



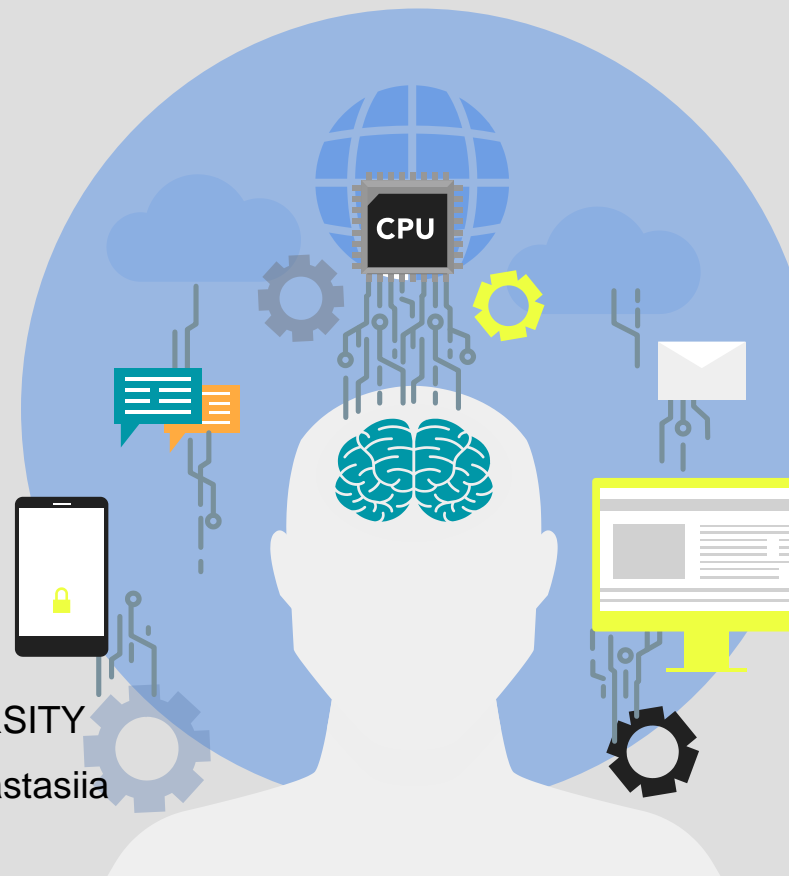
Co-funded by the  
Erasmus+ Programme  
of the European Union

KOİOS  
Research and Innovation Center of Excellence

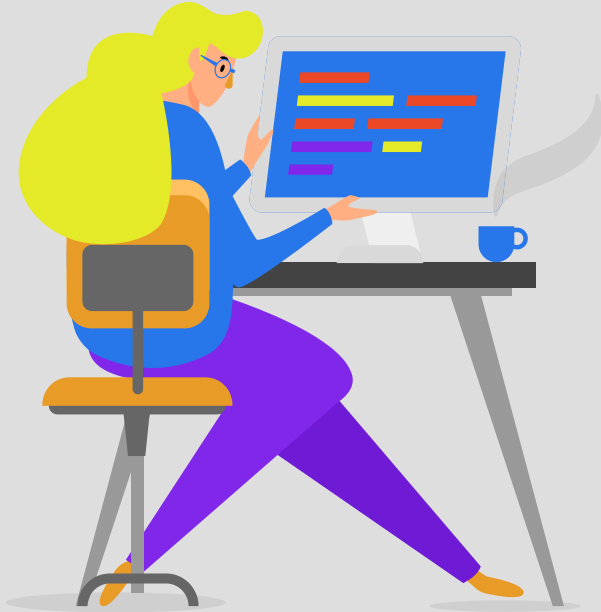
# CybPhys Training School



KRYVYI RIH NATIONAL UNIVERSITY  
Pasichna Yelyzaveta, Misiura Anastasiia



# Main steps



01

## Matrix Profile

What is it?  
What is it for?

02

## Installing

Installing the necessary  
apps

03

## Example 2

Execution and starting  
code

04

## Analysis

Analysis of the data  
obtained

05

## LSTM

What is it?  
What is it for?

06

## Conclusion

Summing up work

# *Matrix profile*

One method to find anomalies and trends within a time series is to perform a similarity join. Essentially, you compare snippets of the time series against itself by computing the distance between each pair of snippets.

**The Matrix Profile** is a relatively new, introduced in 2016, data structure for time series analysis.

The Matrix Profile has two primary components; a **distance profile and profile index**. The distance profile is a vector of **minimum Z-Normalized Euclidean Distances**. The profile index contains the index of its **first nearest-neighbor**. In other words, it is the location of its most similar sub-sequence

# *Sliding window approach*

The algorithms that compute the Matrix Profile use a sliding window approach. With a window size of  $m$ , the algorithm:

1. Computes the distances for the windowed sub-sequence against the entire time series
2. Sets an exclusion zone to ignore trivial matches
3. Updates the distance profile with the minimal values
4. Sets the first nearest-neighbor index

## Matrix Profile Algorithms

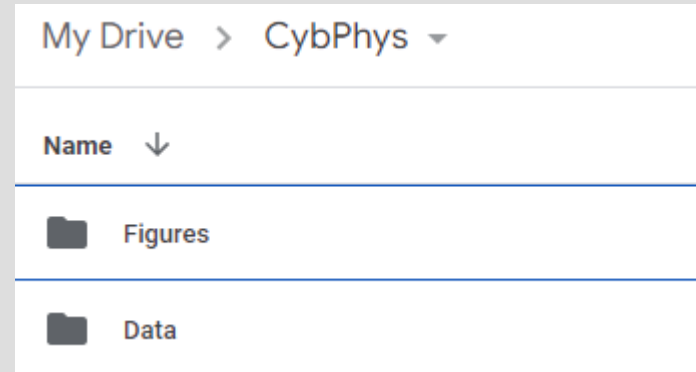
Algorithm	Description
Naive	Inefficient method to compute the Matrix Profile. This is essentially a "brute force" approach.
STAMP	This is one of the first algorithms that came out of Keogh's research group. It is an anytime algorithm.
STOMP	This algorithm is an exact ordered algorithm. It is significantly faster than STAMP.
SCRIMP++	This algorithm combines the anytime component of STAMP with the speed of STOMP.

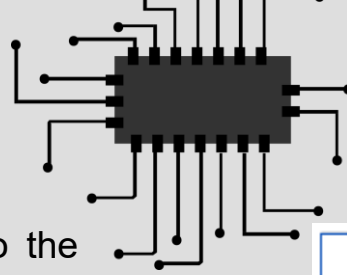
## Matrix Profile Mining Algorithms

Algorithm	Description
Top-K Motifs	Finds the top-K number of repeated patterns in the given time series using the Matrix Profile data structure.
Top-K Discords	Similar to motifs, but for anomalies.
FLUSS	Computes the corrected arc curve for the matrix profile that enables segmentation analysis.
Extract Regimes	Makes use of the output from FLUSS to extract the regimes.

## Prediction of measurement time series using sequence models at various time scales (part of AMPS21)

- ✓ Access Colab free Jupyter Notebook hosted environment, and log in using your Google account.
- ✓ Create a new Jupyter Notebook File > New Notebook Choose Google Drive as cloud storage for your notebook.
- ✓ Access Google Drive, sign in using your Google account and create two new folders: „Data” where the input measurement files will be initially stored and „Figures” for saving the graphical outputs of the analysis.





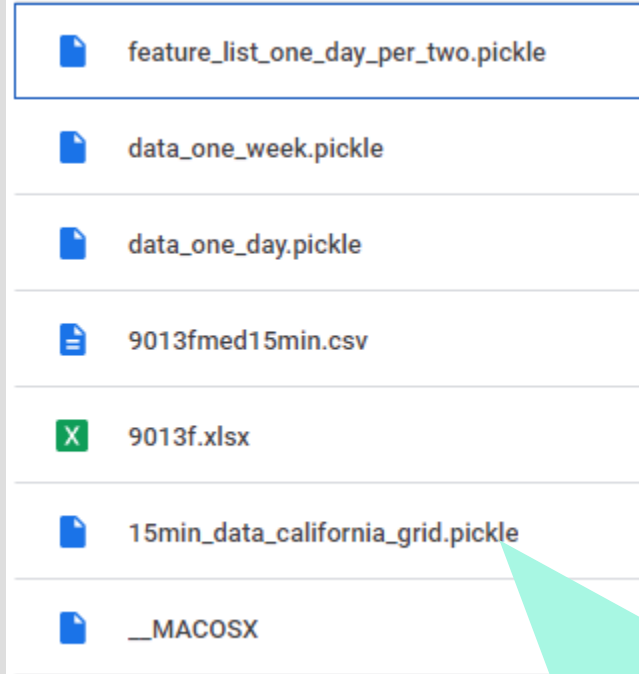
- ✓ Unzip and upload the following data files into the „Data” folder.
- ✓ Allow the notebook to access the files in your Drive.

```
✓ [7] from google.colab import drive  
3s drive.mount('/content/drive')
```

- ✓ Install the Matrix Profile Python package into the Colab environment.

```
✓ [8] !ls '/content/drive/MyDrive/CybPhys/Data'  
0s  
15min_data_california_grid.pickle  data_one_week.pickle  
9013fmed15min.csv                 feature_list_one_day_per_two.pickle  
9013f.xlsx                         __MACOSX  
data_one_day.pickle
```

```
✓ [9] !pip install -U matrixprofile  
3s
```



## ✓ Load packages

```
import pandas as pd
import pickle

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_absolute_percentage_error
import time
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Bidirectional
from keras.layers import ConvLSTM2D
```

## ✓ Opening files as

```
✓ 0s ▶ with open('/content/drive/MyDrive/CybPhys/Data/data_one_day.pickle', 'rb') as f:
    df1 = pickle.load(f)
```

```
✓ 0s ▶ df1
```



	timestamp	power
	timestamp	
	2020-09-01 00:00:00	09/01/2020 00:00:00 150.0
	2020-09-01 00:00:01	09/01/2020 00:00:01 150.0
	2020-09-01 00:00:02	09/01/2020 00:00:02 151.0
	2020-09-01 00:00:03	09/01/2020 00:00:03 148.0
	2020-09-01 00:00:04	09/01/2020 00:00:04 149.0

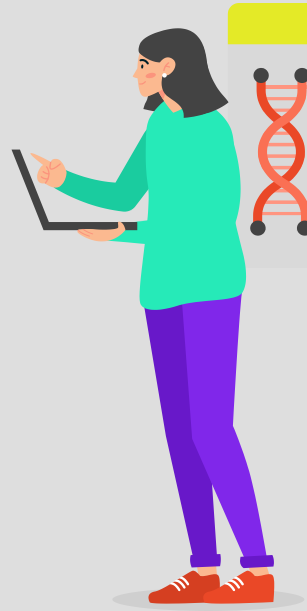
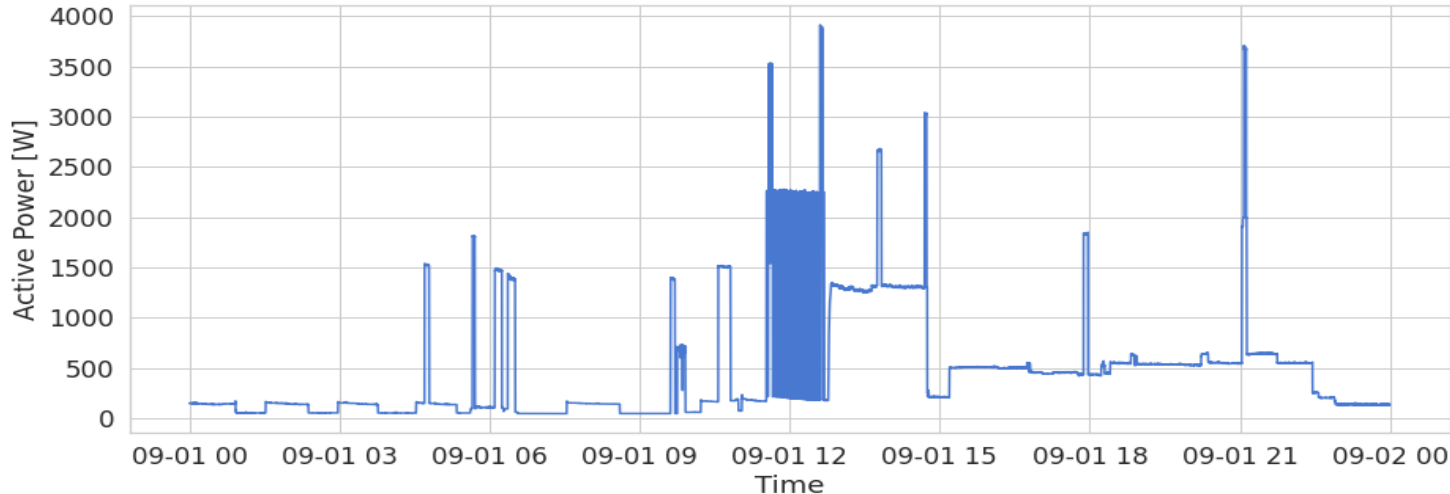


## Plotting active power versus time

```
[19] # Basic plot - one day
sns.set(style='whitegrid', palette='muted', font_scale=1.5)

plt.figure(figsize=(15, 6))
plt.plot(df1["power"][:10])

plt.ylabel("Active Power [W]"); plt.xlabel("Time")
plt.savefig('/content/drive/MyDrive/CybPhys/Figures/one_day.pdf', format='pdf', dpi=1000)
plt.show()
```

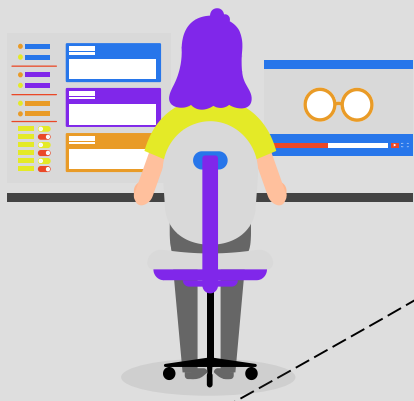


# Data preparation step

```
# split a univariate sequence into samples for model training
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

```
✓ [20] df1["power"][:5]
```

```
timestamp
2020-09-01 00:00:00    150.0
2020-09-01 00:00:05    148.0
2020-09-01 00:00:10    149.0
2020-09-01 00:00:15    151.0
2020-09-01 00:00:20    146.0
...
2020-09-01 23:59:35    135.0
2020-09-01 23:59:40    137.0
2020-09-01 23:59:45    131.0
2020-09-01 23:59:50    137.0
2020-09-01 23:59:55    131.0
Name: power, Length: 17280, dtype: float64
```



```
# define input sequence - HERE WE SET THE DECIMATION RATE FOR THE DATASET
raw_seq = df1["power"][:5]
# choose a number of time steps
n_steps = 12
# split into samples
X, y = split_sequence(raw_seq, n_steps)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

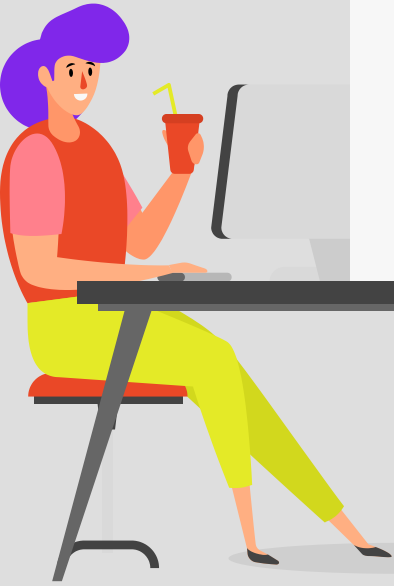
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], n_features))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], n_features))
```

# Model training

```
[22] # LSTM-1 model training
      start = time.time()

      # define model
      model = Sequential()
      model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
      model.add(Dense(1))
      model.compile(optimizer='adam', loss='mse')
      # fit model
      model.fit(X_train, y_train, epochs=30, verbose=1)

      end = time.time()
      print(end - start)
```

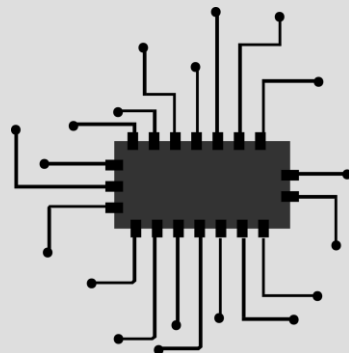


# Prediction for model and plotting

```
[23] # Prediction for LSTM-1 model
y_pred = model.predict(X_test)

print(mean_squared_error(y_test,y_pred))
print(mean_absolute_error(y_test,y_pred))
print(mean_absolute_percentage_error(y_test,y_pred))

162/162 [=====] - 1s 3ms/step
42671.197303711655
65.53393054638929
0.1392342823717484
```



```
[25] # True versus predicted
sns.set(style='whitegrid', palette='muted', font_scale=1.5)

plt.figure(figsize=(15, 6))
plt.plot(y_test[0:100])
plt.plot(y_pred[0:100], color='red')

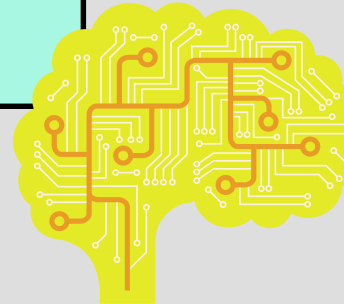
# show legend
plt.legend(['True', 'Predicted'])
plt.ylabel("Active Power [W]"); plt.xlabel("Time")
plt.savefig('/content/drive/MyDrive/CybPhys/Figures/true_vs_predicted.pdf', format='pdf', dpi=1000)
plt.show()
```

# Long Short-Term Memory

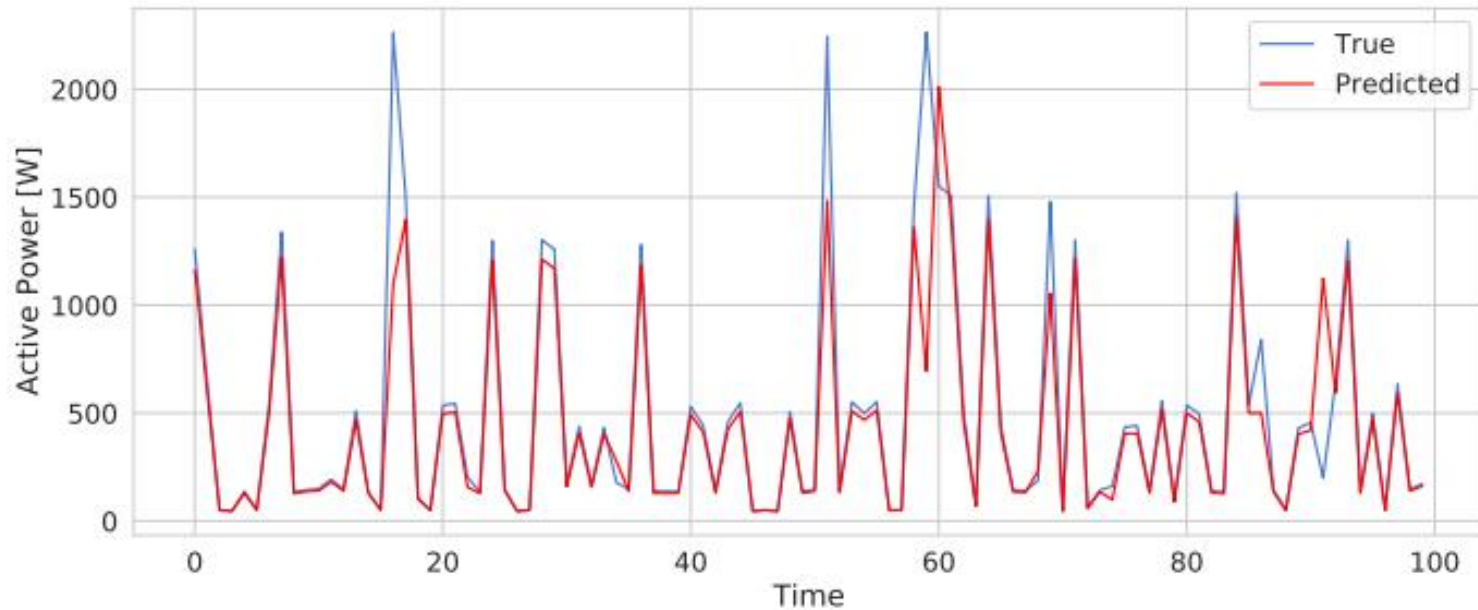
The Long Short-Term Memory network, or LSTM for short, is a type of recurrent neural network that achieves state-of-the-art results on challenging prediction problems.

The internal memory means outputs of the network are conditional on the recent context in the input sequence, not what has just been presented as input to the network.

LSTMs are explicitly designed to avoid the long-term dependency problem.



## Prediction result



# Conclusions

We learned how to detect anomalies and trends in time series using the Matrix profile, as well as predict data using the Long Short-Term Memory.





## GENERAL OVERVIEW

Critical infrastructures are assets or systems, which are essential for the maintenance of vital societal functions. Water, the electrical networks, telecommunication networks, distribution and transportation systems. Without these, other basic infrastructures (e.g., banking, hospitals, schools, tourism, etc.) cannot operate as intended. Critical infrastructures provide the foundation on which communities are built and when properly functioning, they enable economic growth and social well-being.

As urbanization increases, critical infrastructures worldwide are expanding and are becoming more complex, necessitating greater efficiency and improved capabilities in order to sustain their effective operation. Equipment failures are also occurring more frequently as large segments are outdated. Such failures may lead to serious degradation in performance or, even worse, to cascading overall system failure and breakdown.


Moreover, the safety and security of critical infrastructure systems against malicious attacks (such as denial-of-service) and natural disasters are becoming crucial issues

for citizens, businesses, and governments who expect that these infrastructures will provide uninterrupted service day and under any circumstances. Unanticipated events can also occur (facilities, earthquakes etc.) which create extraordinary conditions requiring immediate response to prevent fatalities and limit damage.

The problem of monitoring, control, management and security of critical infrastructures systems (CIS) will become even more challenging in the future. Currently, an important proportion of the world population lives in urban areas and it is predicted that an even more significant percent in the developing and developed world will be urbanized in the near future. However, existing critical infrastructures were not designed to accommodate such enormous demands. Moreover, due to wide-ranging deregulation, the use of renewable energy and a massive expansion of wireless communications, critical infrastructures, as well as the associated data, software and management systems, are becoming increasingly more heterogeneous, distributed, and inter-dependent making the seamless integration of all components that make up the CIS an immense challenge.


### INTELLIGENT SYSTEMS AND NETWORKS

Monitoring  
Control  
Management  
Security




Big Data  
↓  
Smart  
Decisions

**CRITICAL INFRASTRUCTURE SYSTEMS**



Heterogeneous  
Interdependent  
Interconnected


**INFORMATION AND COMMUNICATION TECHNOLOGIES**





Sensors  
Actuators  
Internet of Things

Risks, Faults, Attacks

Big Data





  
 Imperial College London

Energy and Power  
 Water Systems/Env  
 Intelligent Transp  
 Telecommunication  
 Emergency Manage

1 Pa  
 Te

1 Panepistimiou A  
 Tel



