PRESENTATION

# NOWCASTING

DATA SCIENCE
PROJECT2
OPEN PROJECT

by กะหล่ำปลีน้ำปลา

# กะหล่ำปลีน้ำปลา Member's

**Supanart Barnsongkit**

6230522621

**Pon-ek Tangmunchittham**

6231341521

**Punthat Siriwan**

6231342121

**Tanapon Suwanmanee**

6230218921

กะหล่ำปลีน้ำปลา

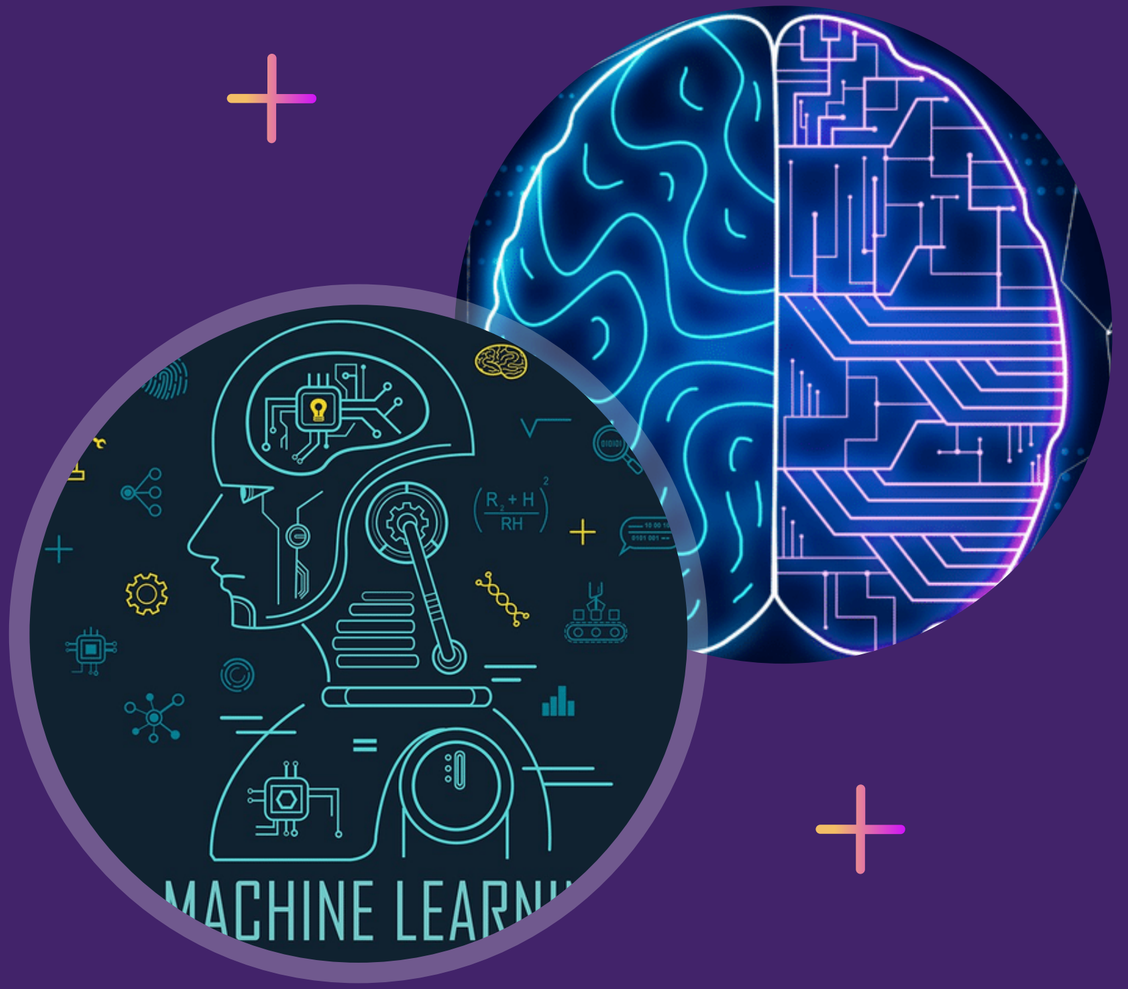
# Nowcasting

Now + Forecasting
&
(Real-time Prediction Dashboard)

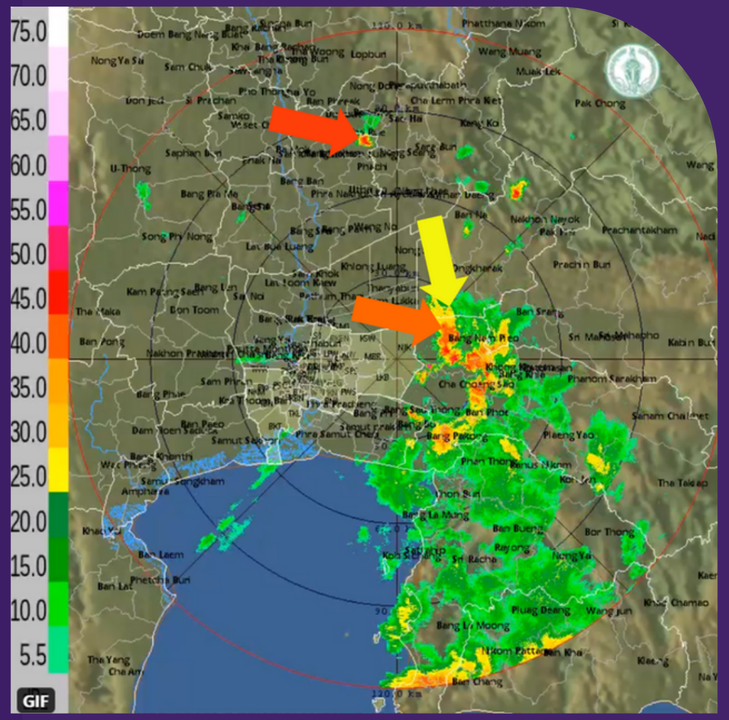
2022 PRESENTATION

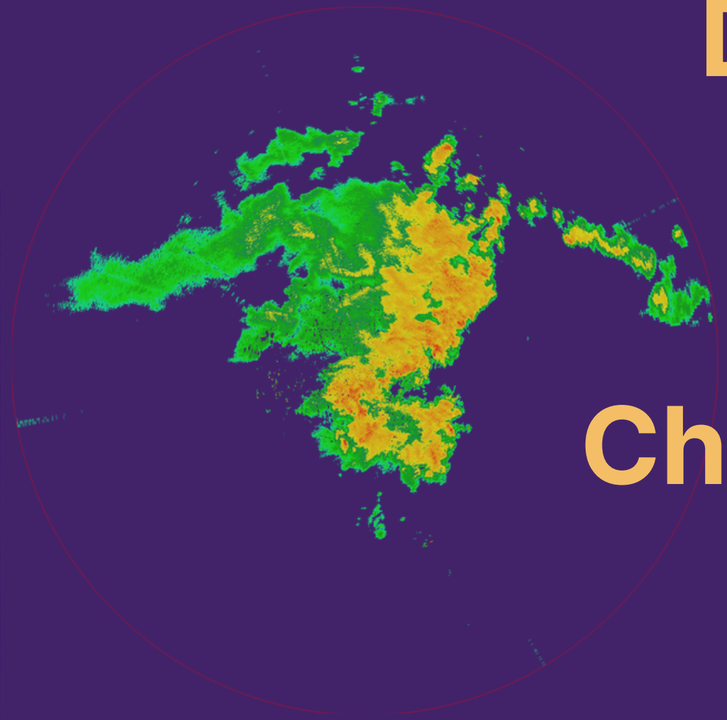
กะหล่ำปลีน้ำปลา

# Module 1
# Machine Learning



กะหล่ำปลีน้ำปลา



# Module 1: ML
## Data Preparation

- 86,083 images
- RGB image
- RGB image -> 1 channel image

## Challenges in ML part

- high resolution image
- how to visualize output from a model
- large dataset
- not enough GPU ram to train high resolution image
- 80000++ images time to train 1 epoch is 9 hr

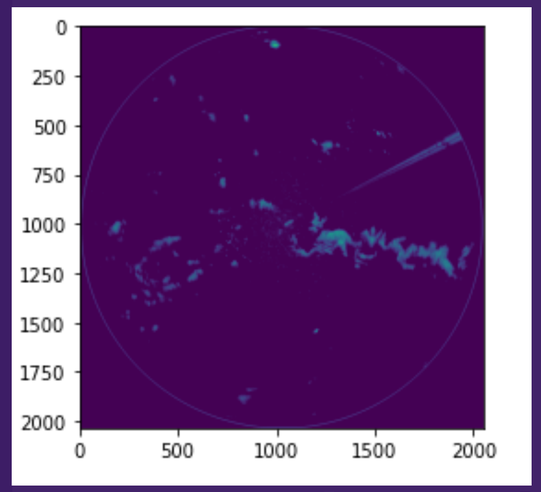
Image → Model → Image

กะหล่ำปลีน้ำปลา

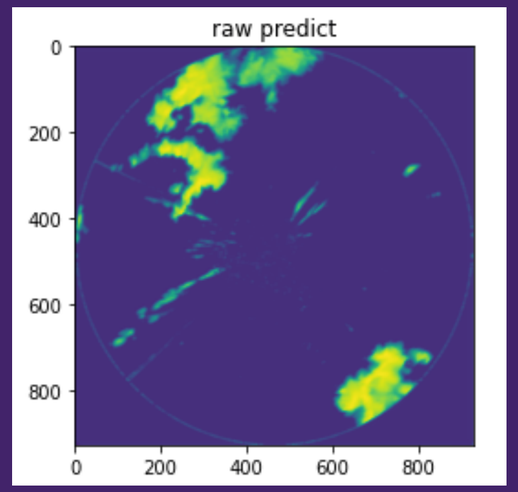
# Module 1: ML

## Data pre-processing

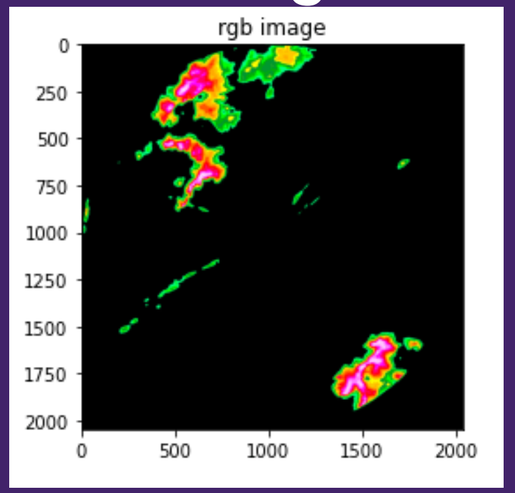
- change background color to black
- RGB image -> 1 channel image
- normalize image



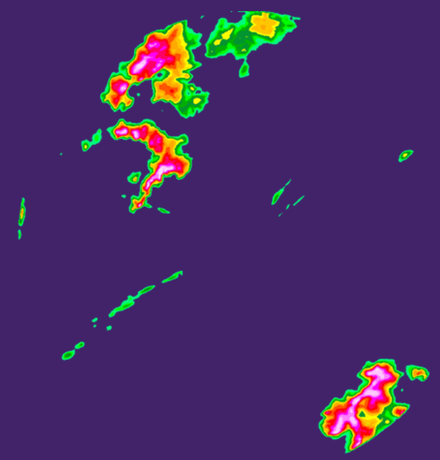

Input 1 channel image



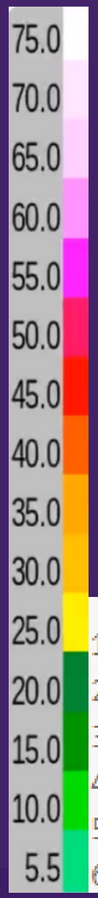

Raw output




RGB output



RGBA output




## Data post-processing

- process output from model
- 1 channel image -> RGB image
- RGB image -> RGBA image

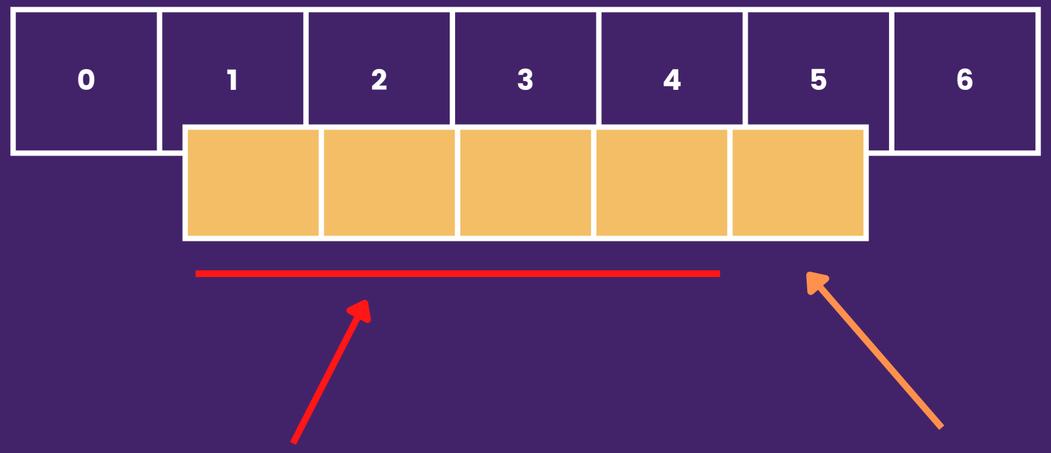
```python
def rgb2intensity(pixel):
  pixel = tuple(pixel)
  if pixel == (0,0,0):
    return 0.0
  elif pixel == (0,255,128):
    return 5.0
```

```python
def intensity2rgb(intensity):
  value = {
      0: [0,0,0],
      5: [0,255,128],
      10: [0,255,0],
      15: [0,175,0],
```

กะหล่ำปลีน้ำปลา

# Module 1: ML
# Training Step

**Example**
**4 step_in , 1 step_out**




**4 timeframe as Input X**

**1 timeframe as output y**

- Custom data generator
- Using sliding window to feed data to training model
- No shuffle in train and val set

```
1 train_rain_dataset[0][0].shape
```

```
(16, 928, 928, 4)
```

**Example Input Shape w/ rainnet model**

กะหล่ำปลีน้ำปลา

# Module 1: ML
# Model 1 convLSTM

## ConvLSTM_pytorch

This file contains the implementation of Convolutional LSTM in PyTorch made by me and DavideA.

We started from this implementation and heavily refactored it add added features to match our needs.

Please note that in this repository we implement the following dynamics:

$$
\begin{aligned}
\mathbf{i_t} &= \text{Sigmoid}\left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xi}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{hi}}\right) + \mathbf{b_i}\right) \\
\mathbf{f_t} &= \text{Sigmoid}\left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xf}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{hf}}\right) + \mathbf{b_f}\right) \\
\mathbf{o_t} &= \text{Sigmoid}\left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xo}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{ho}}\right) + \mathbf{b_o}\right) \\
\mathbf{g_t} &= \text{Tanh} \quad \left(\text{Conv}\left(\mathbf{x_t}; \mathbf{w_{xg}}\right) + \text{Conv}\left(\mathbf{h_{t-1}}; \mathbf{w_{hg}}\right) + \mathbf{b_g}\right) \\
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \mathbf{g_t} \\
\mathbf{h_t} &= \mathbf{o_t} \odot \text{Tanh}(\mathbf{c_t})
\end{aligned}
,
$$

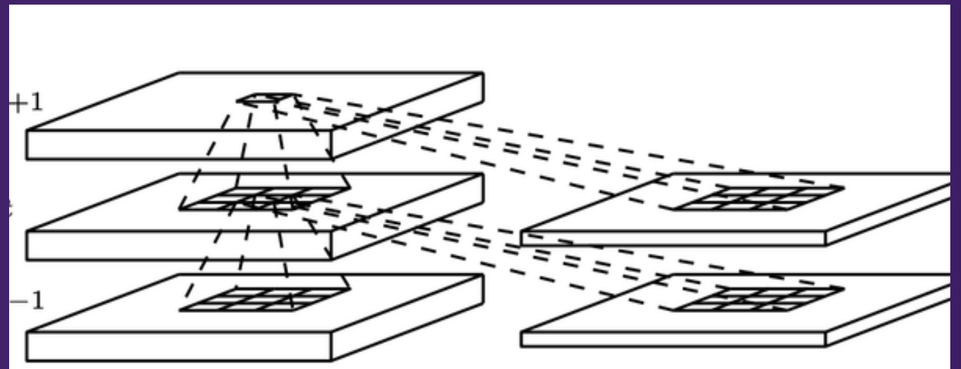



Figure 2: Inner structure of ConvLSTM

กะหล่ำปลีน้ำปลา

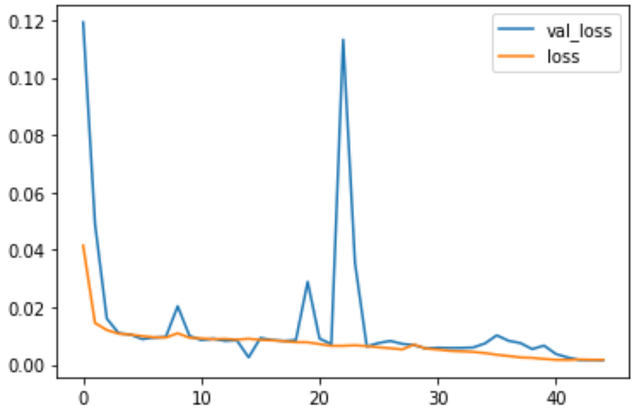
# Module 1: ML
# Model 1 convLSTM

## Input (B x 300 x 300 x 4) -> Model ->  B x 300 x 300 x 1

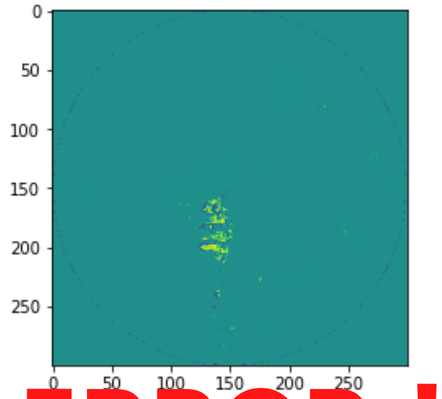
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_lstm2d_4 (ConvLSTM2D)  (None, None, 300, 300, 6  815616
                             4)

 batch_normalization_4 (Batc  (None, None, 300, 300, 6  256
 hNormalization)             4)

 conv_lstm2d_5 (ConvLSTM2D)  (None, None, 300, 300, 3  602240
                             2)

 batch_normalization_5 (Batc  (None, None, 300, 300, 3  128
 hNormalization)             2)

 conv_lstm2d_6 (ConvLSTM2D)  (None, None, 300, 300, 3  401536
                             2)

 batch_normalization_6 (Batc  (None, None, 300, 300, 3  128
 hNormalization)             2)

 conv_lstm2d_7 (ConvLSTM2D)  (None, 300, 300, 32)      401536

 batch_normalization_7 (Batc  (None, 300, 300, 32)     128
 hNormalization)

 conv2d_1 (Conv2D)           (None, 300, 300, 1)       33

=================================================================
Total params: 2,221,601
Trainable params: 2,221,281
Non-trainable params: 320
_________________________________________________________________
None
```

```
Epoch 1/20
4859/4859 [==============================] - 7390s 2s/step - loss: 0.1423 - val_loss: 0.1348
Epoch 2/20
4859/4859 [==============================] - 7377s 2s/step - loss: 0.1343 - val_loss: 0.1348
Epoch 3/20
4859/4859 [==============================] - 7377s 2s/step - loss: 0.1342 - val_loss: 0.1348
Epoch 4/20
4859/4859 [==============================] - 7377s 2s/step - loss: 0.1342 - val_loss: 0.1348
Epoch 5/20
4859/4859 [==============================] - 7382s 2s/step - loss: 0.1342 - val_loss: 0.1348
Epoch 6/20
2487/4859 [==============>..............] - ETA: 54:01 - loss: 0.1337
```

```
1 fig, ax1 = plt.subplots(1,1)
2 line1, = ax1.plot(history.history["val_loss"],label='val_loss')
3 line2, = ax1.plot(history.history["loss"],label='loss')
4 ax1.legend(handles=[line1, line2])
5 plt.show()
```

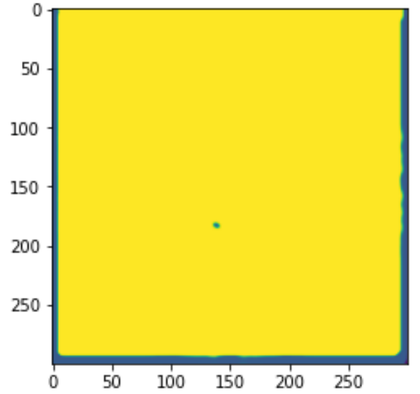



```
1 visulize_output(np.squeeze(y))
```




**ERROR !!**

```
1 visulize_output(np.squeeze(y_pred))
```



กะหล่ำปลีน้ำปลา

# Module 1: ML
# Model 2 optical flow

## rainymotion

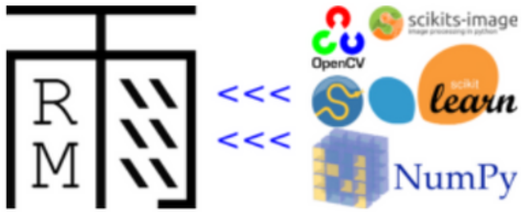
`rainymotion`: Python library for radar-based precipitation nowcasting based on optical flow techniques

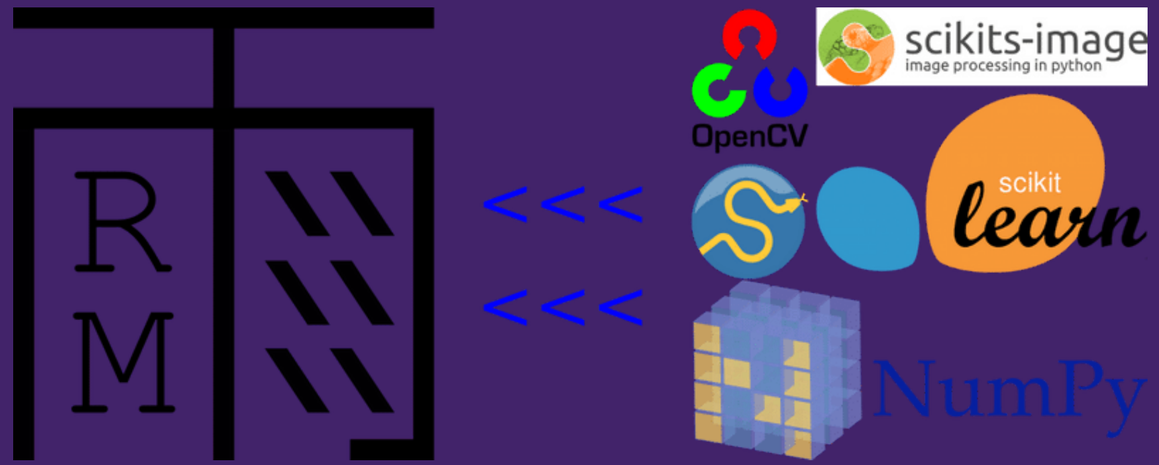
### Idea

The main idea of the `rainymotion` library is to provide an open baseline solution for radar-based precipitation nowcasting.

### Development

`rainymotion` is based only on free and open source software -- we tried to make a clue between the best scientific libraries to make them work together for providing reliable precipitation nowcasts.



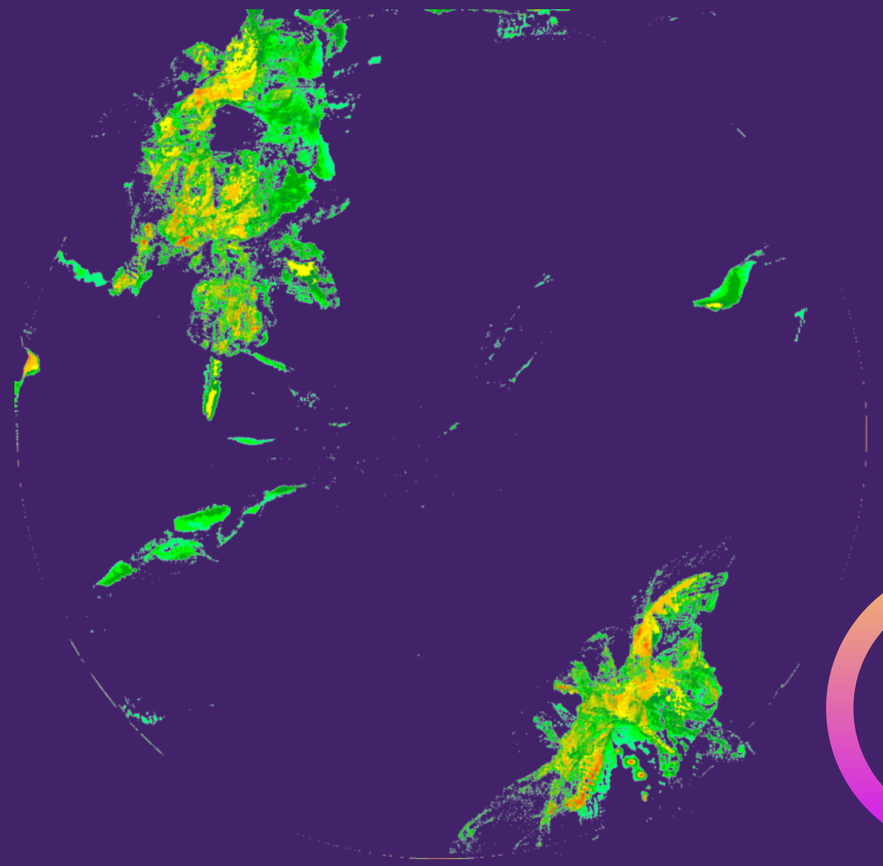



กะหล่ำปลีน้ำปลา

# Module 1: ML
# Model 2 optical flow

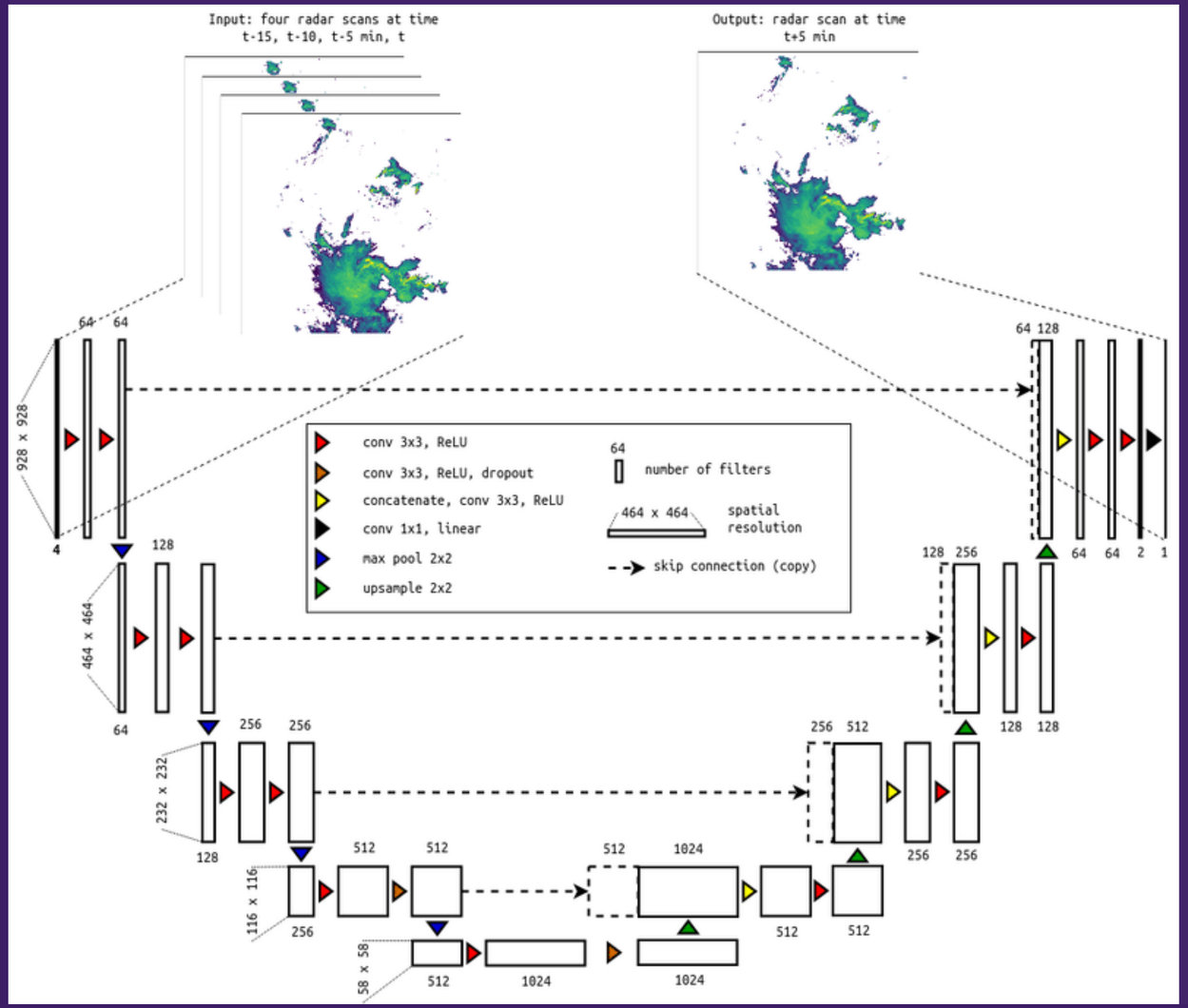
## 4 time frame -> model -> 12 time frame

```python
def predict_optical_flow(filenames):

  img1 = Image.open(f'/content/colab_directory/train_radarNJ/bkk_radar_images_all/{filenames[0]}.png').convert('RGB')
  img2 = Image.open(f'/content/colab_directory/train_radarNJ/bkk_radar_images_all/{filenames[1]}.png').convert('RGB')
  img3 = Image.open(f'/content/colab_directory/train_radarNJ/bkk_radar_images_all/{filenames[2]}.png').convert('RGB')
  img4 = Image.open(f'/content/colab_directory/train_radarNJ/bkk_radar_images_all/{filenames[3]}.png').convert('RGB')

  tmp_x = []

  tmp_x.append(pre_processing_optical_flow(img1))
  tmp_x.append(pre_processing_optical_flow(img2))
  tmp_x.append(pre_processing_optical_flow(img3))
  tmp_x.append(pre_processing_optical_flow(img4))

  X = np.array(tmp_x)

  model = rainymotion.models.Dense()

  # load the data using your custom DataLoader function
  model.input_data = X

  nowcasts = model.run()

  return nowcasts
```



กะหล่ำปลีน้ำปลา

# Module 1: ML
# Model 3 Auto-Encoder: RainNet




## RainNet: a convolutional neural network for radar-based precipitation nowcasting

RaiNet

### Brief description

Here we introduce RainNet -- a convolutional neural network for radar-based precipitation nowcasting. RainNet was trained to predict continuous precipitation intensities at a lead time of five minutes, using several years of quality-controlled weather radar composites provided by the German Weather Service (DWD).

The source code of the RainNet model written using *Keras* functional API is in the file `rainnet.py`.

The pretrained instance of `keras` `Model` for RainNet, as well as RainNet's pretrained weights are available on Zenodo:

DOI 10.5281/zenodo.3630429

# Module 1: ML
# Model 3 Auto-Encoder: RainNet

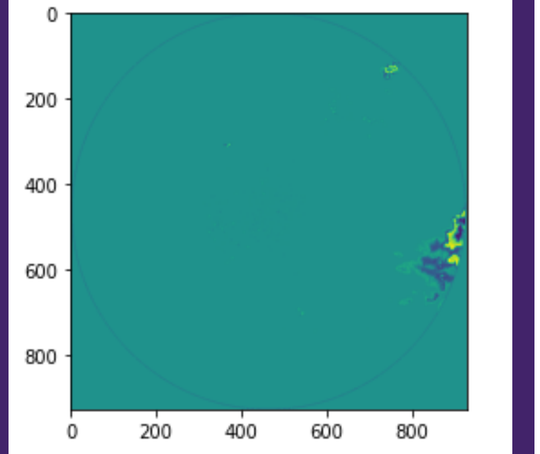
กะหล่ำปลีน้ำปลา

**Input (B x 928 x 928 x 4) -> Model ->  B x 928 x 928 x 1**

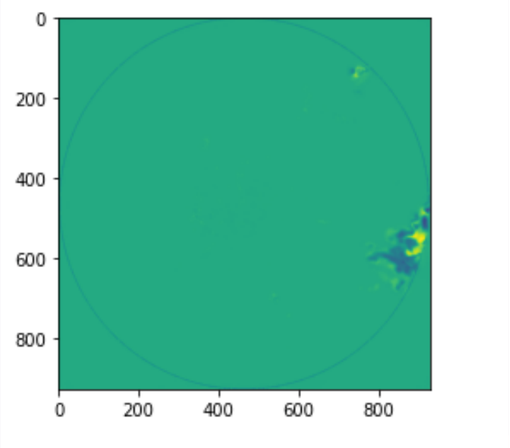
```
up_sampling2d_5 (UpSampling2D)  (None, 928, 928, 128  0          ['activation_27[0][0]']
                                )

concatenate_5 (Concatenate)     (None, 928, 928, 192  0          ['up_sampling2d_5[0][0]',
                                )                                 'activation_13[0][0]']

conv2d_28 (Conv2D)              (None, 928, 928, 64)  110656     ['concatenate_5[0][0]']

activation_28 (Activation)      (None, 928, 928, 64)  0          ['conv2d_28[0][0]']

conv2d_29 (Conv2D)              (None, 928, 928, 64)  36928      ['activation_28[0][0]']

activation_29 (Activation)      (None, 928, 928, 64)  0          ['conv2d_29[0][0]']

conv2d_30 (Conv2D)              (None, 928, 928, 2)   1154       ['activation_29[0][0]']

conv2d_31 (Conv2D)              (None, 928, 928, 1)   3          ['conv2d_30[0][0]']

==================================================================================================
Total params: 31,380,613
Trainable params: 31,380,613
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
[ ] history = model.fit(x=train_rain_dataset,validation_data=val_rain_dataset,epochs=5,verbose=1,callbacks=cb_list)

Epoch 1/5
2995/2995 [==============================] - ETA: 0s - loss: 0.0032WARNING:tensorflow:Can save best model only wi
2995/2995 [==============================] - 1449s 478ms/step - loss: 0.0032 - val_loss: 0.0011
Epoch 2/5
2995/2995 [==============================] - ETA: 0s - loss: 0.0029WARNING:tensorflow:Can save best model only wi
2995/2995 [==============================] - 1432s 478ms/step - loss: 0.0029 - val_loss: 0.0012
Epoch 3/5
2995/2995 [==============================] - ETA: 0s - loss: 0.0033WARNING:tensorflow:Can save best model only wi
2995/2995 [==============================] - 1432s 478ms/step - loss: 0.0033 - val_loss: 0.0012
Epoch 4/5
2995/2995 [==============================] - ETA: 0s - loss: 0.0024WARNING:tensorflow:Can save best model only wi
2995/2995 [==============================] - 1432s 478ms/step - loss: 0.0024 - val_loss: 0.0013
Epoch 5/5
2995/2995 [==============================] - ETA: 0s - loss: 0.0023WARNING:tensorflow:Can save best model only wi
2995/2995 [==============================] - 1432s 478ms/step - loss: 0.0023 - val_loss: 0.0013
```

```
1 visulize_output(np.squeeze(y))
```



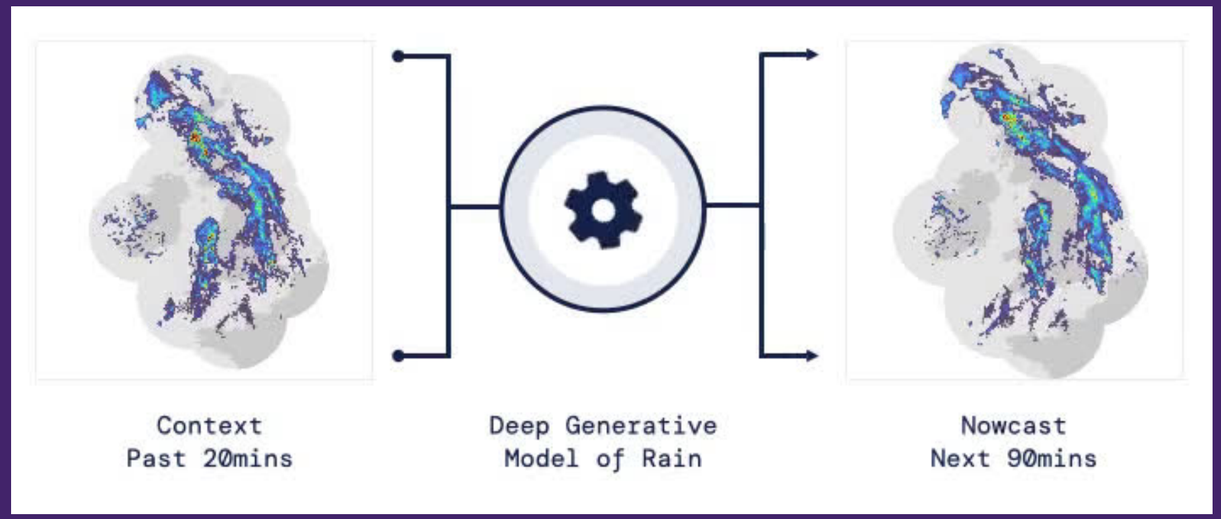

```
1 visulize_output(np.squeeze(y_pred))
```

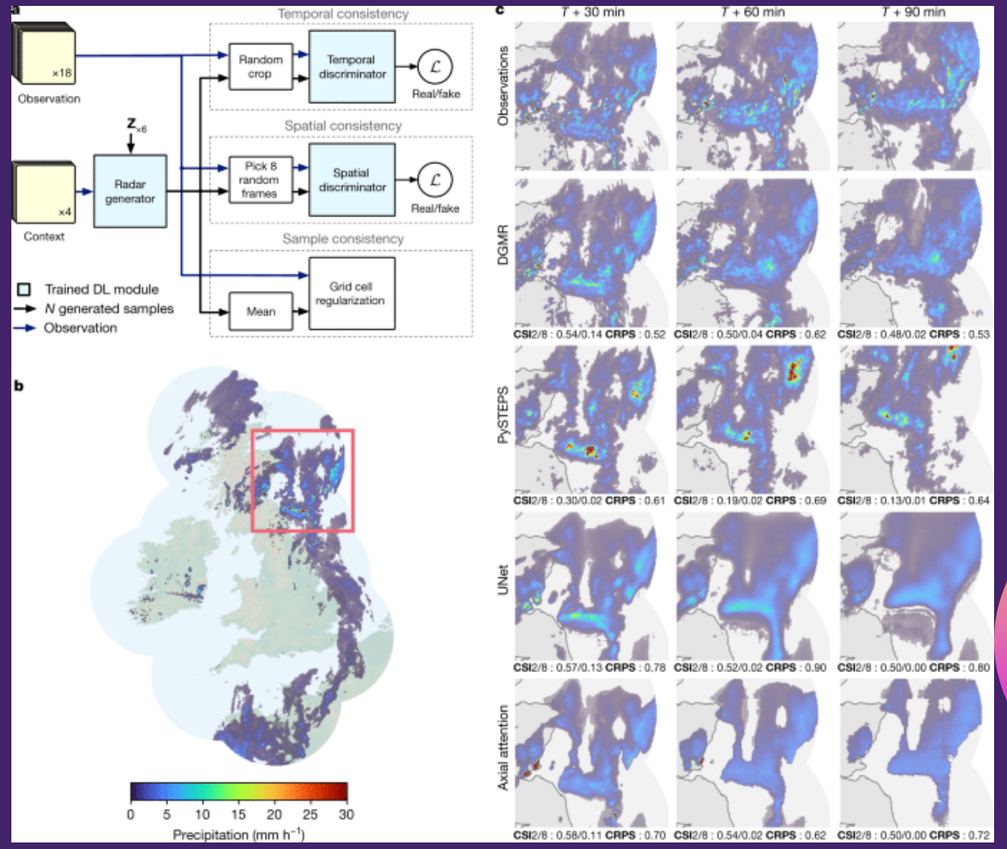


กะหล่ำปลีน้ำปลา

# Module 1: ML
# Other model

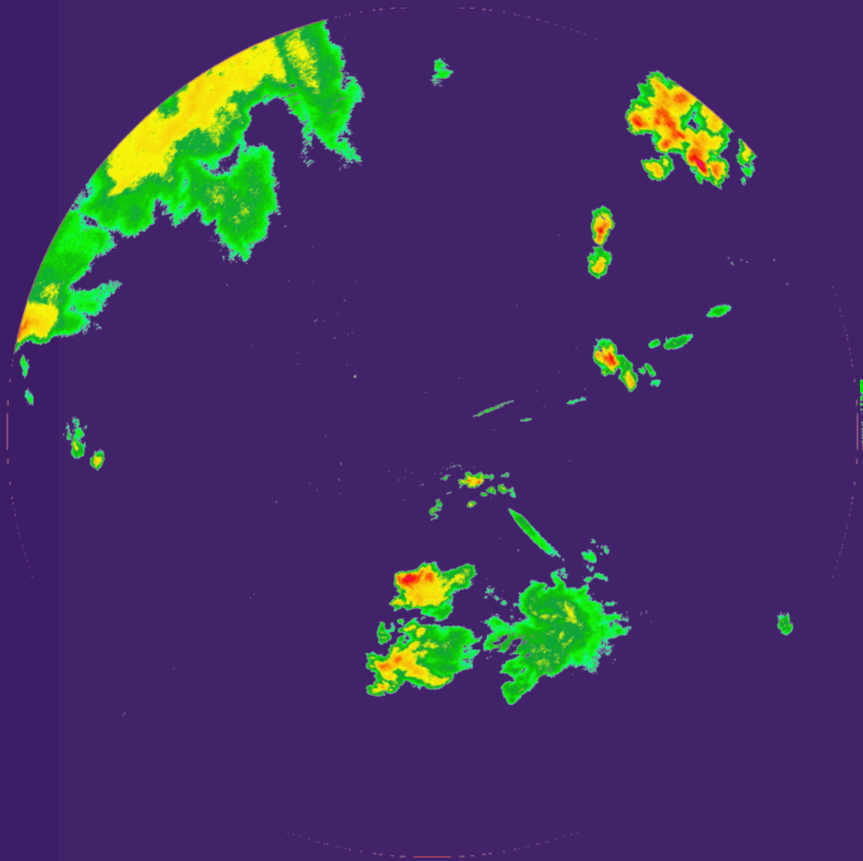
## Deep Generative Model of Radar (DGMR)



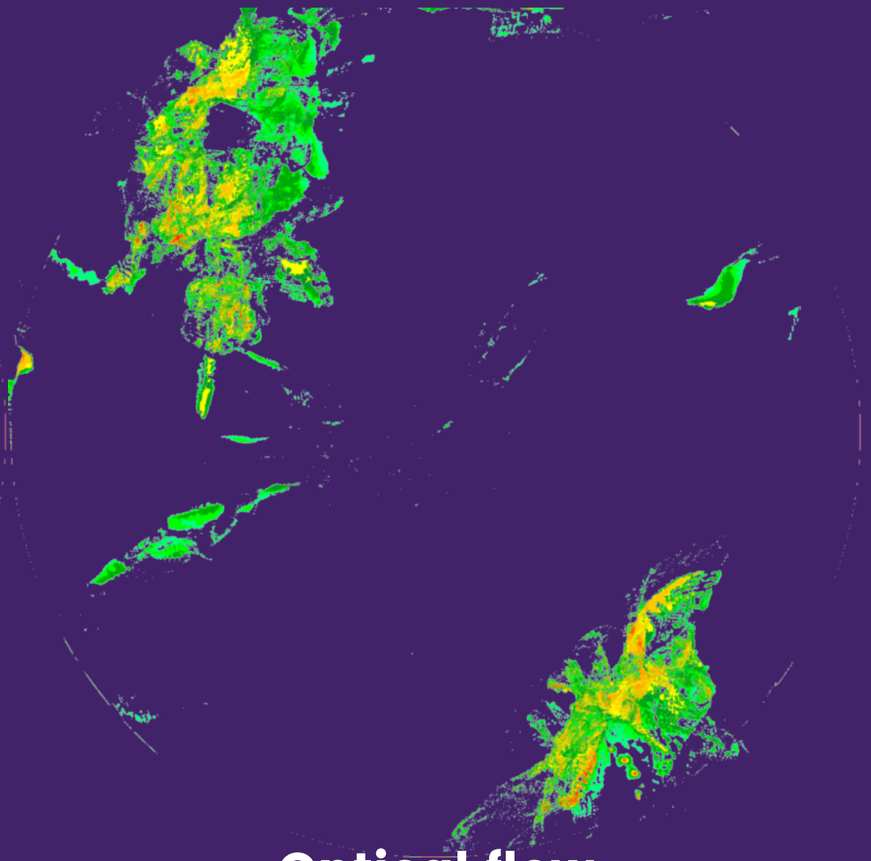



กะหล่ำปลีน้ำปลา

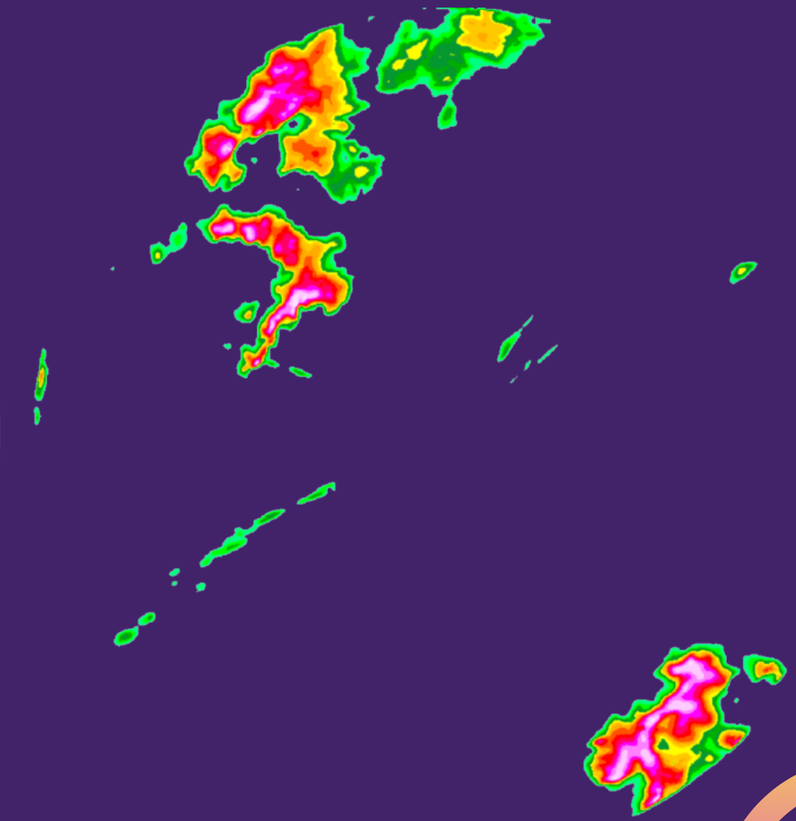
# Module 1: ML
# Output Visualization



Ground Truth



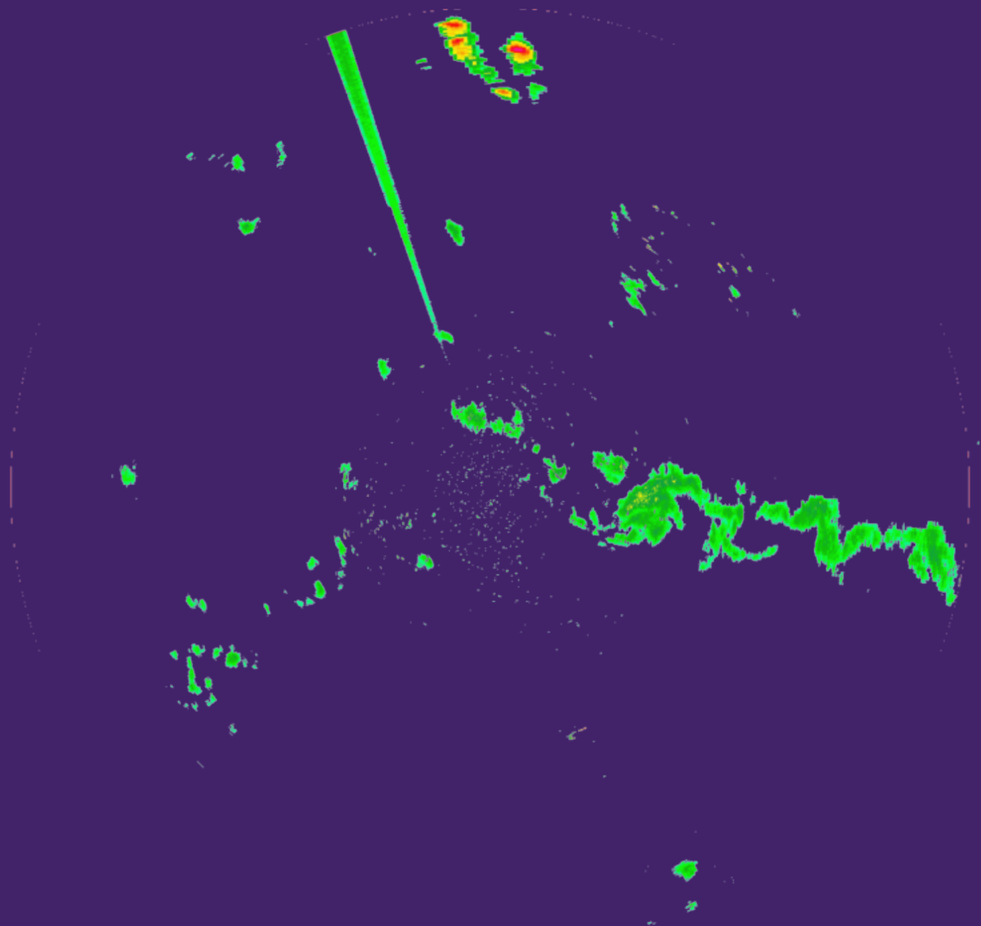
Optical flow



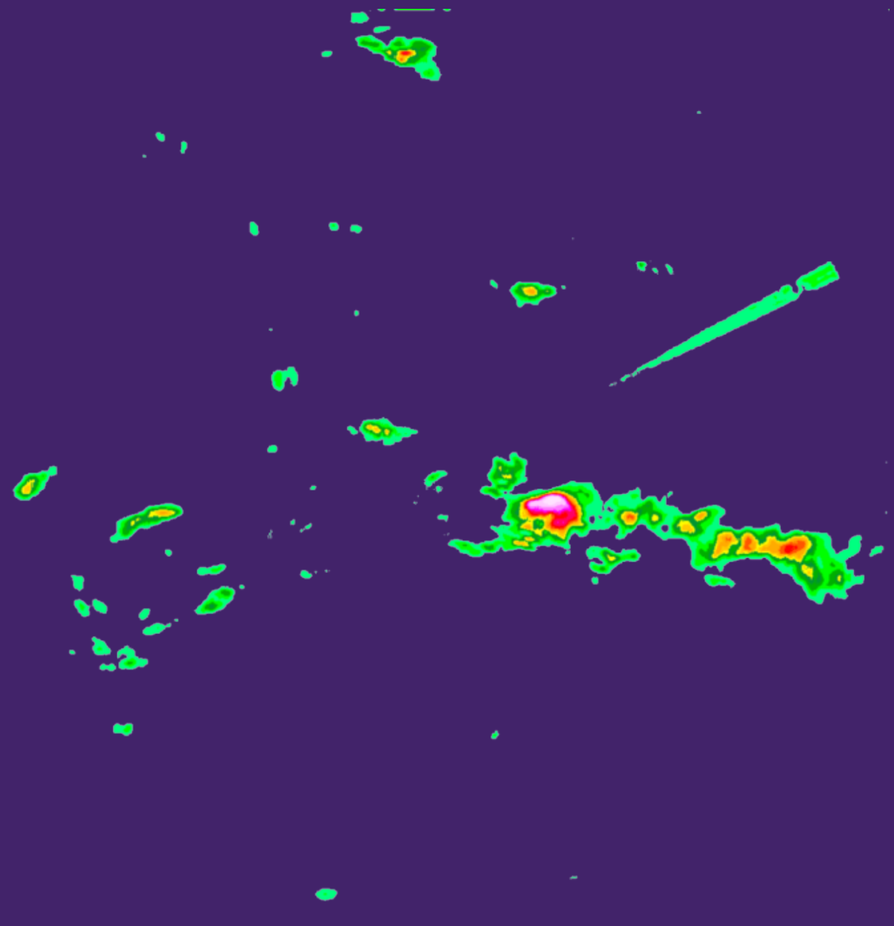
rainnet

กะหล่ำปลีน้ำปลา

# Module 1: ML
# Output Visualization



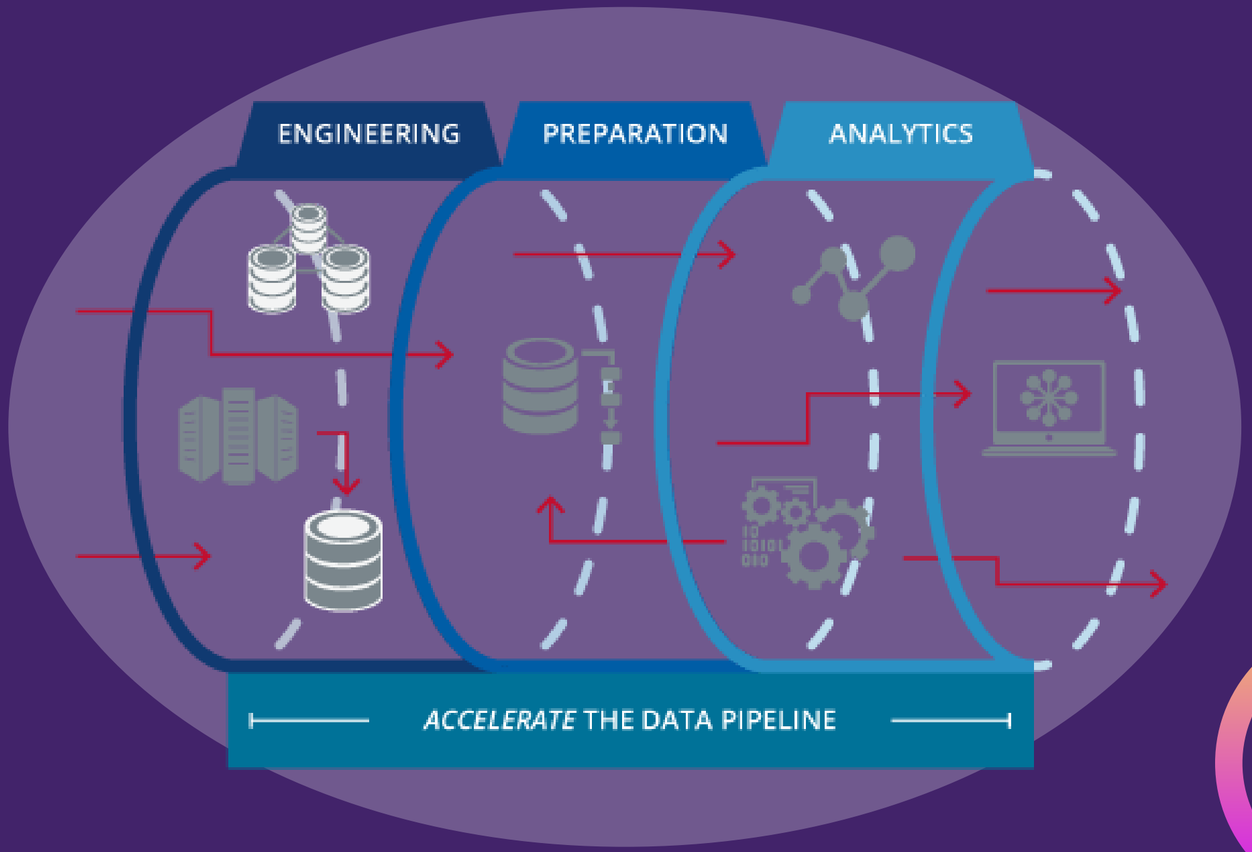
Ground Truth



rainnet

กะหล่ำปลีน้ำปลา

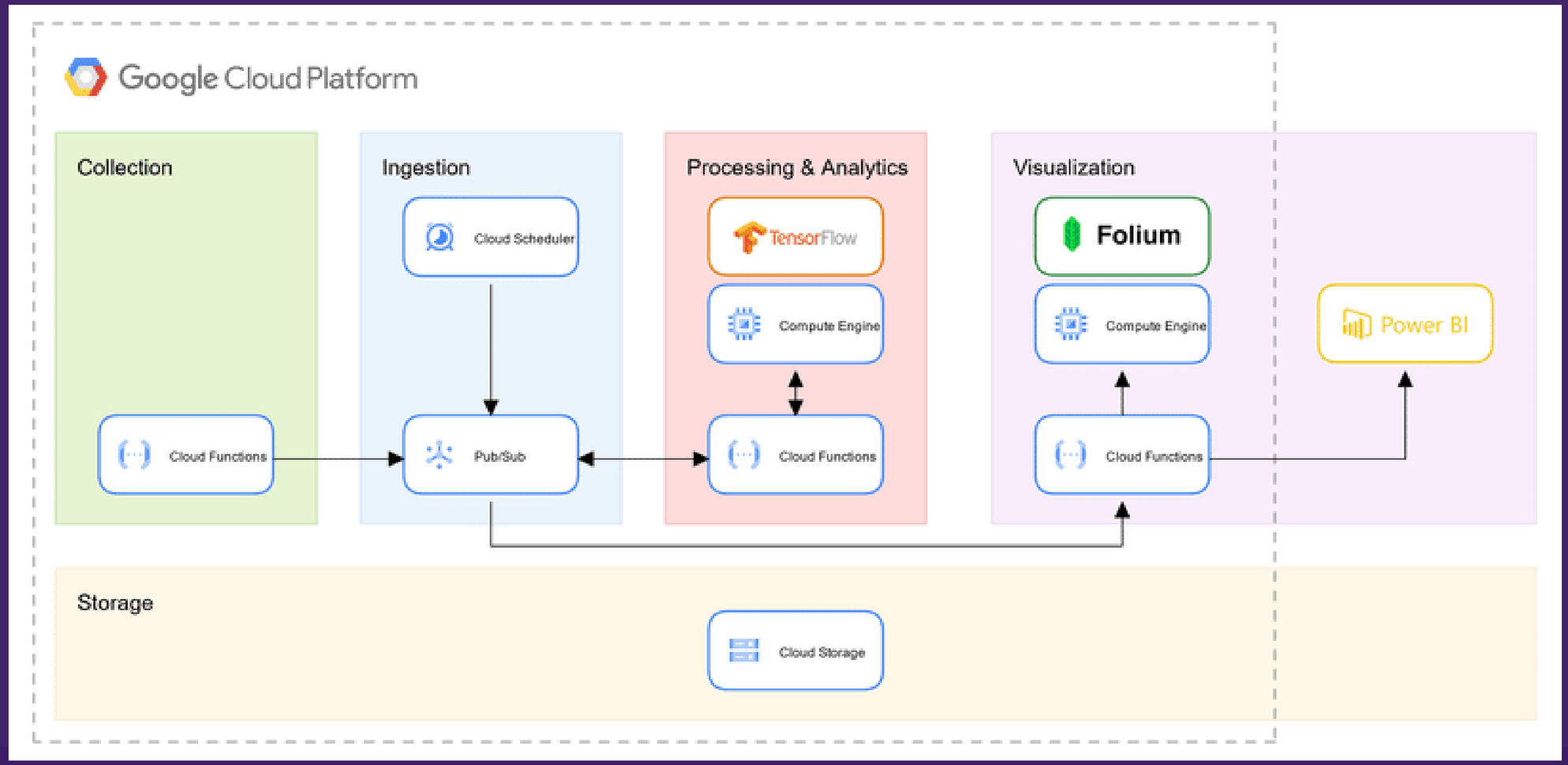
# Module 2
# Data Engineering



กะหล่ำปลีน้ำปลา

# Data Pipeline Architecture

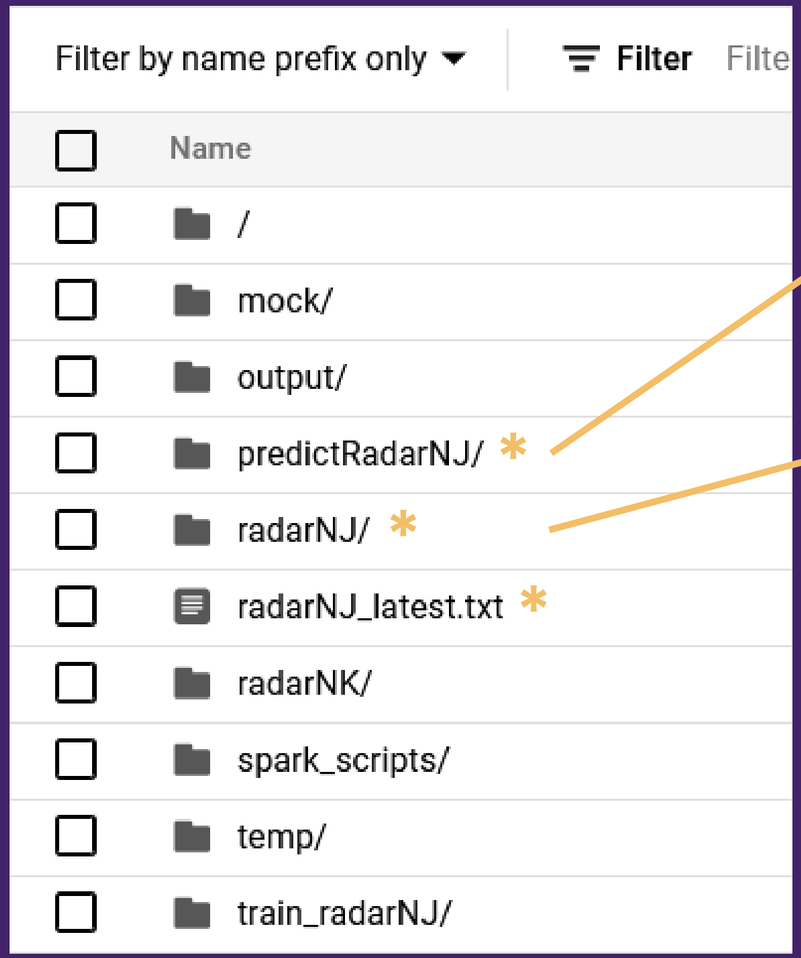
## Cloud Service - GCP

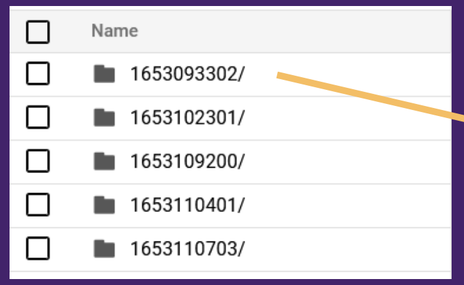


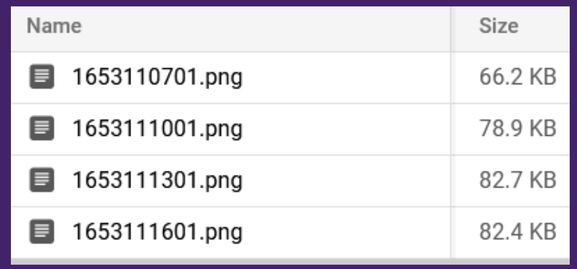
กะหล่ำปลีน้ำปลา

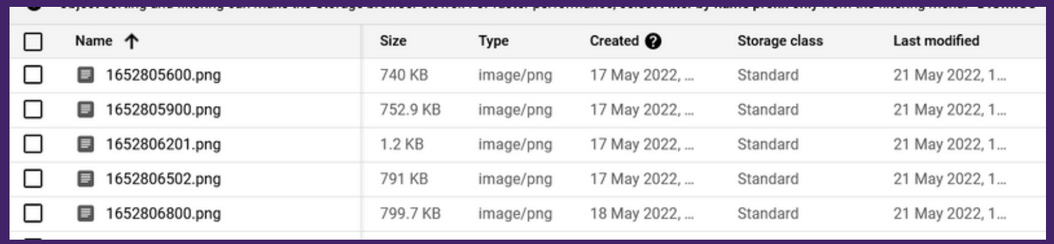
# Data Storage

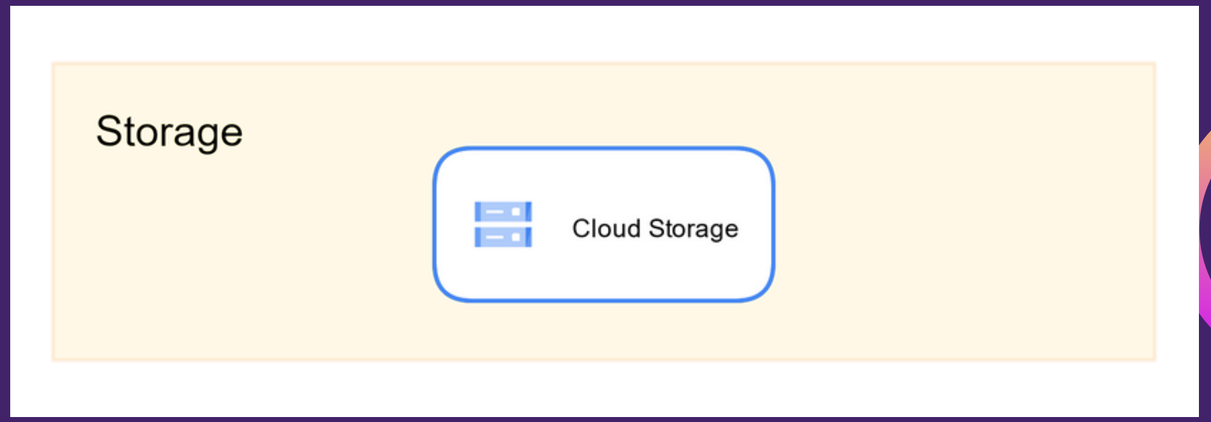
## Google Cloud Storage



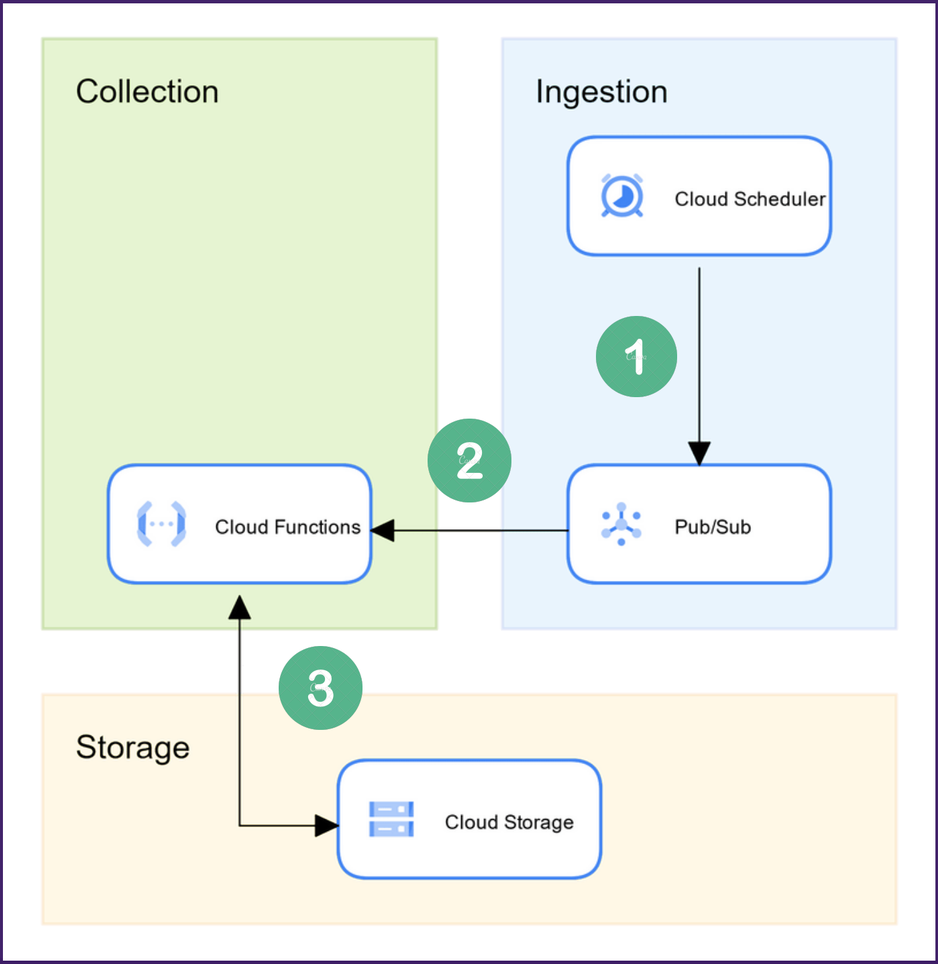



กะหล่ำปลีน้ำปลา

# Data Collection & Ingestion

## Web Scraping w/ Cloud Functions




1

Frequency *
*/5 * * * *

Select a Cloud Pub/Sub topic *
projects/datasci-kalampree/topics/scrape-radar-img-trigger

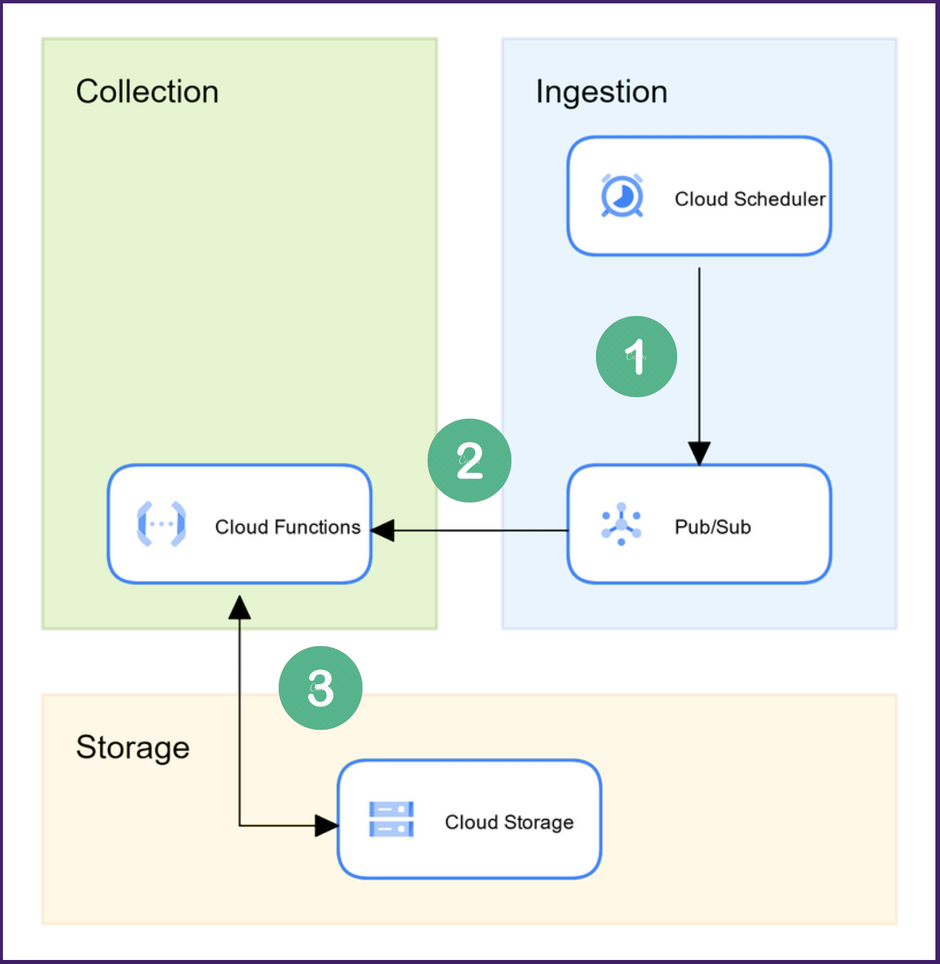
2

| Name | Region | Trigger |
|---|---|---|
| scrape-radar-img | asia-southeast1 | Topic: scrape-radar-img-trigger |
| scrape-radar-img-nj | asia-southeast1 | Topic: scrape-radar-img-trigger |

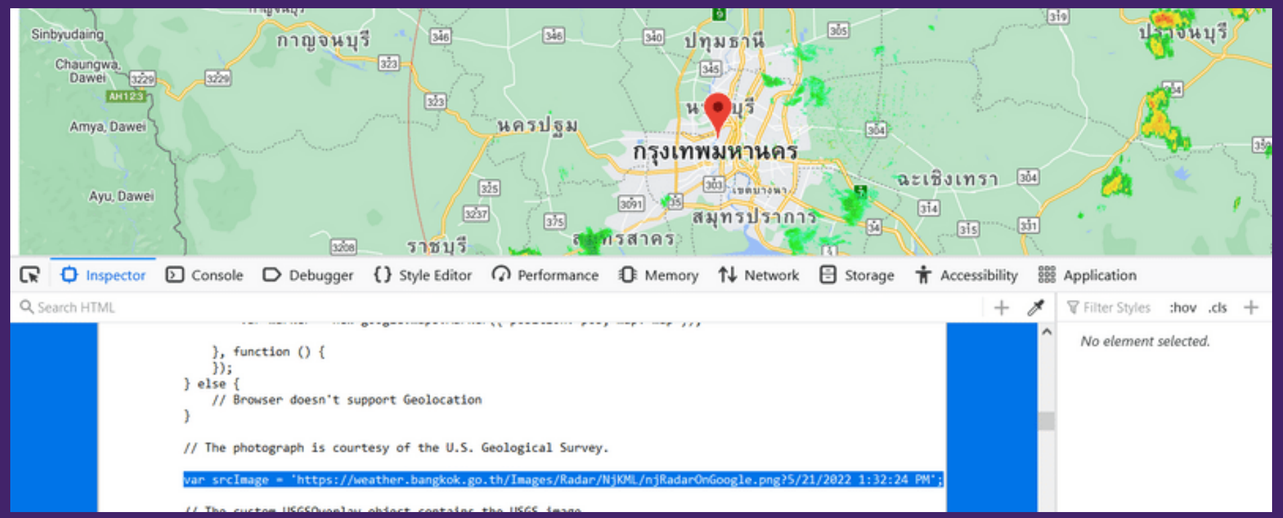
กะหล่ำปลีน้ำปลา

# Data Collection & Ingestion

## Web Scraping w/ Cloud Functions




3



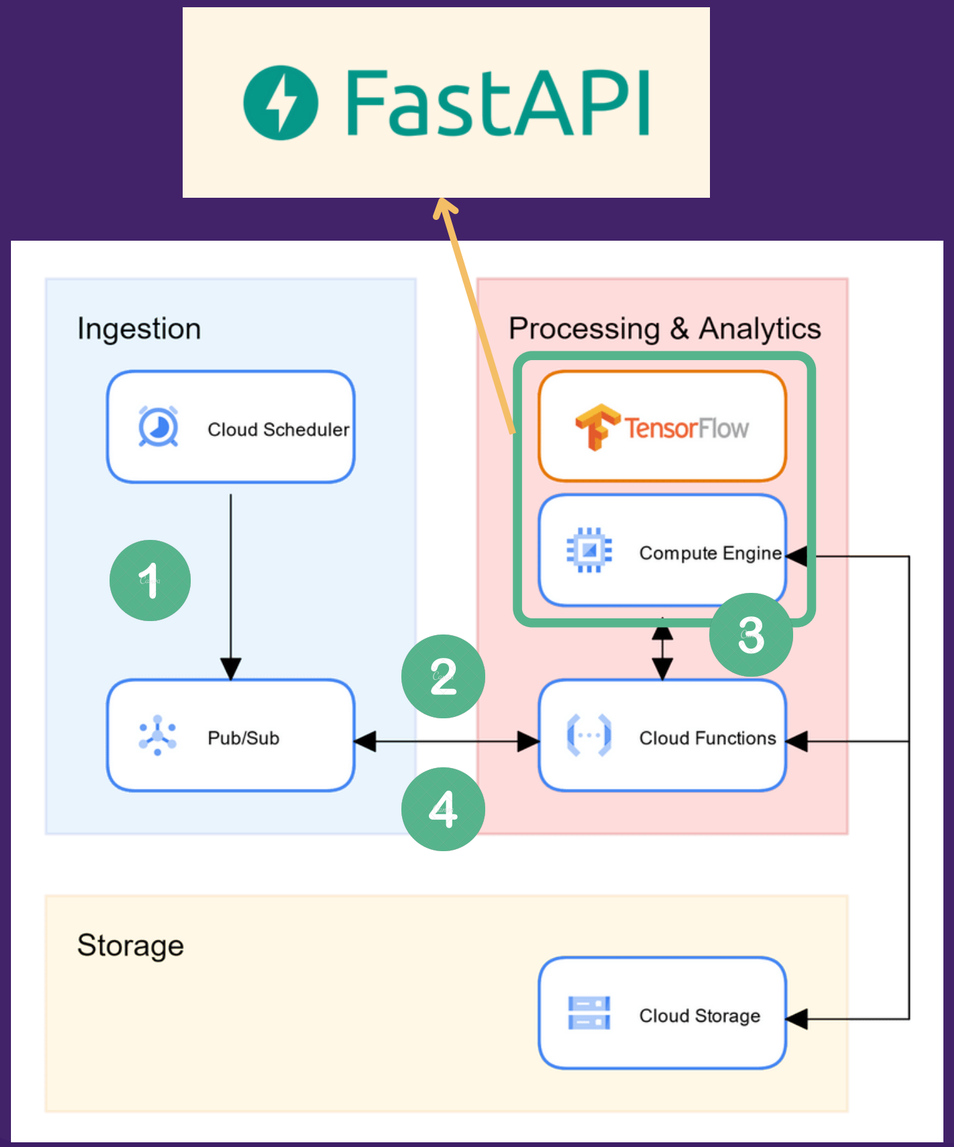

https://weather.bangkok.go.th/radar/GoogleMapNongchok.aspx

```
img_name, img_path = download_image("https://weather.bangkok.go.th/Images/Radar/NjKML/njRadarOnGoogle.png")
upload_path = "radarNJ/" + img_name
upload_blob(BUCKET_NAME, img_path, upload_path)
```

กะหล่ำปลีน้ำปลา

# Data Processing & Analytics

## Model Deployment w/ FastAPI




1 2

**Topic details**

| | |
|---|---|
| Topic name | projects/datasci-kalampree/topics/need-inference-trigger |

3

```
208
209 class RequestDTO(BaseModel):
210     filepaths: List[str]
211
212
213 @app.post("/predict/")
214 async def predict(req: RequestDTO):
215     img_list = append_picture(req.filepaths)
216     nowcasts = prediction_n_time_frame(model, i
217     imgs_filenames = []
218     for i in range(1, len(nowcasts)+1):
219         img = create_alpha_image(nowcasts[i-1])
220         img = Image.fromarray(img)
221         file_local_path = f'./prediction/{str(i
```

```
{
    "filepaths": [
        "radarNJ/1653091202.png",
        "radarNJ/1653091502.png",
        "radarNJ/1653091801.png",
        "radarNJ/1653093302.png"
    ]
}
```

4

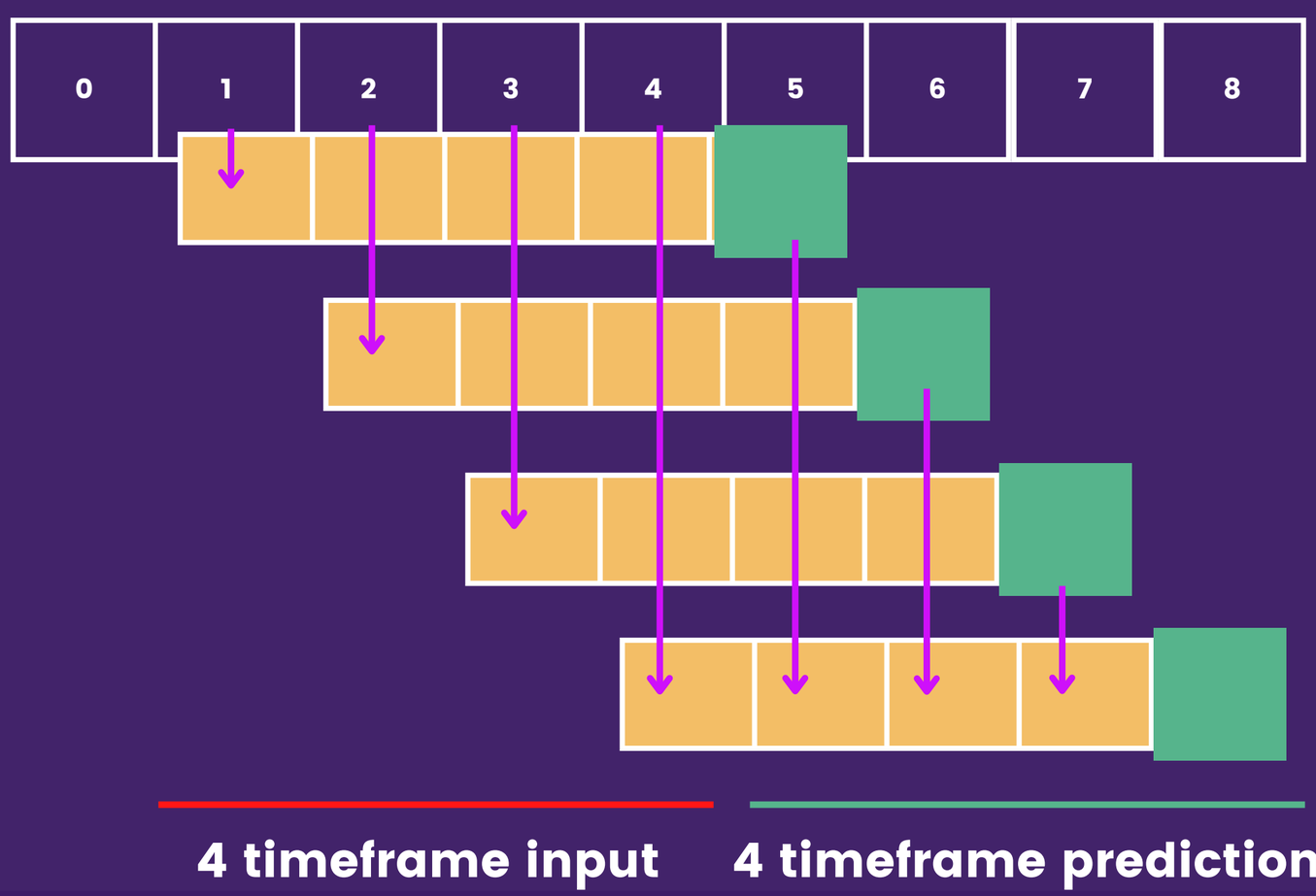
```
pred_filepaths = model_predict(filepaths)
payload = ",".join(pred_filepaths)
pub_future = publisher.publish(topic_path, payload.encode("utf-8"))
```

| | |
|---|---|
| Topic name | projects/datasci-kalampree/topics/finished-inference-trigger |

กะหล่ำปลีน้ำปลา

# Data Processing & Analytics

## Data Preparation for RainNet - Tensorflow

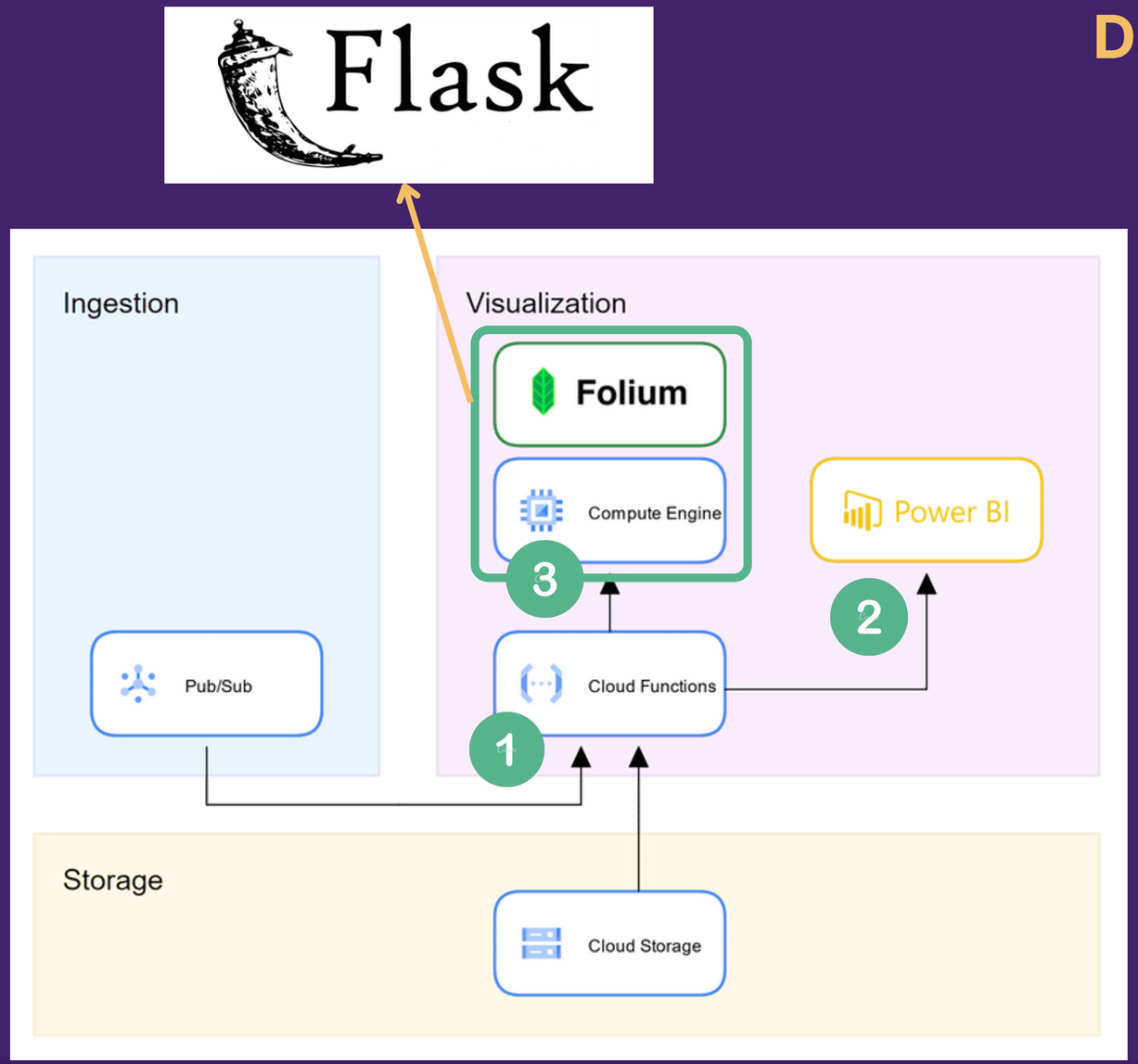



Pre-Processing

กะหล่ำปลีน้ำปลา

# Data Presentation

## Data Postprocessing for Visualization




Topic name projects/datasci-kalampree/topics/finished-inference-trigger

**Message: "filename1.png,filename2.png,filename3.png,filename4.png"**

1

```
for prov in loc_latlong:
    value = getPixelValueFromProvince(prov, img)
    print(value)
    lat,long = loc_latlong[prov]
    temp = name.split("/")[-1]
    temp = temp.replace(".png","")
    timestamp=epoch_to_datetime(int(temp))
    data2.append([timestamp, prov, lat, long, value])
```

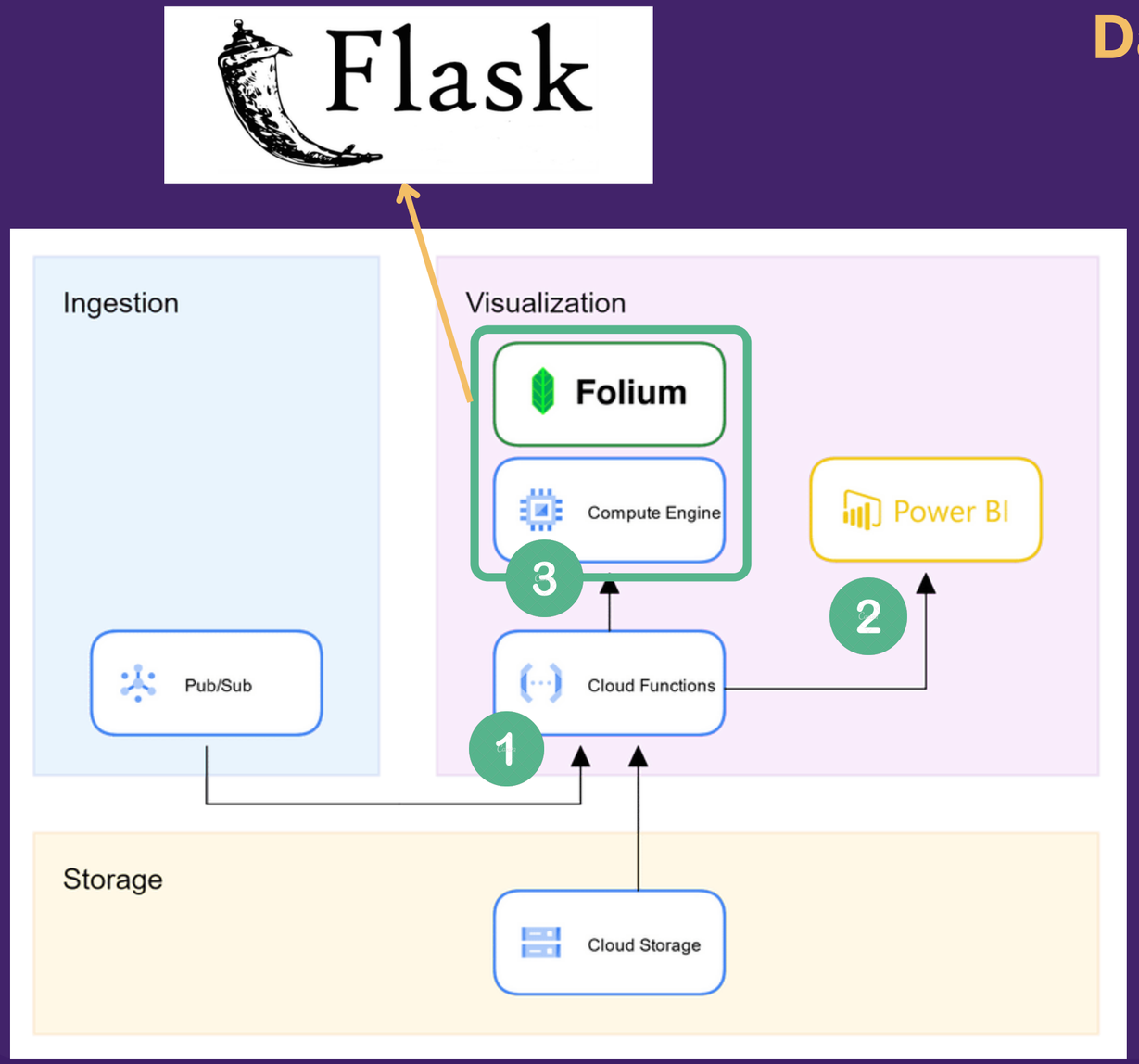
2

```
for d in data2:
    tmp = [{
        "Date" : d[0],
        "Time" : d[0],
        "Lat" : d[2],
        "Long" : d[3],
        "Rain" : d[4],
        "Name" : d[1],
        "ID" : id
        }
    ]
    headers = {
        "Content-Type": "application/json"
    }
    response = requests.request(
        method="POST",
        url=url,
        headers=headers,
        data=json.dumps(tmp)
    )
    print(response)
```

กะหล่ำปลีน้ำปลา

# Data Presentation

## Data Postprocessing for Visualization




3

```
data_dict={"image_path":img_name,"data":data2}
send_to_server(data_dict)
```

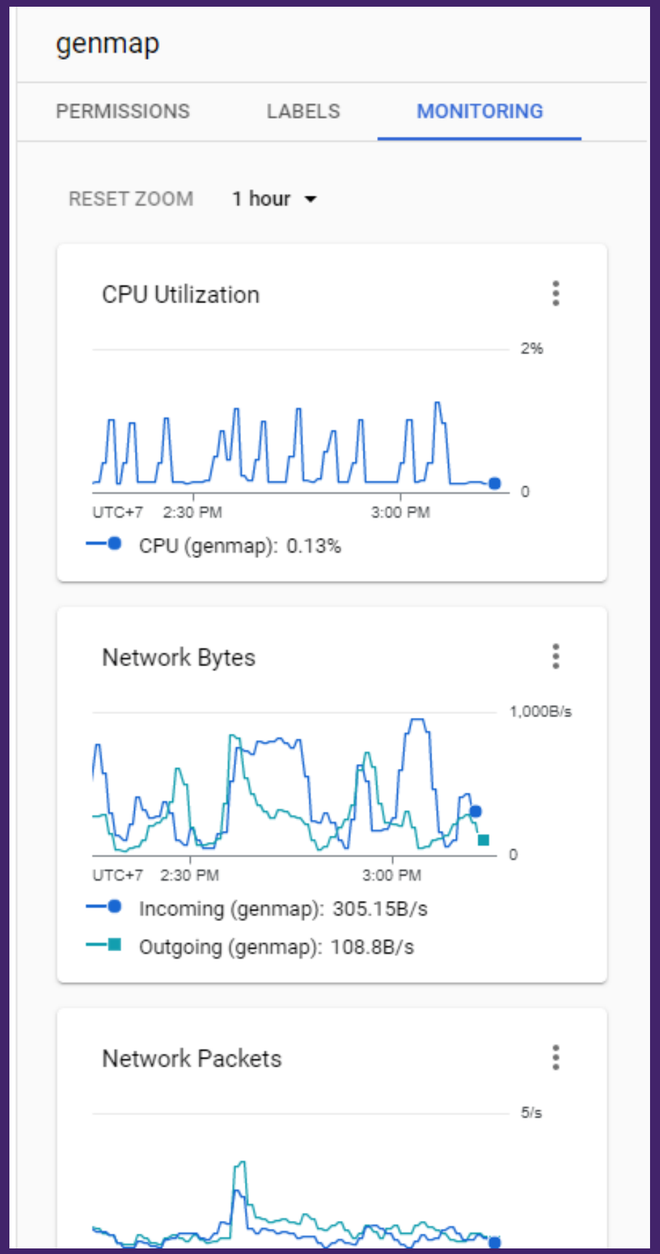
Continue in Module 3...

กะหล่ำปลีน้ำปลา

# Module 3
# Data Visualization

# Visual Map - Timeslider with Folium

## Data Postprocessing for Visualization





| dBZ | คำอธิบาย |
|---|---|
| 10 | หมอกบางมาก |
| 20 | ฝนกำลังอ่อนที่สุด |
| 30 | ฝนกำลังอ่อน |
| 40 | ฝนกำลังปานกลาง |
| 50 | ฝนกำลังแรง |
| 55 | ฝนกำลังแรงมาก |
| >55 | ลูกเห็บหรือน้ำแข็ง |
| 75 | ลูกเห็บหนักมาก |

```python
@app.route('/post', methods=['POST'])
def post_img_dir():
    if request.method == 'POST':
        json_data = request.json
        print(json_data)
        print(type(json_data))
        get_img(json_data)
        gen_map(json_data)
        return "success"
```

```python
@app.route('/map', methods=['GET'])
def render_map():
    if request.method == 'GET':
        return render_template('rain_rate_map.html')
```

**-get_img** ดึงภาพ radar prediction จาก google cloud storage

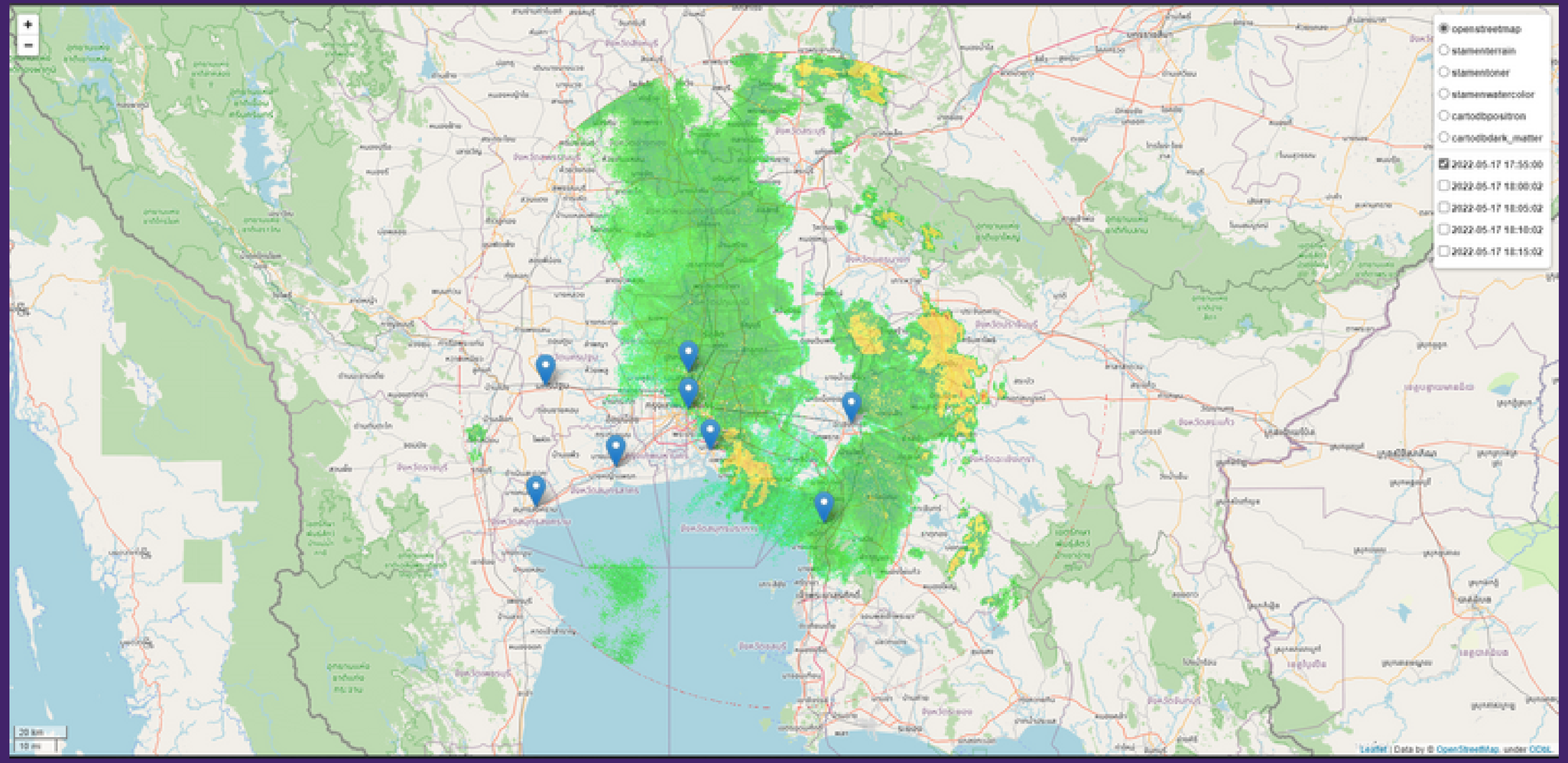
**-gen_map** ทำการสร้าง visual map ที่ประกอบด้วย weather prediction และ rain rate บน html

**-render_map** เมื่อมีการส่ง get request มายังพอร์ต จะทำการคืน html ที่เป็น visual map กลับไป

กะหล่ำปลีน้ำปลา

# Module 3: Visualization

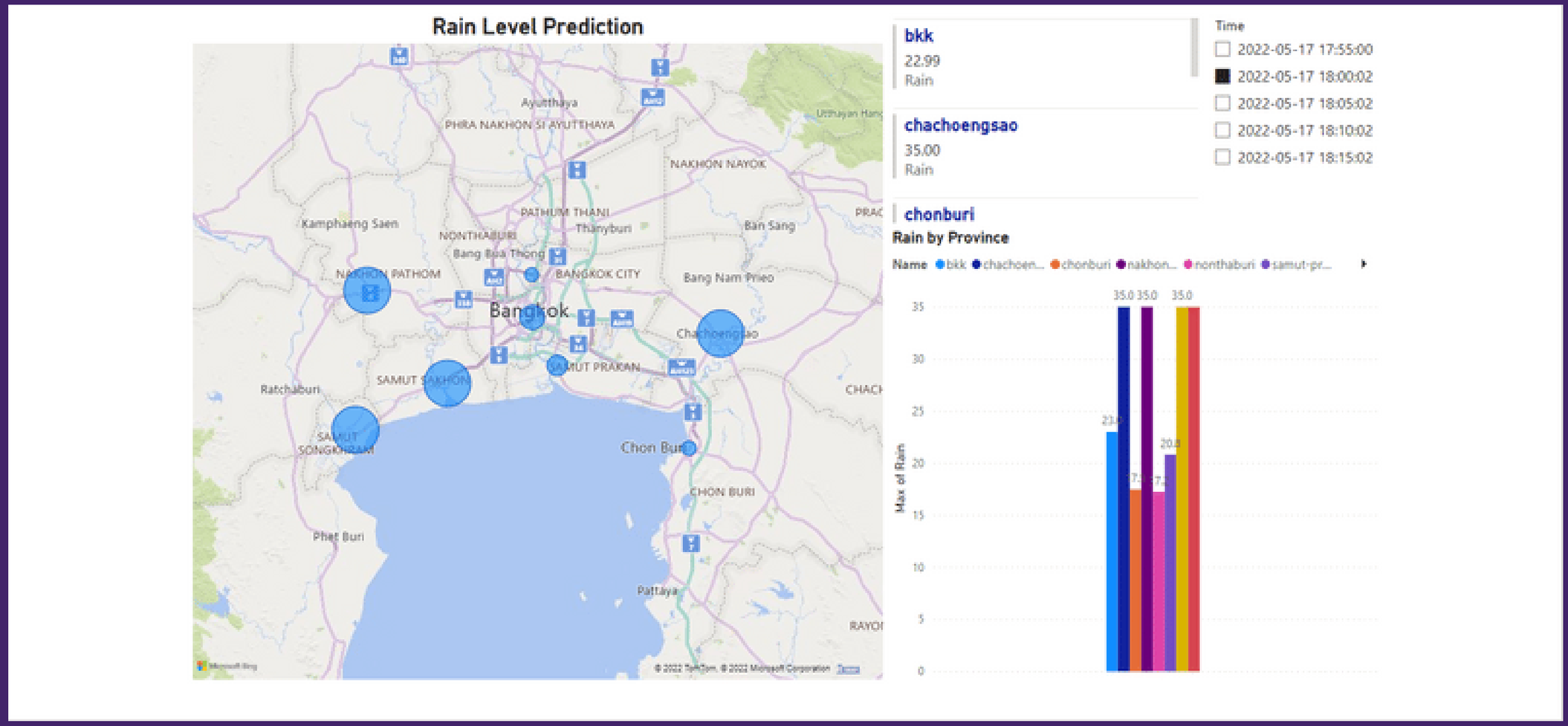
## Visual Map - Timeslider with Folium




http://34.142.140.29:8000/map

กะหล่ำปลีน้ำปลา

# Module 3: Visualization
# Power BI - Realtime Dashboard




https://app.powerbi.com/links/bwxNT5sMBw?ctid=271d5e7b-1350-4b96-ab84-52dbda4cf40c&pbi_source=linkShare

# Demo

# THANK YOU