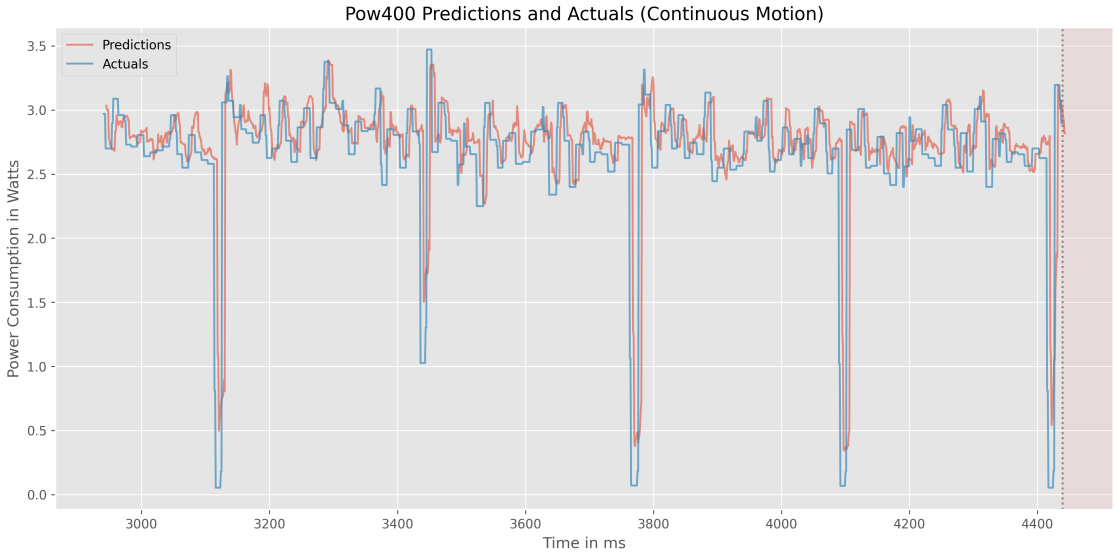
2.2.2 With Forecast Horizon of t+5 [Range: last 1500 Timestamps]

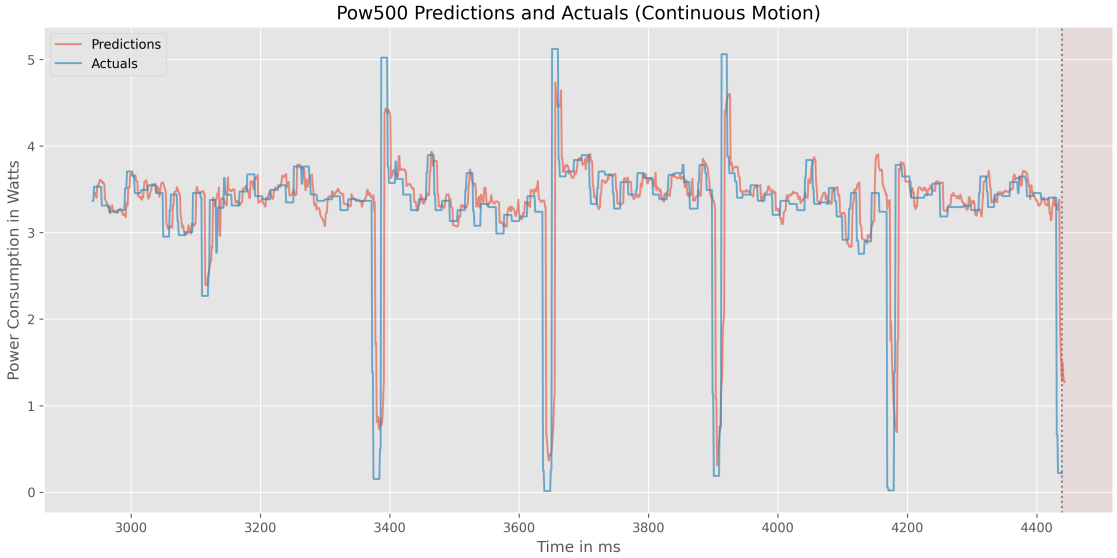
[39]:
```
print_act_and_pred_tables(re_yhat2,re_y_test2,5, -1500,-1)
```

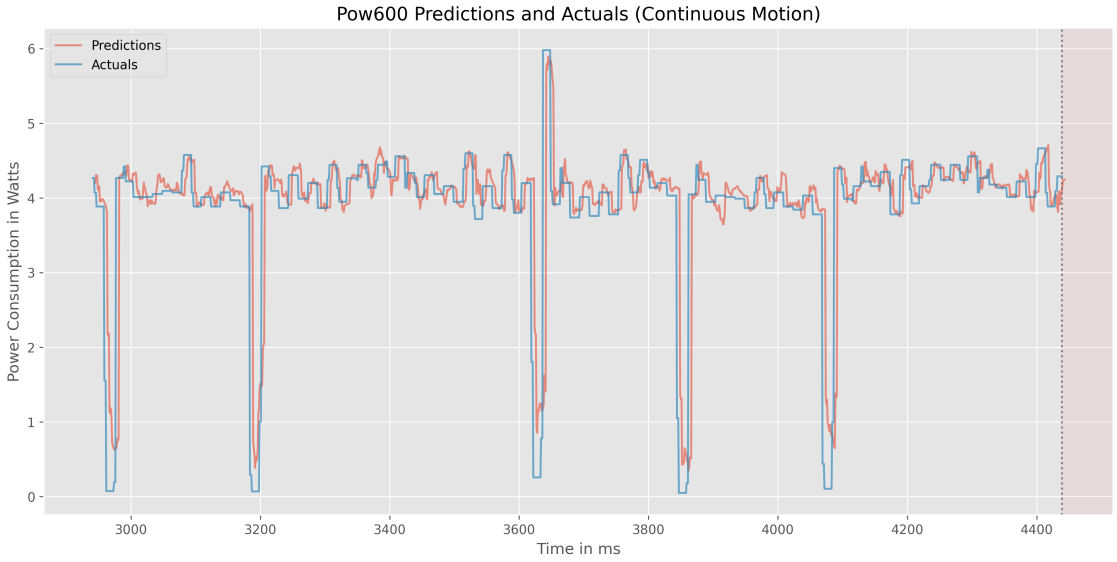


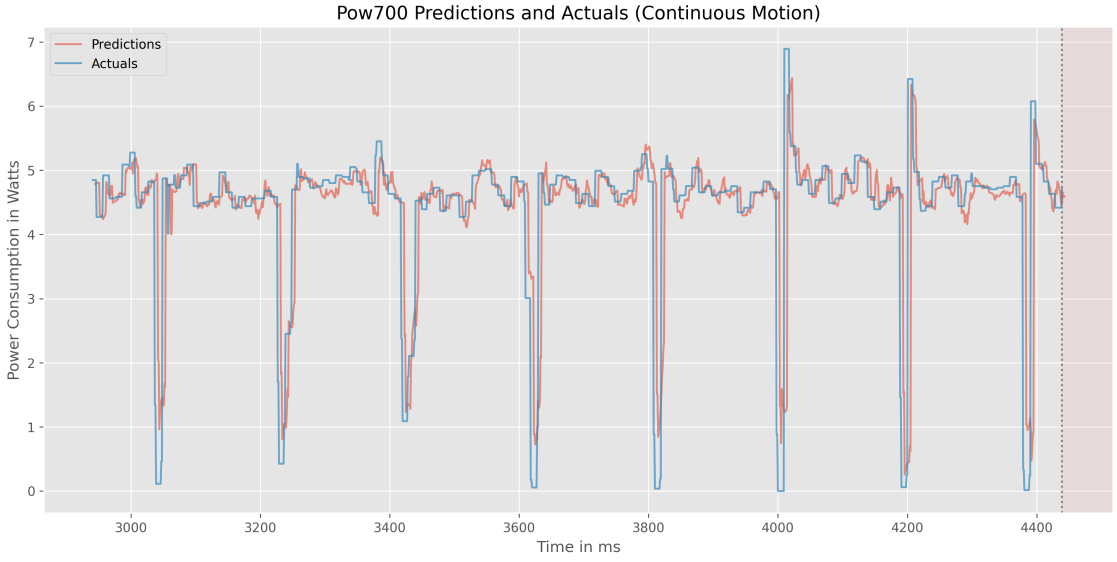

Pow200 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

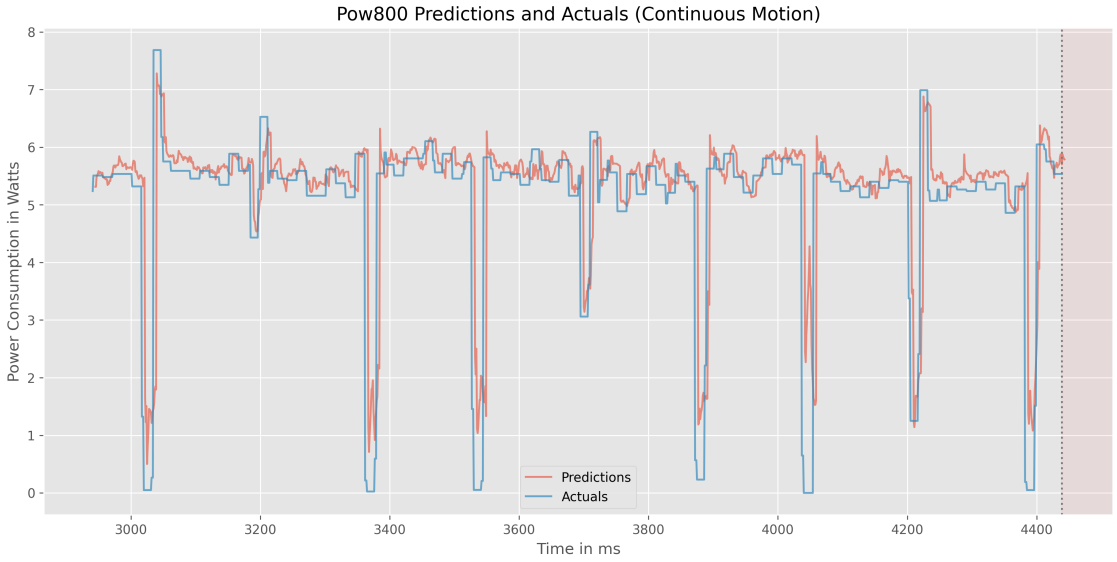

Pow300 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

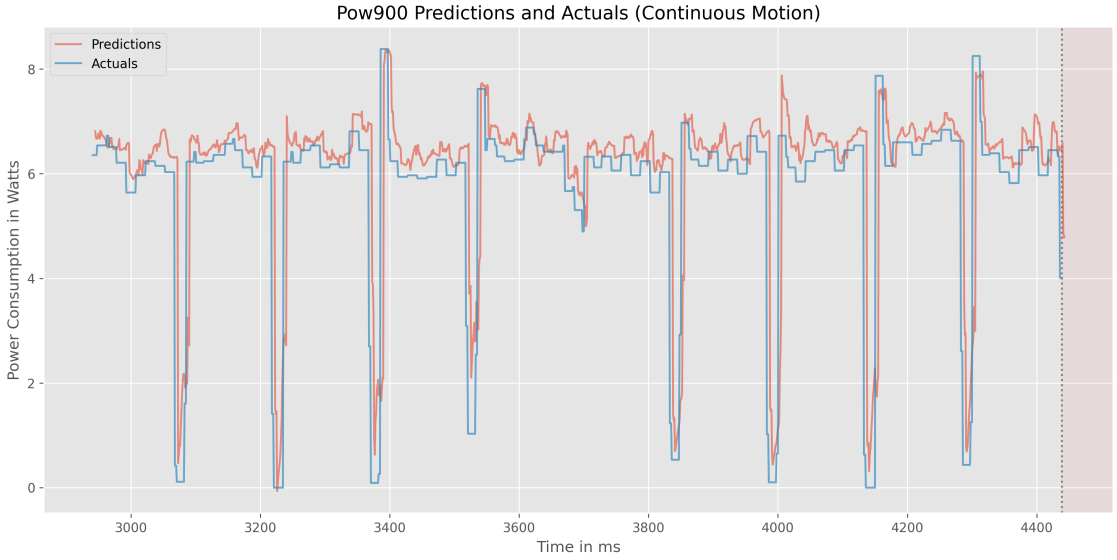

Pow400 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
3000
3200
3400
3600
3800
4000
4200
4400

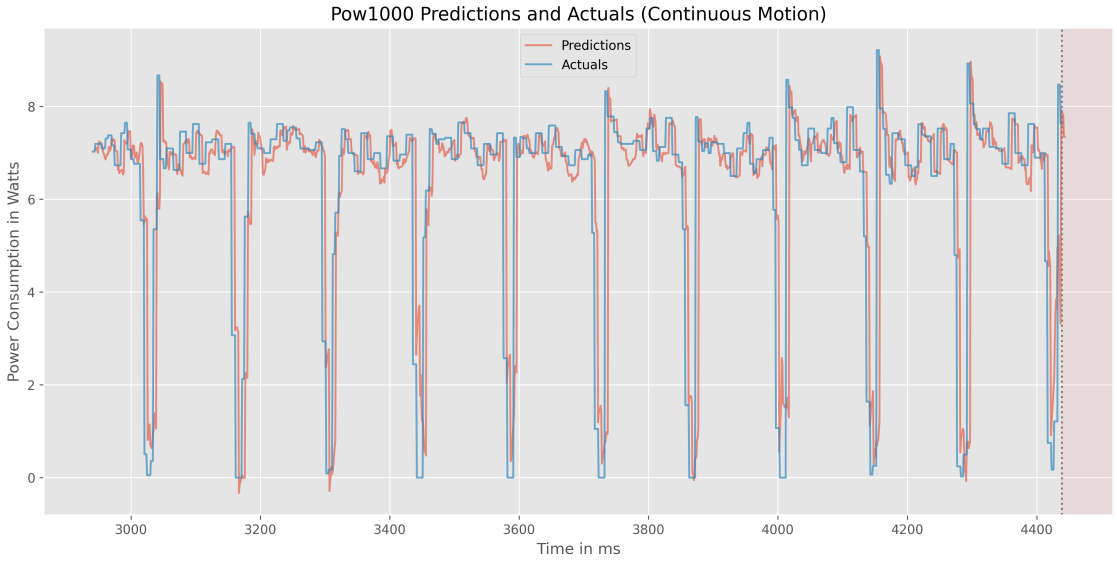

Pow500 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
3000
3200
3400
3600
3800
4000
4200
4400

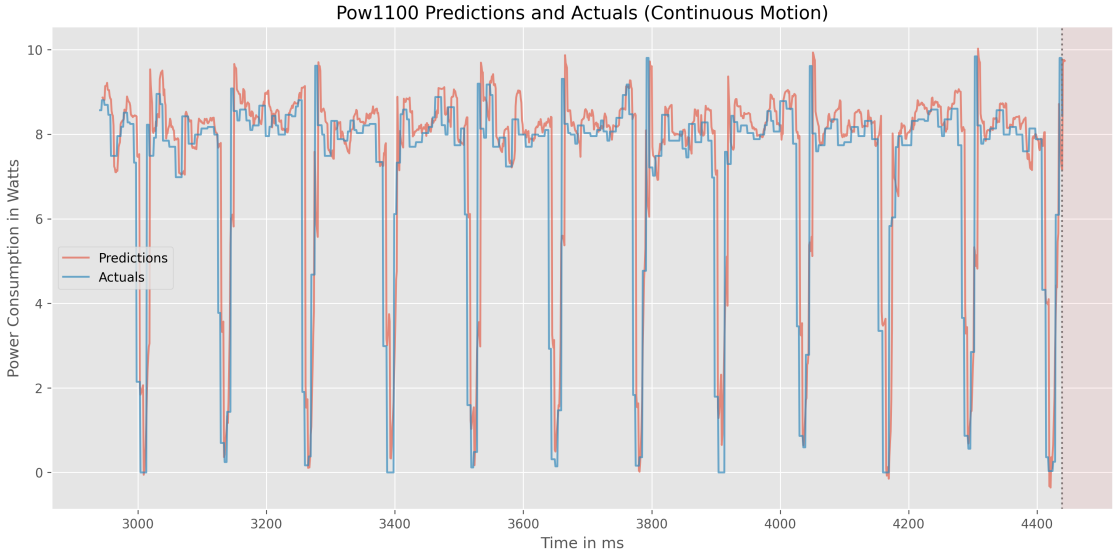

Pow600 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
3000
3200
3400
3600
3800
4000
4200
4400

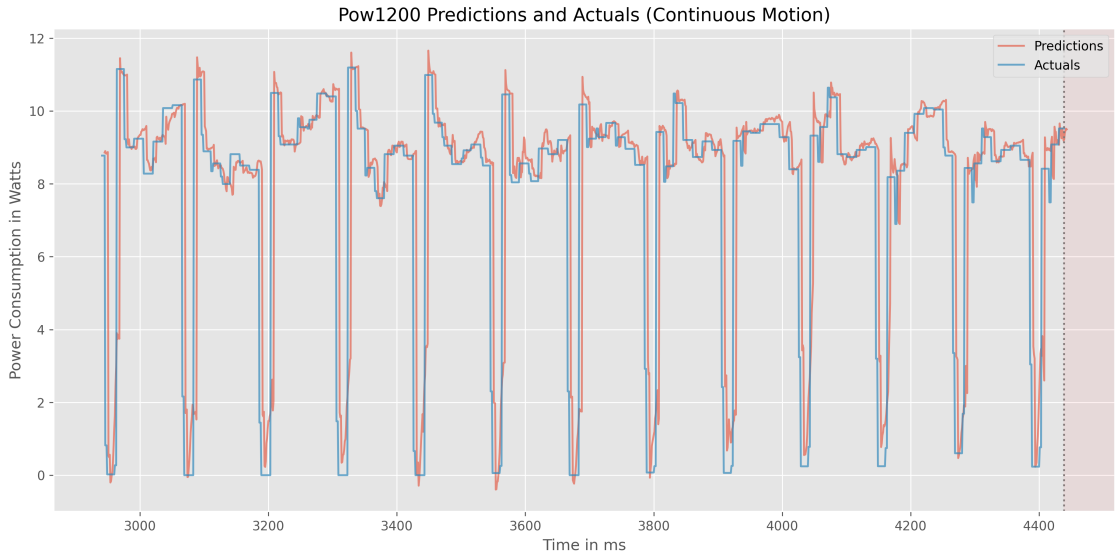

Pow700 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
7
3000
3200
3400
3600
3800
4000
4200
4400

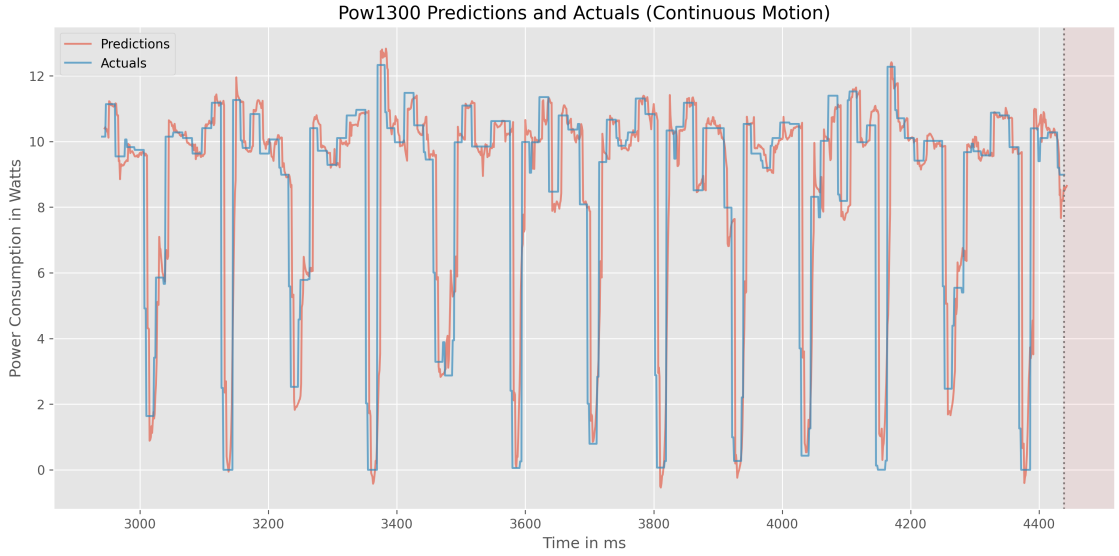

Pow800 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
Time in ms
Predictions
Actuals
0
1
2
3
4
5
6
7
8
3000
3200
3400
3600
3800
4000
4200
4400

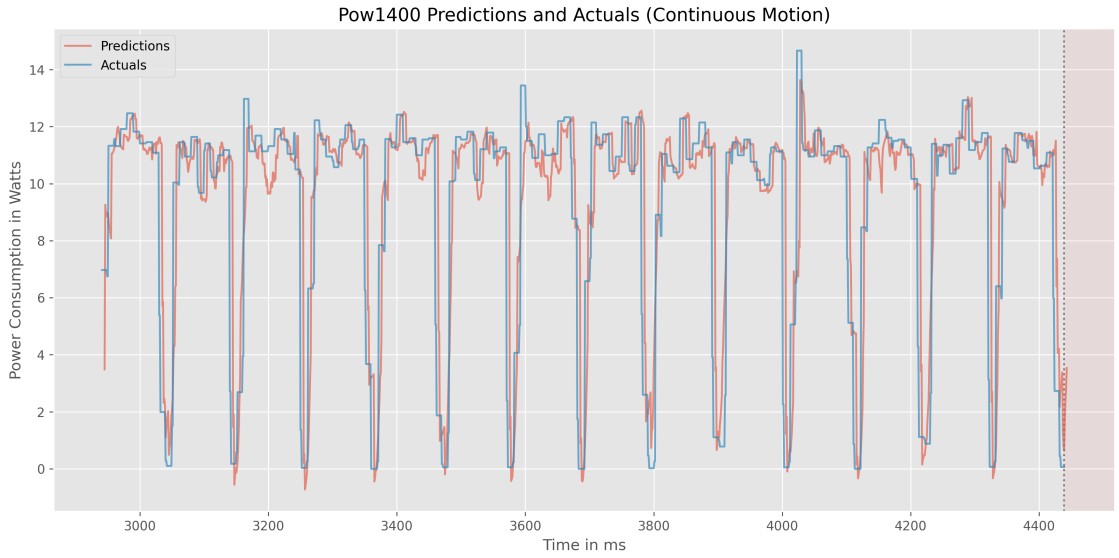

Pow900 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
Time in ms
Predictions
Actuals
0
2
4
6
8
3000
3200
3400
3600
3800
4000
4200
4400

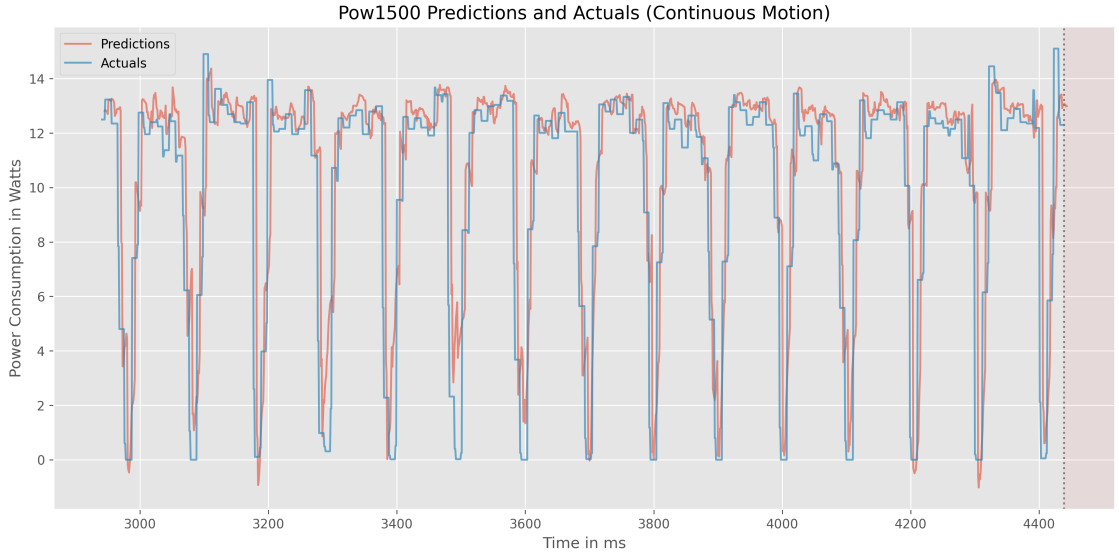

Pow1000 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
3000
3200
3400
3600
3800
4000
4200
4400
Time in ms

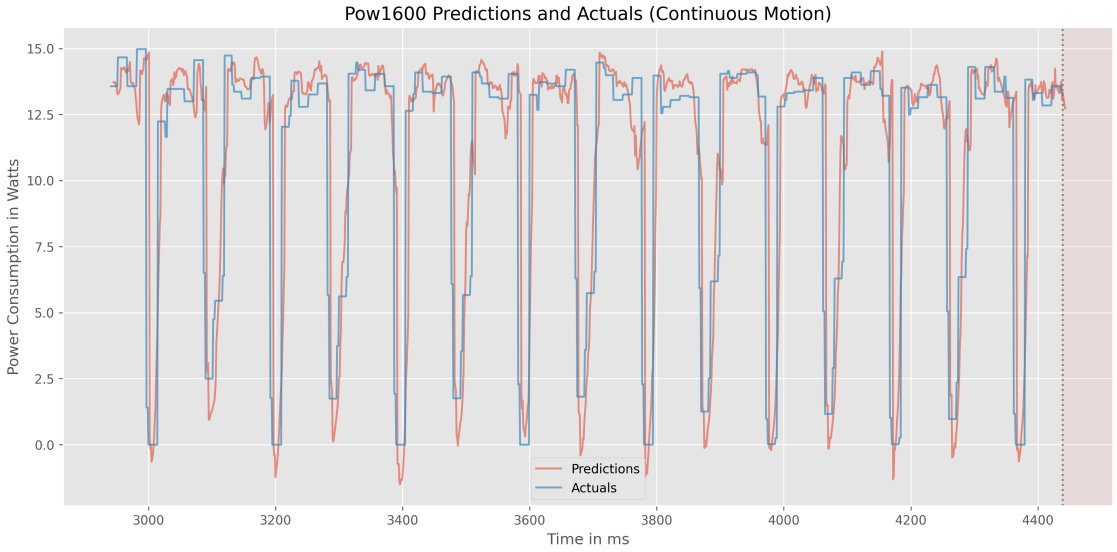

Pow1100 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
3000
3200
3400
3600
3800
4000
4200
4400
Time in ms

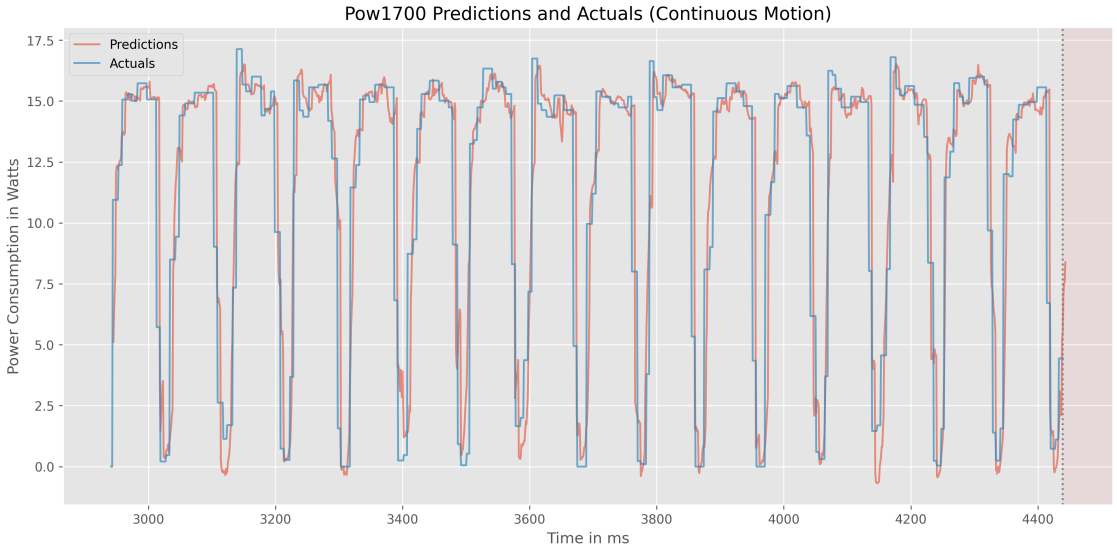

Pow1200 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
3000
3200
3400
3600
3800
4000
4200
4400

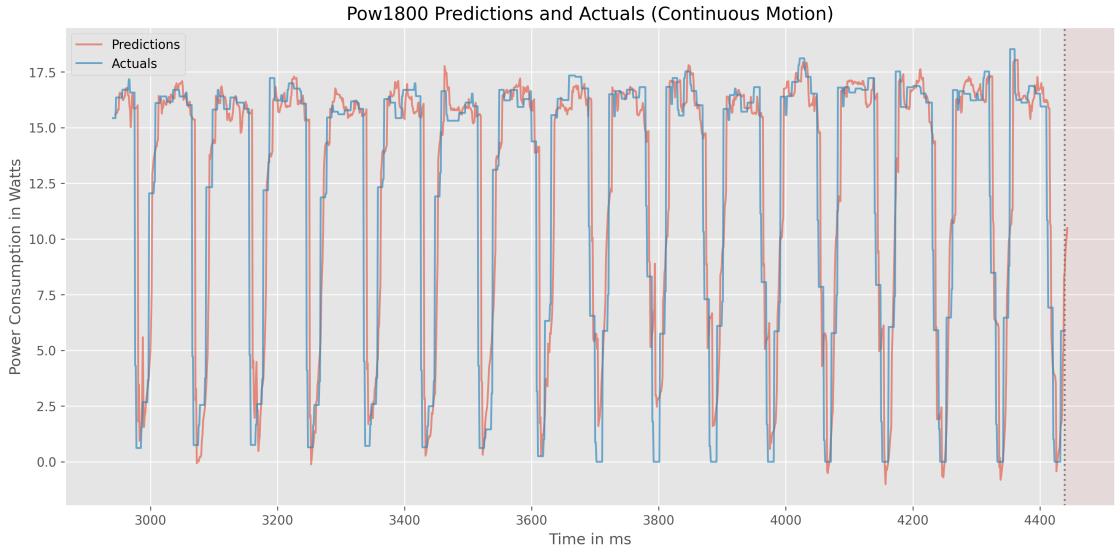

Pow1300 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
3000
3200
3400
3600
3800
4000
4200
4400

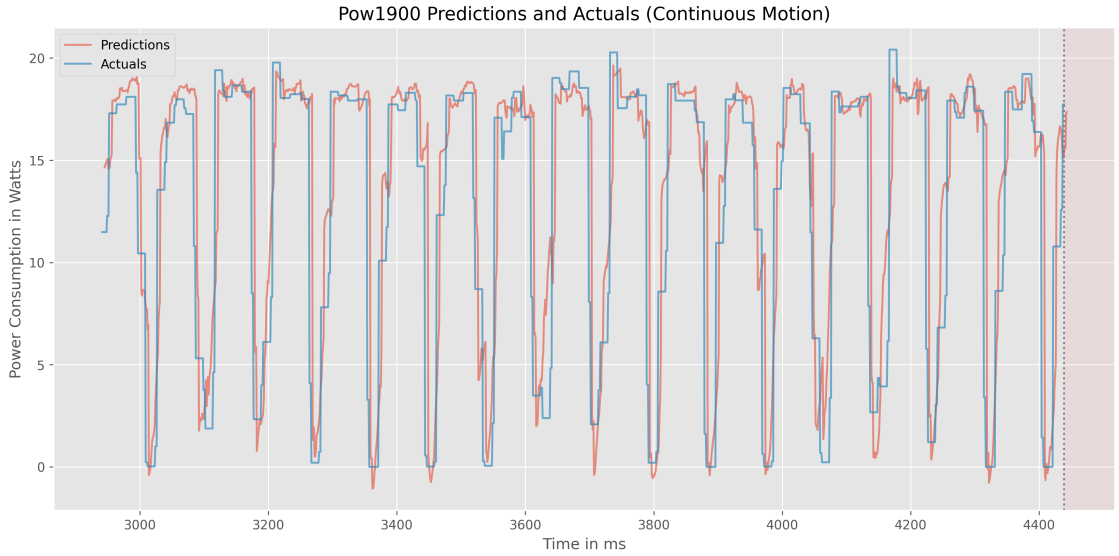

Pow1400 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
3000
3200
3400
3600
3800
4000
4200
4400

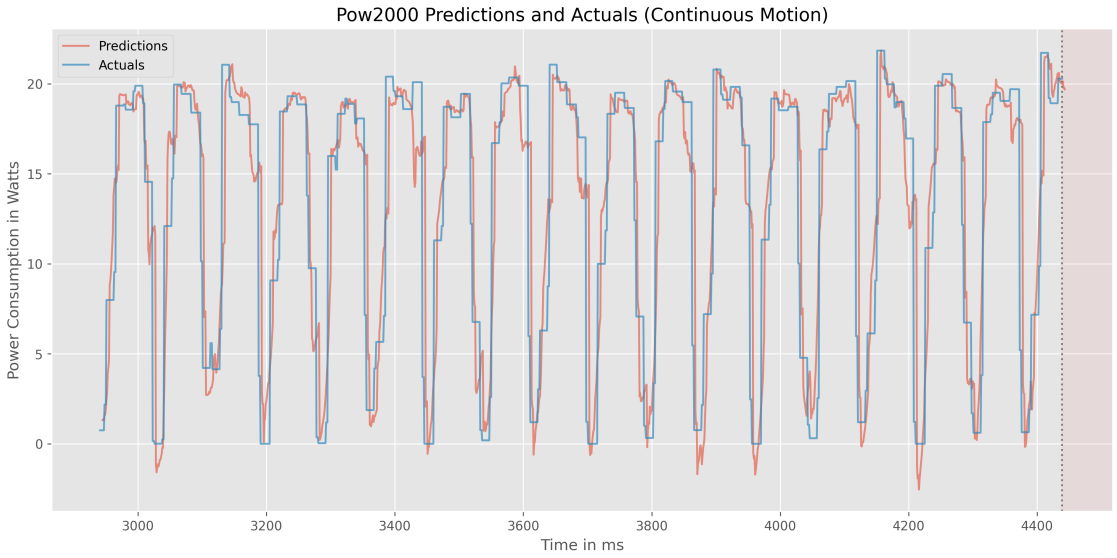

Pow1500 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
3000
3200
3400
3600
3800
4000
4200
4400


Pow1600 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
Time in ms
Predictions
Actuals
15.0
12.5
10.0
7.5
5.0
2.5
0.0
3000
3200
3400
3600
3800
4000
4200
4400


Pow1700 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
Time in ms
Predictions
Actuals
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
3000
3200
3400
3600
3800
4000
4200
4400


Pow1800 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms


Pow1900 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms

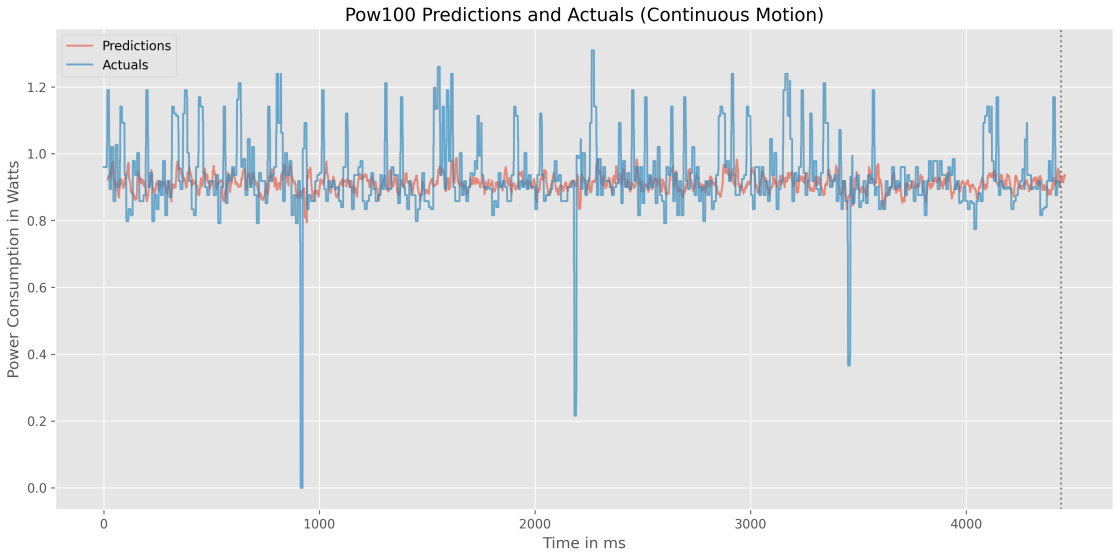


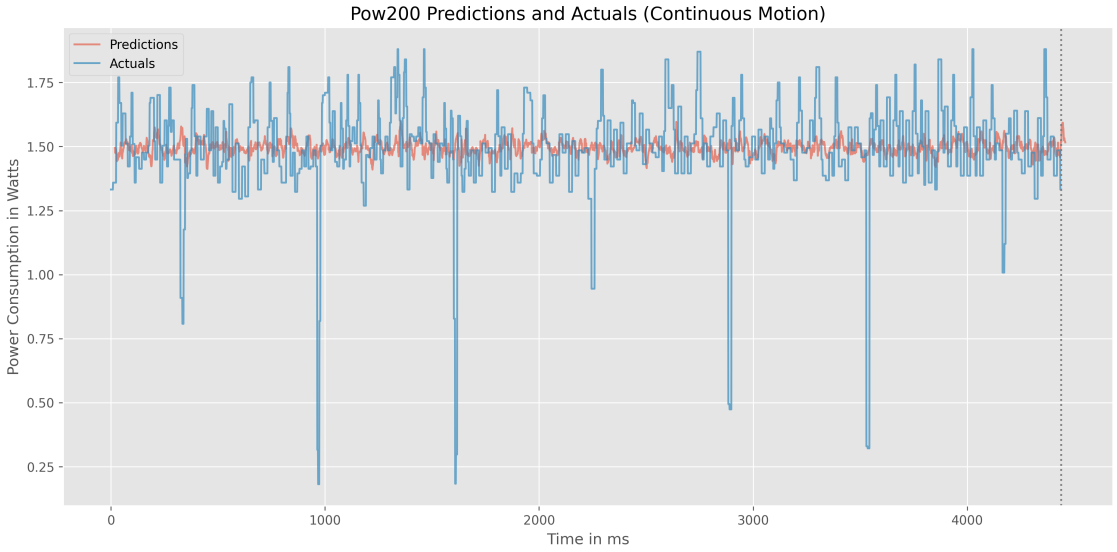
2.2.3 With Forecast Horizon of t+20 [Range: All Timestamps]

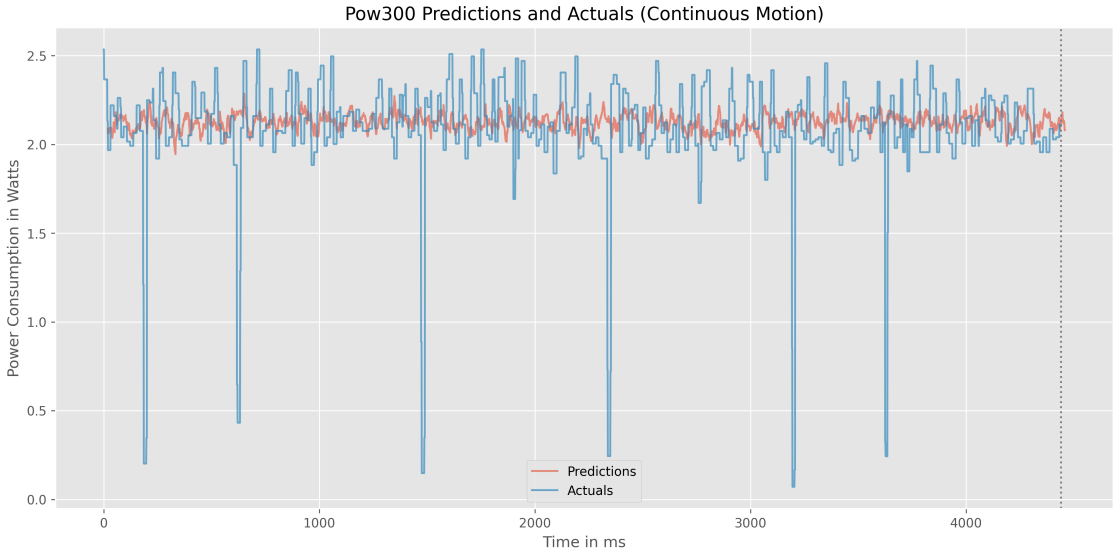
[40]:
```
from sklearn.metrics import r2_score
def print_act_and_pred_tables_all(yhat,ytest,forecast_horizon,␣
 ↪start_graph,end_graph):
  pow_preds = []
  pow_actuals = []
  for i in range(20):
    pow_preds.append(yhat[:, forecast_horizon-1, i]) # 1 refers to the forecast␣
 ↪horizon --> t+1; shape 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
```

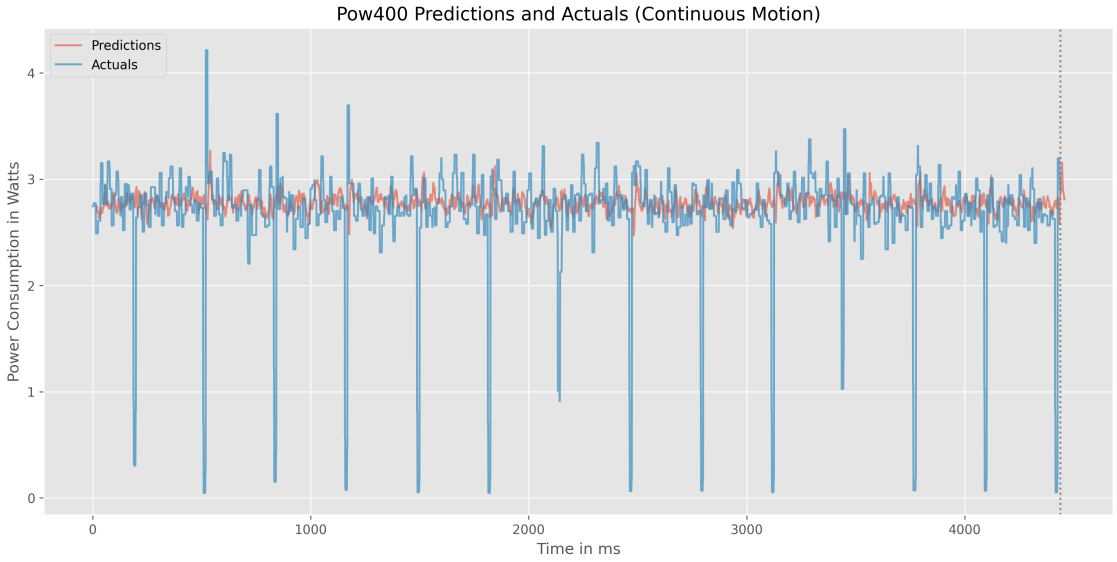
```
    p_temp = predictions.to_frame()
    p_temp['time_index_shift'] = predictions.index+forecast_horizon-1
    p_temp.set_index('time_index_shift', inplace=True)
    # Plot the data
    plt.plot(p_temp, alpha=0.6)
    plt.plot(actuals, alpha=0.7)
    # Set the plot title and axis labels
    plt.title(f'Pow{100*(i+1)}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

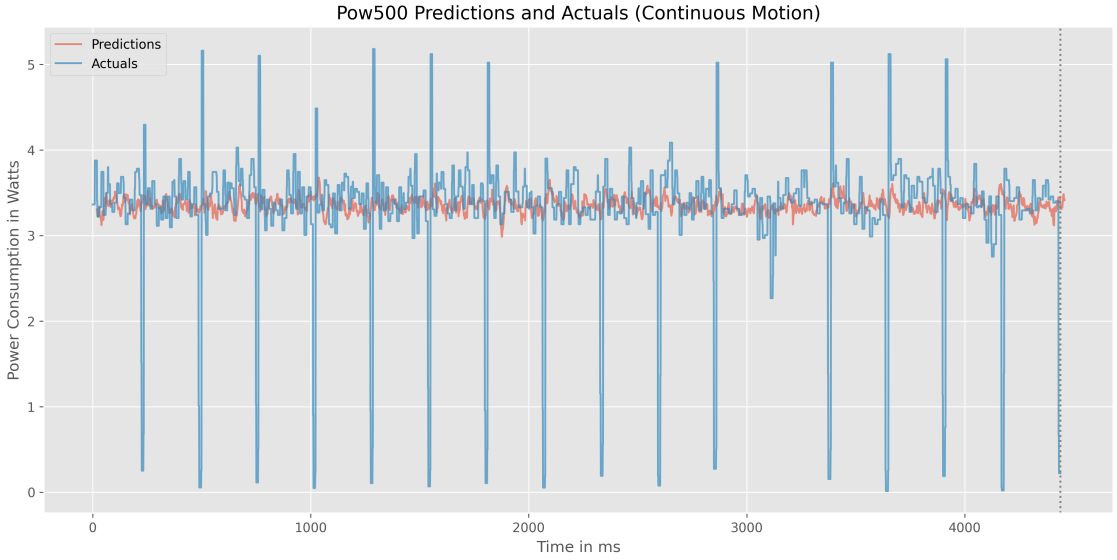
[41]:
```
print_act_and_pred_tables_all(re_yhat2,re_y_test2,20,0,-1)
```

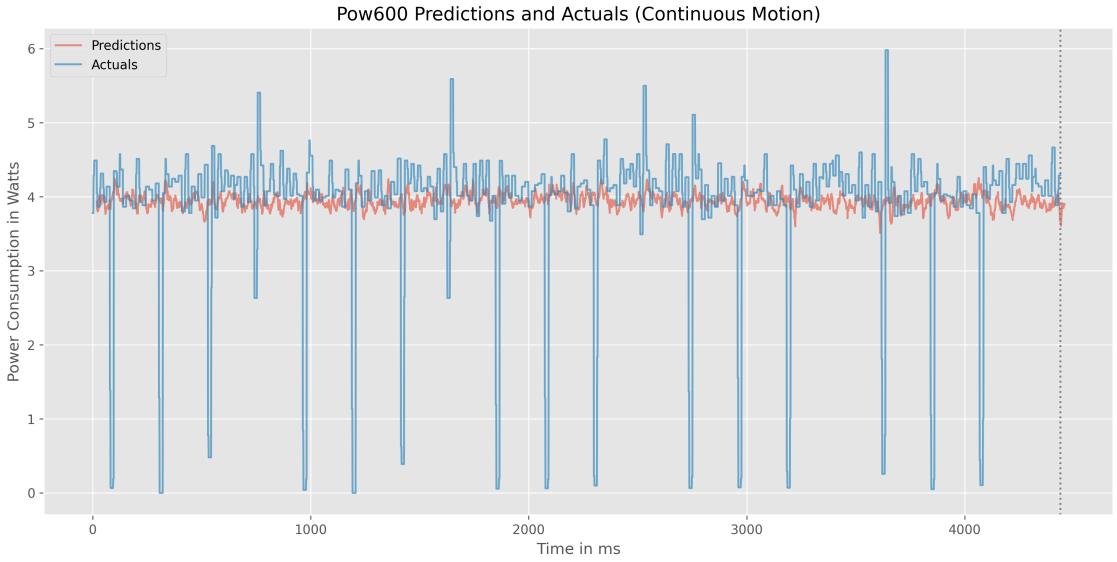


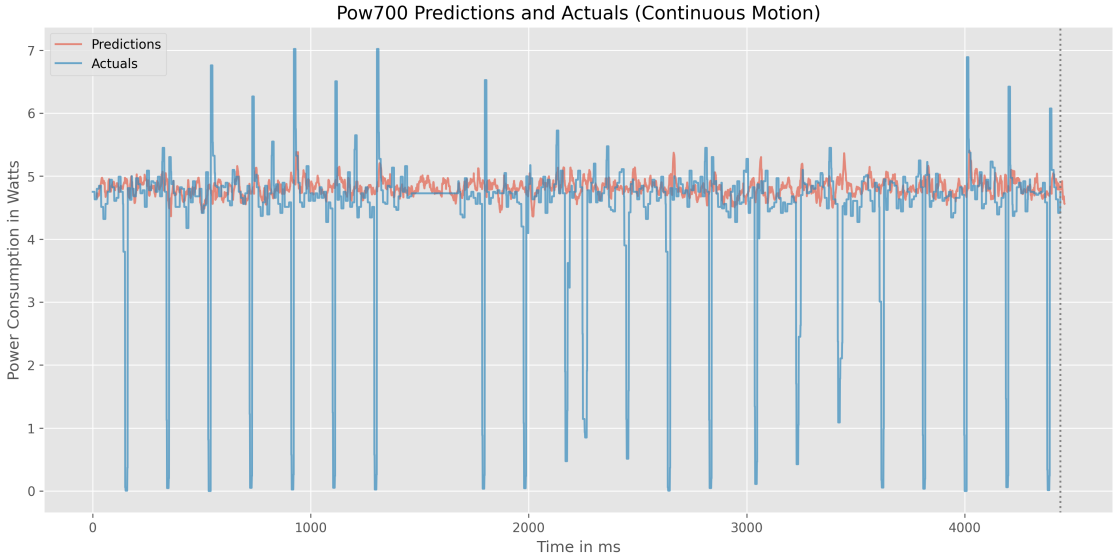

Pow200 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
1000
2000
3000
4000
Time in ms

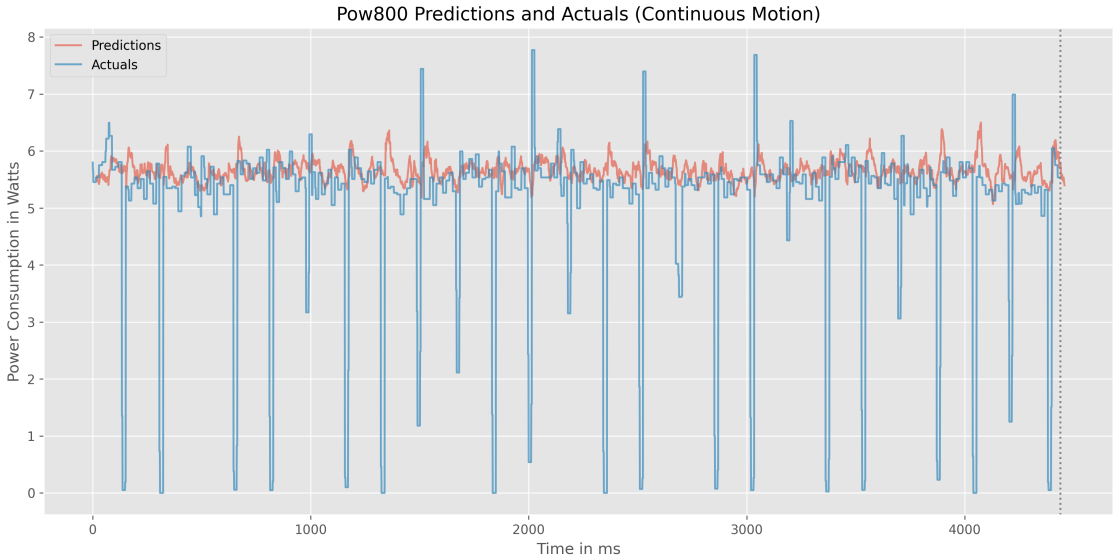

Pow300 Predictions and Actuals (Continuous Motion)
Power Consumption in Watts
2.5
2.0
1.5
1.0
0.5
0.0
Predictions
Actuals
0
1000
2000
3000
4000
Time in ms

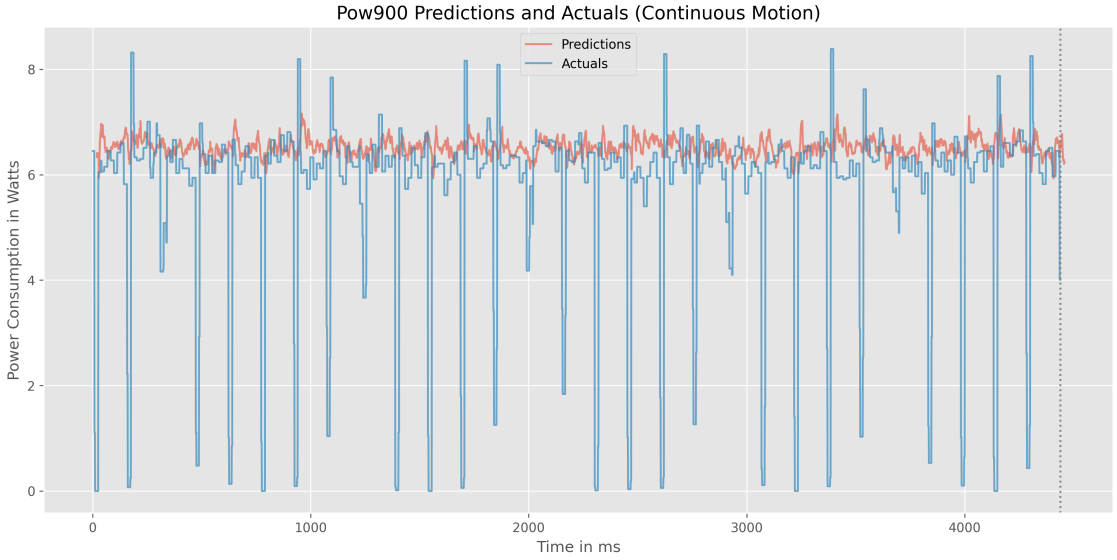

Pow400 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
0
1
2
3
4
0
1000
2000
3000
4000
Time in ms

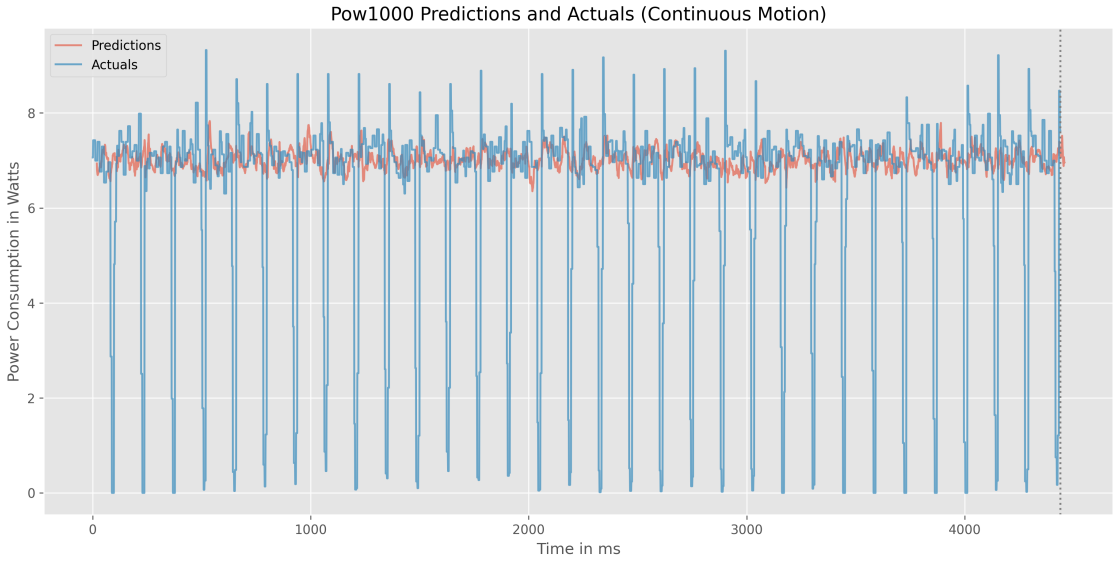

Pow500 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
0
1
2
3
4
5
0
1000
2000
3000
4000
Time in ms

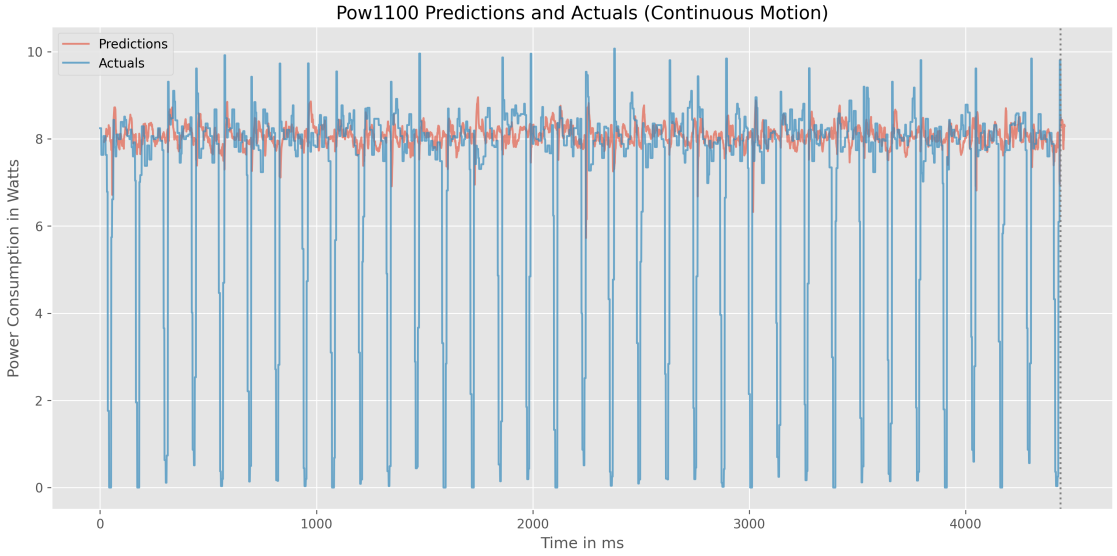

Pow600 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
0
1000
2000
3000
4000

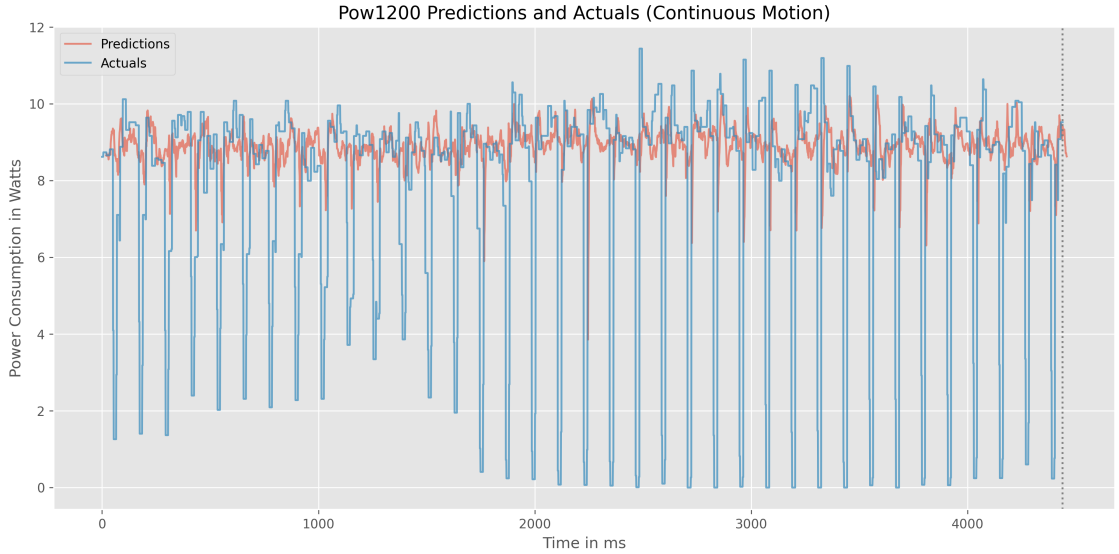

Pow700 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1
2
3
4
5
6
7
0
1000
2000
3000
4000

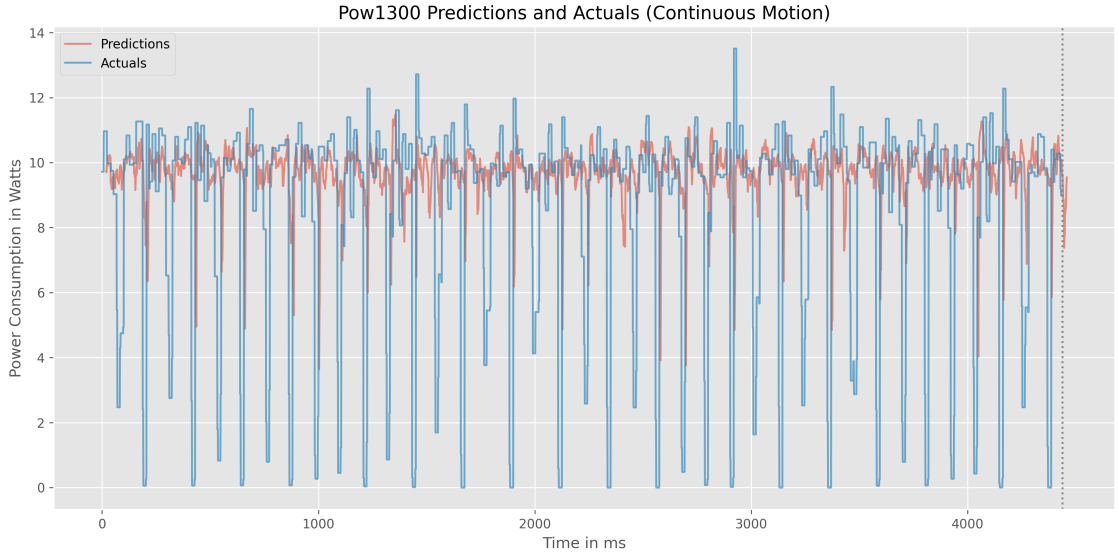

Pow800 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1000
2000
3000
4000
0
1
2
3
4
5
6
7
8

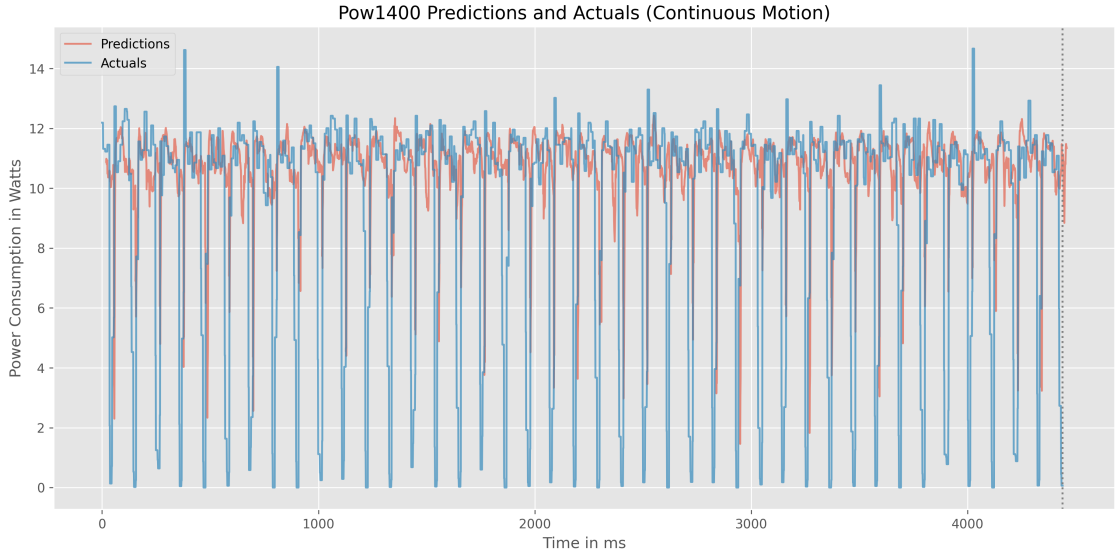

Pow900 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
1000
2000
3000
4000
0
2
4
6
8

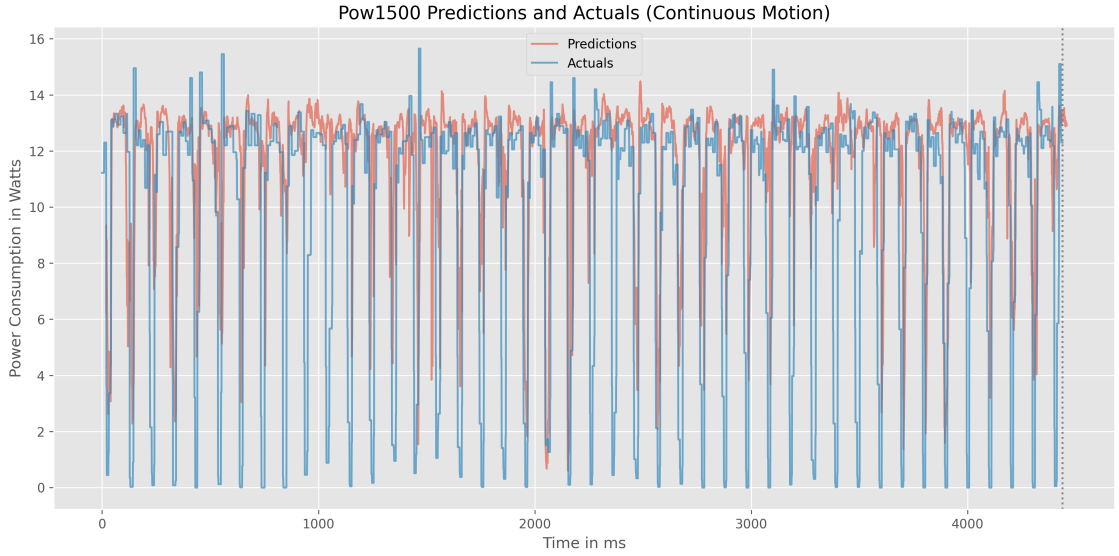

Pow1000 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
0
1000
2000
3000
4000
Time in ms

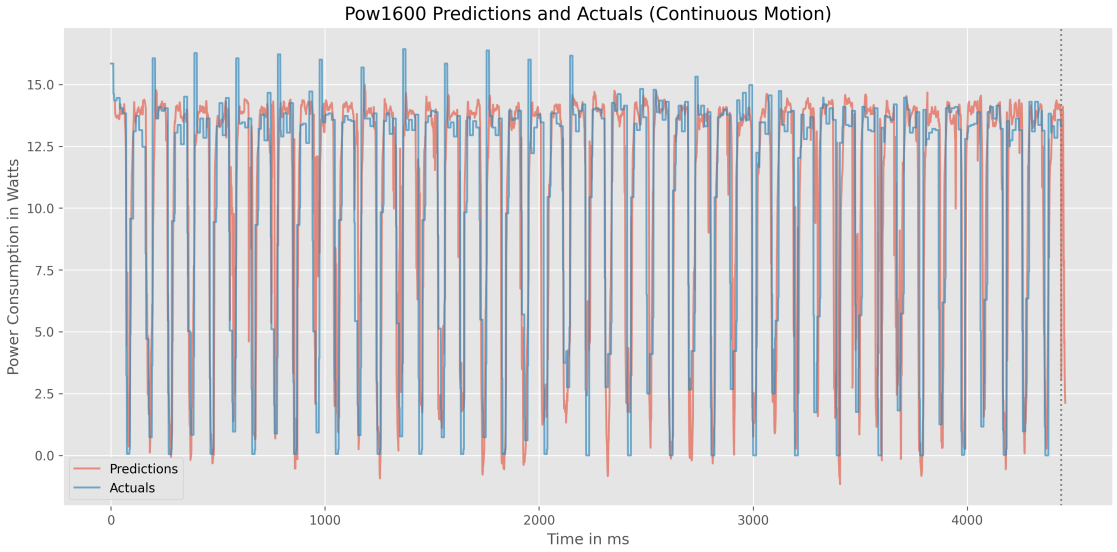

Pow1100 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
0
2
4
6
8
10
0
1000
2000
3000
4000
Time in ms

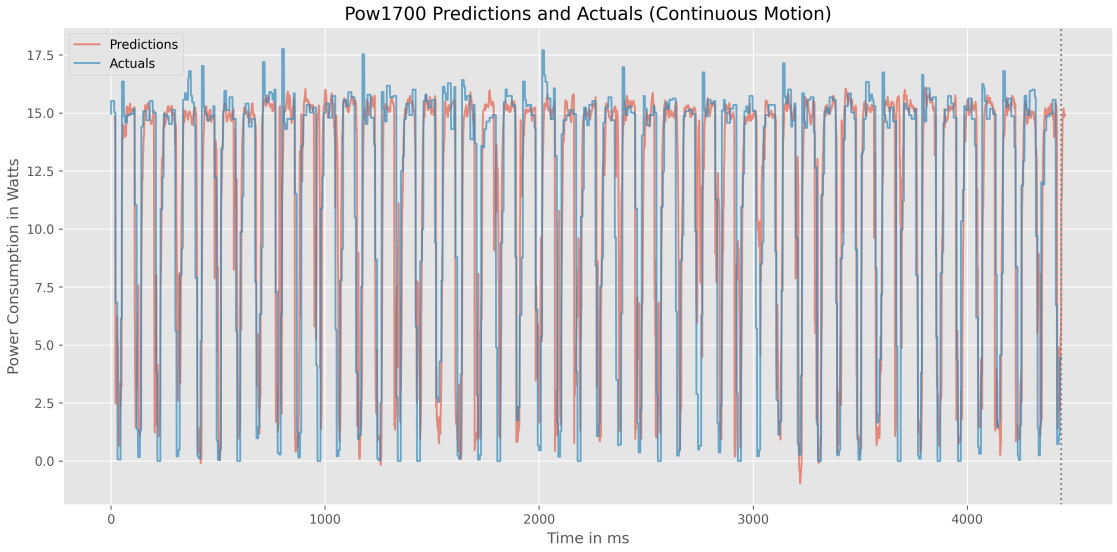

Pow1200 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
0
1000
2000
3000
4000

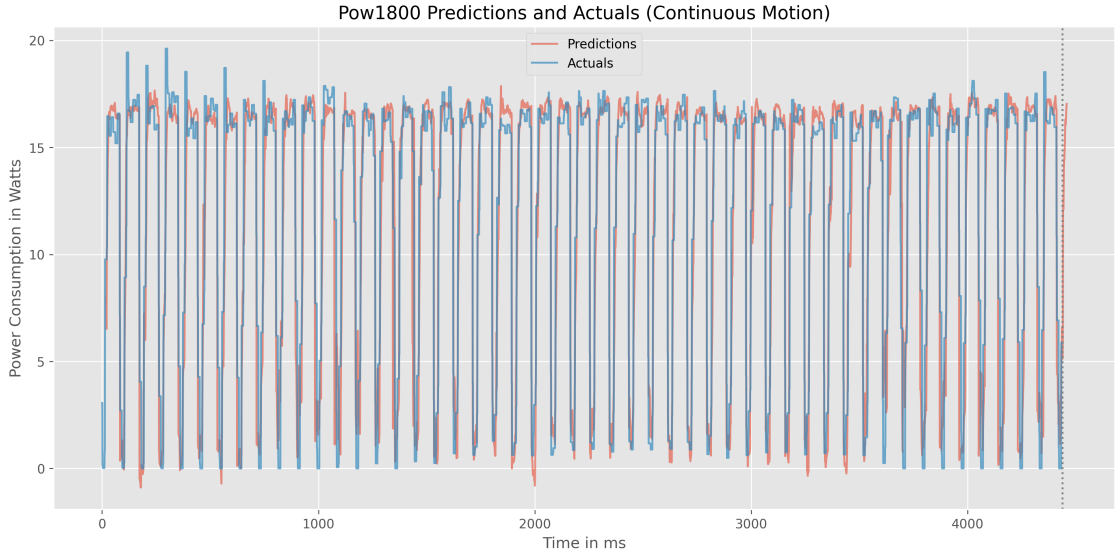

Pow1300 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
0
1000
2000
3000
4000

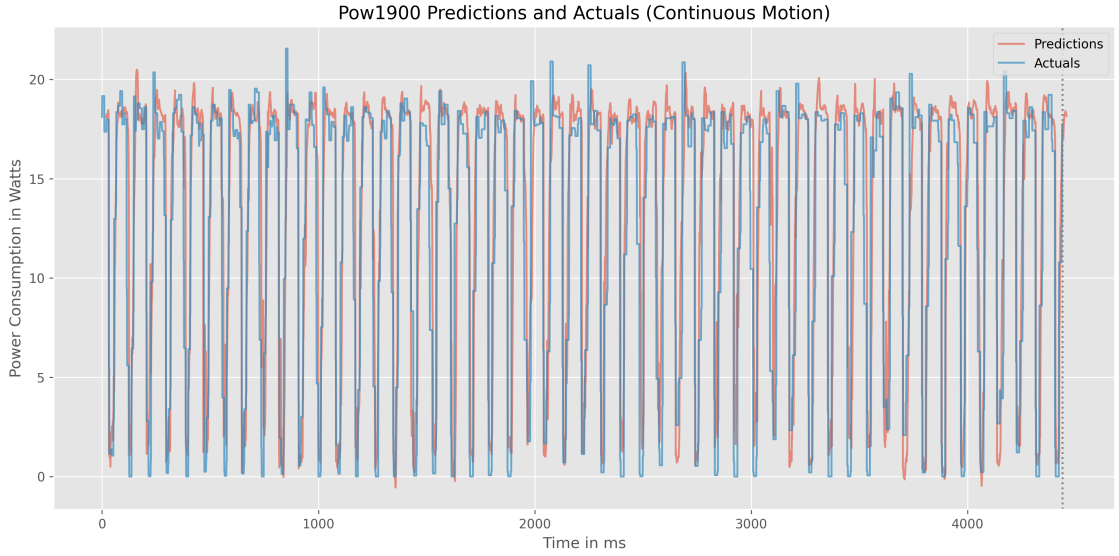

Pow1400 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
0
1000
2000
3000
4000

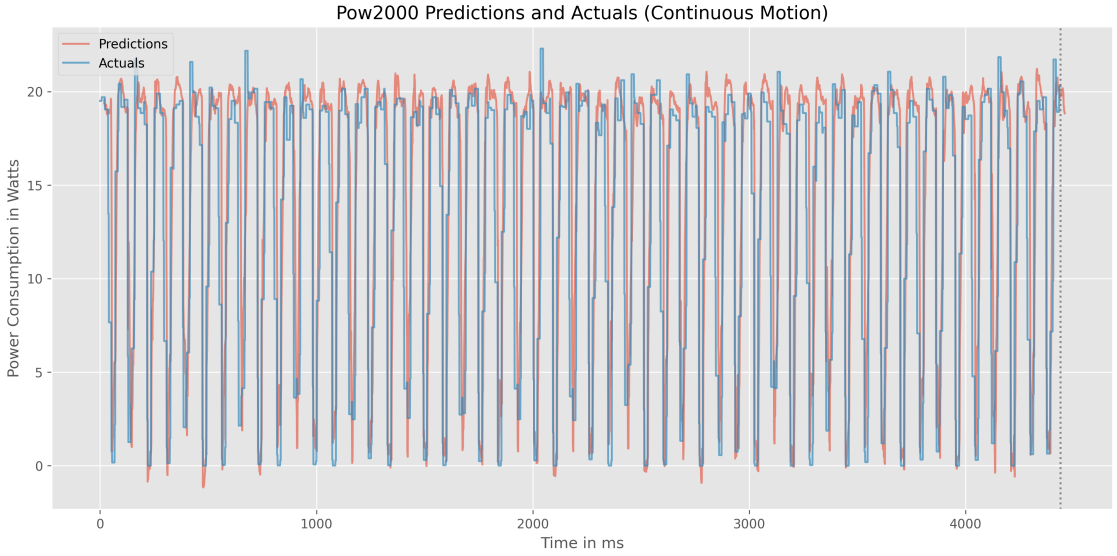

Pow1500 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
Time in ms
0
2
4
6
8
10
12
14
16
0
1000
2000
3000
4000


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


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


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


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


Pow2000 Predictions and Actuals (Continuous Motion)
Predictions
Actuals
Power Consumption in Watts
0
5
10
15
20
0
1000
2000
3000
4000
Time in ms