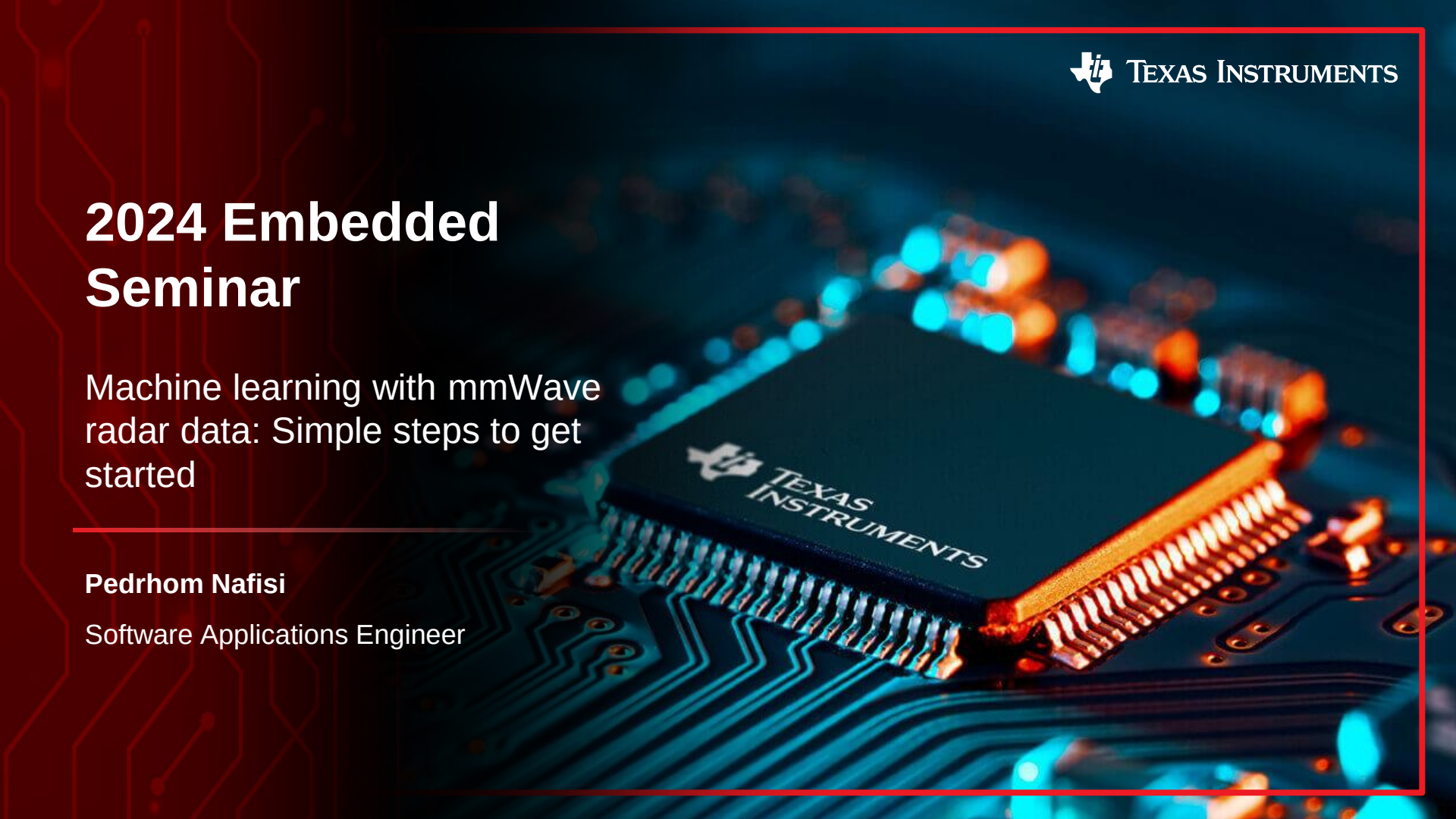
2024 Embedded Seminar

Machine learning with mmWave radar data: Simple steps to get started

Pedrhom Nafisi

Software Applications Engineer



Motivation: ML for mmWave sensors

- TI mmWave Radar Sensors can be used for applications such as gesture, human/non-human classification, and surface classification with home-grown or 3P ML models

- Machine learning (ML) done on-chip can provide a **more reliable and accurate classification** for these applications, moving its capability for identification of objects closer to that of more expensive vision systems.

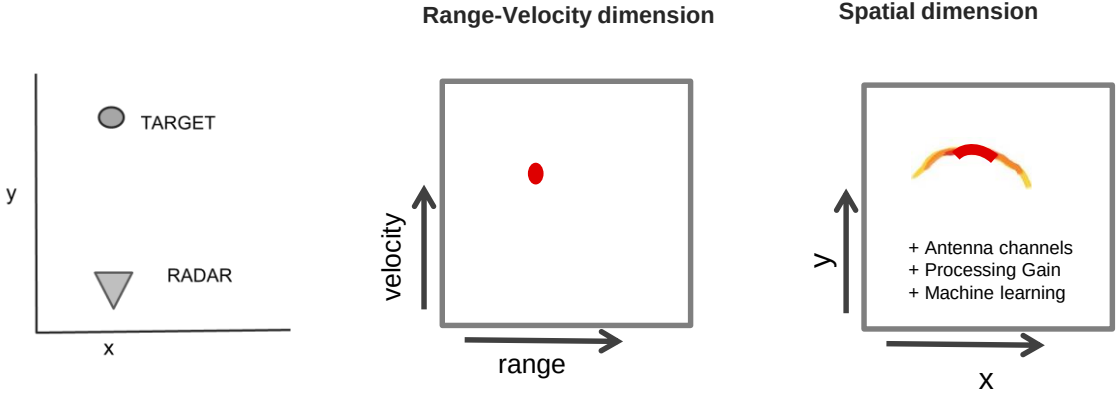
- There is a need for a **documented ML flow** that users can re-produce, understand, and modify/scale for their own purpose, into their own TI-based radar products via open-source or TI licensed tools, libraries, and frameworks.

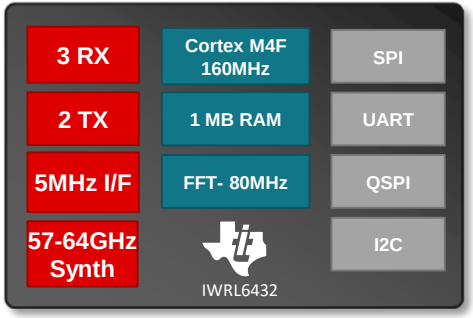
mmWave Data: Range, velocity, and angle

- Radar can sense object range, velocity, and angle.
  - Has very good range and velocity resolution
  - Angular resolution (or ‘spatial resolution’) is limited

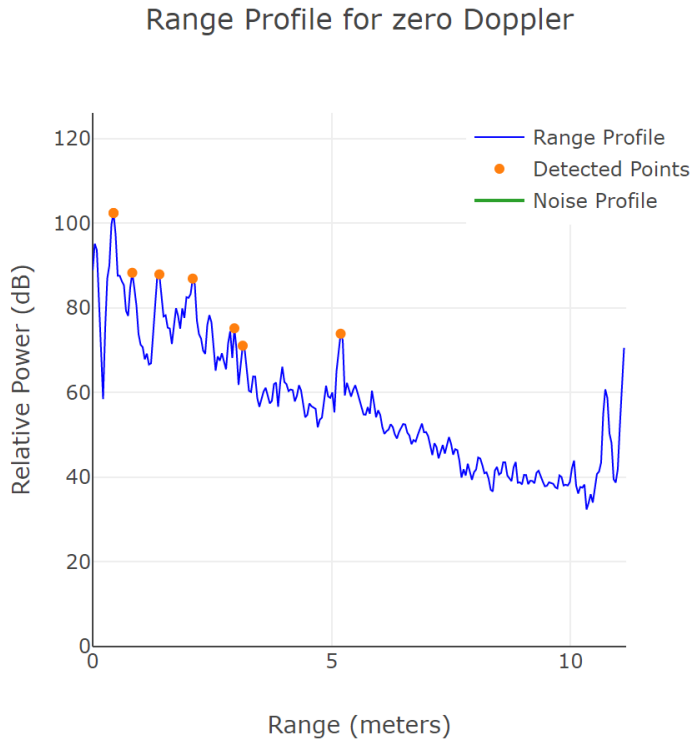
Comparing radar and vision strengths	Radar	Vision
Range resolution		
Velocity resolution		
Spatial resolution		
All weather		
Privacy		
Dust, light, smoke		

Radar images are very sharp in the range-velocity domain but less precise in the spatial domain





mmWave Data: Range profile and configuring

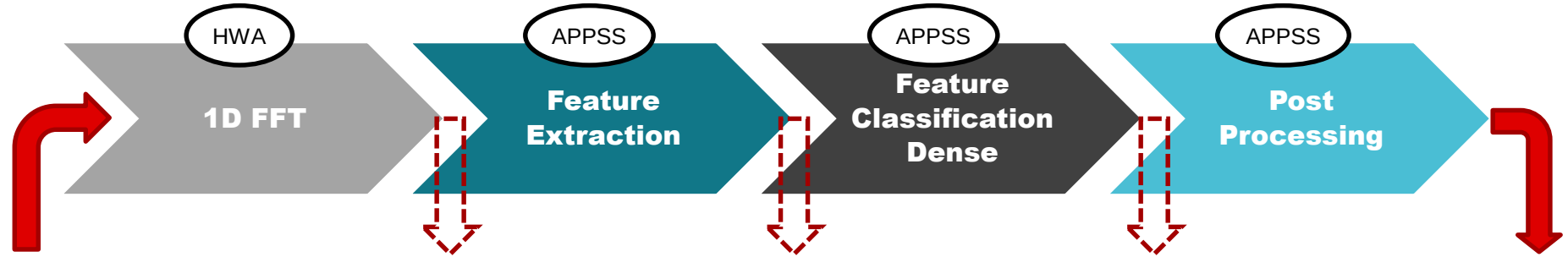


- **Range profile (SDK)**
  - Range FFT output and zero doppler bin (detection matrix)
  - Accumulated / normalised antenna inputs
  - Linear Amplitude
- **Radar configuration (scene)**
  - Radar front-end (chirping) needs to be programmed according to scene

Application: Indoor Robotics

- Cleaning robots (vacuum)
  - Surface classification (cleaning)
  - Low-level obstacle detection (avoidance)
  - Home security (dog or thief)
  - Speed measurement (traction)
- Logistic robots
  - Boundary detection
  - Object detection and collision avoidance
  - Higher speeds
- Surface finishing
  - Surface grading / grinding / polishing

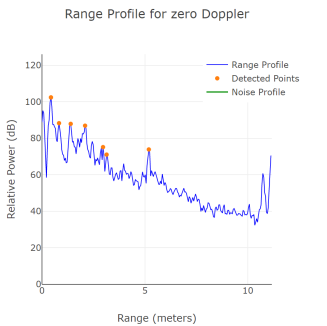
Surface classification: xWRL6432 processing



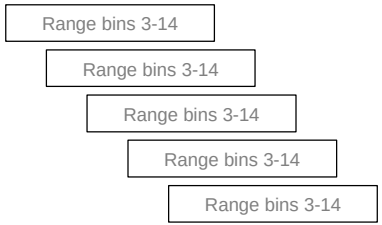
Input:
ADC samples

```
95ff 80ff e3ff e0ff 7bff 60ff f1ff f4f0
75ff 79ff e2ff fdff 72ff 6eff 1b00 3d00
48ff 19ff 3d00 4300 e2fe cafe 0900 f9ff
b1fe 90fe d7ff a8ff 9ffe bcfe 77ff 4bff
b7fe d2fe 38ff 1fff e3fe 11ff 1eff 2cff
47ff 34ff 41ff 5cff 4cff 54ff 7cff 6aff
44ff 2fff 6bff 5cff 0cff 29ff 20ff 20ff
2aff 3aff fdfe effe 68ff 74ff 11ff 26ff
87ff 73ff 45ff 35ff 6fff 78ff 3aff 4fff
4bff 47ff 29ff 3dff 64ff 5eff 45ff 48ff
4aff 4cff 4dff 3aff 4dff 47ff 37ff 37ff
38ff 47ff 28ff 38ff 58ff 8cff 2eff 43ff
acff 97ff 80ff 9bff 8cff 88ff 9aff b9ff
78ff 5eff b0ff 99ff 52ff 40ff 98ff 8dff
30ff 23ff 85ff 64ff 33ff 40ff 3eff 32ff
5cff 77ff 5cff 74ff 57ff 64ff 78ff 89ff
62ff 6aff 9fff 83ff 50ff 6aff 68ff 62ff
76ff 66ff 75ff 8dff 77ff 52ff 99ff 8eff
55ff 43ff 7bff 6cff 37ff 5bff 3fff 3cff
7fff adff 48ff 71ff cdff d2ff 9bff c3ff
```

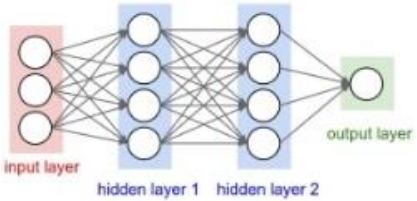
Intermediate O/P:
Range Profile



Intermediate O/P:
5 rolling frame feature vector



Intermediate O/P:
Classified Surface



Output:
0 or 1

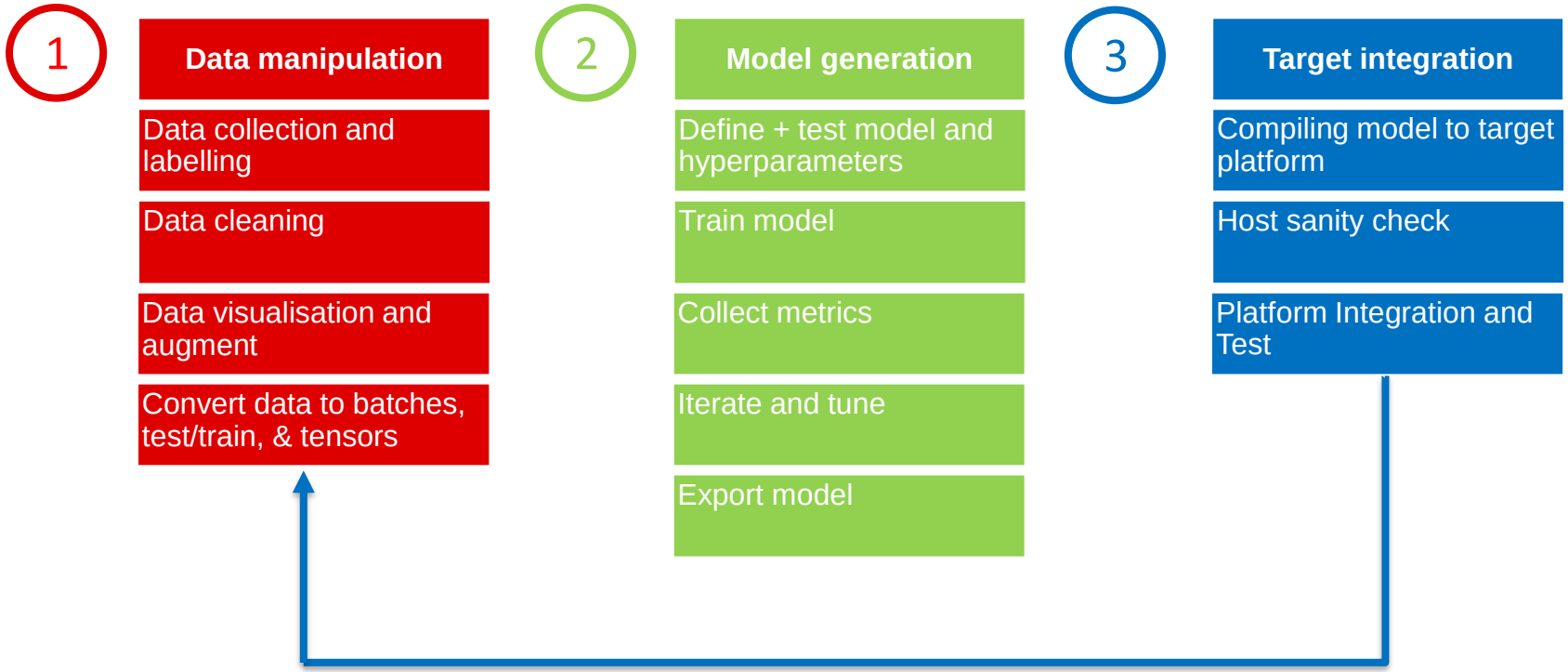
Dry
Wet

Machine learning flow and results

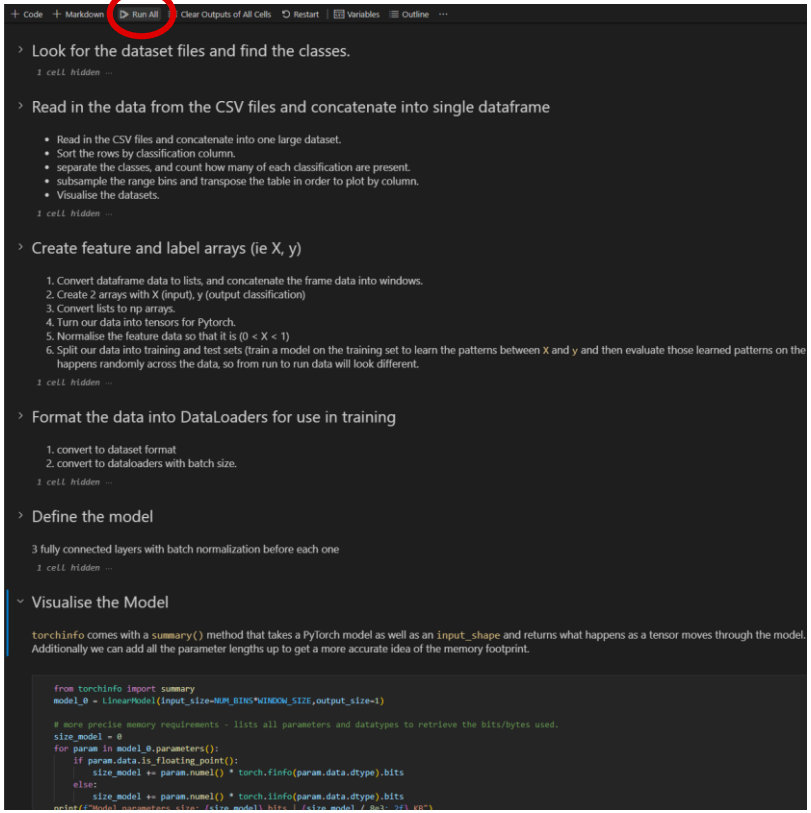
ML Flow: Environment and setup

- **Model**
  - **Language/ML library**: Python 3.11+ / Pytorch 2.x /… + helper modules.
  - **ML Development IDE**: Jupyter notebook / Visual Studio Code
  - **ML Compiler**: Apache TVM 0.16 (Tensor Virtual Machine) - TI version (TI NNC)
- **Target**
  - **IDE**: CCS12.x
  - **Compilers**: TI CLANG (Cortex M4/M0)

ML Flow: Major steps

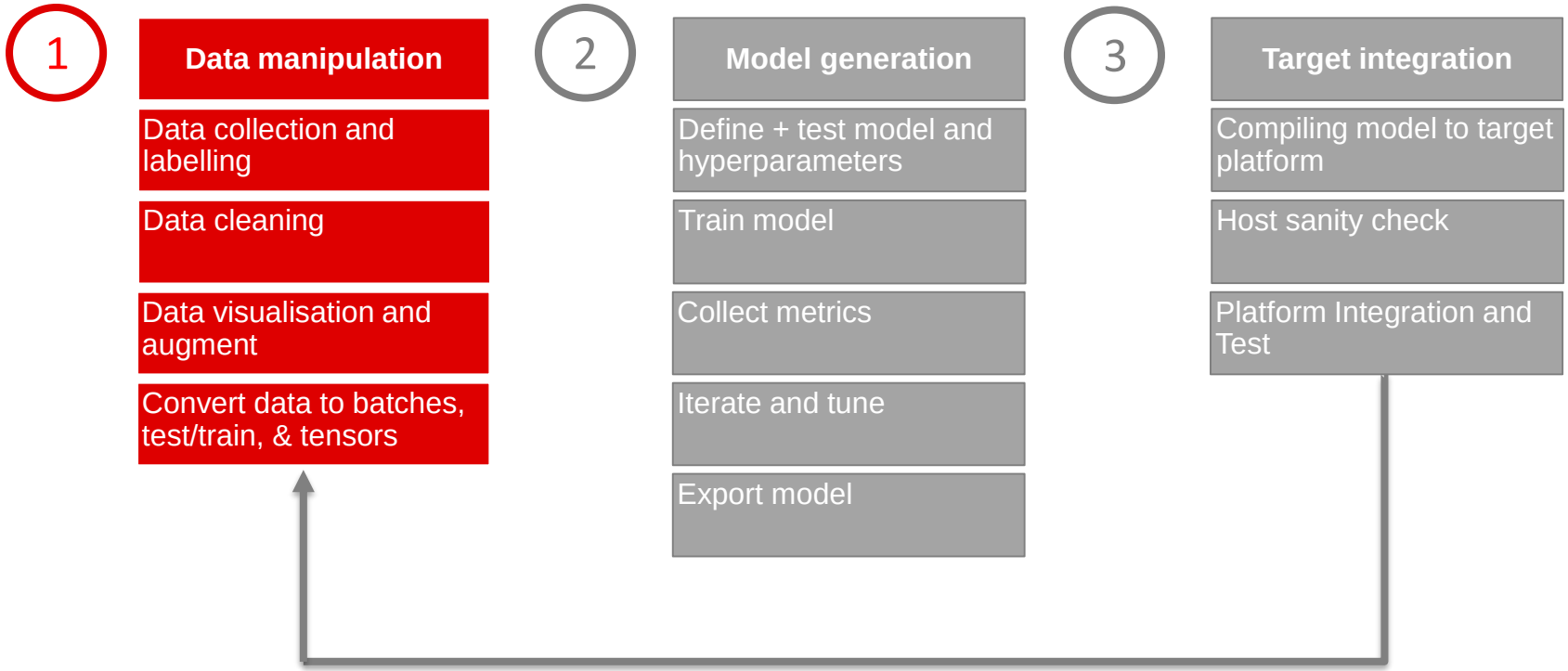


ML Flow | Juypter notebook

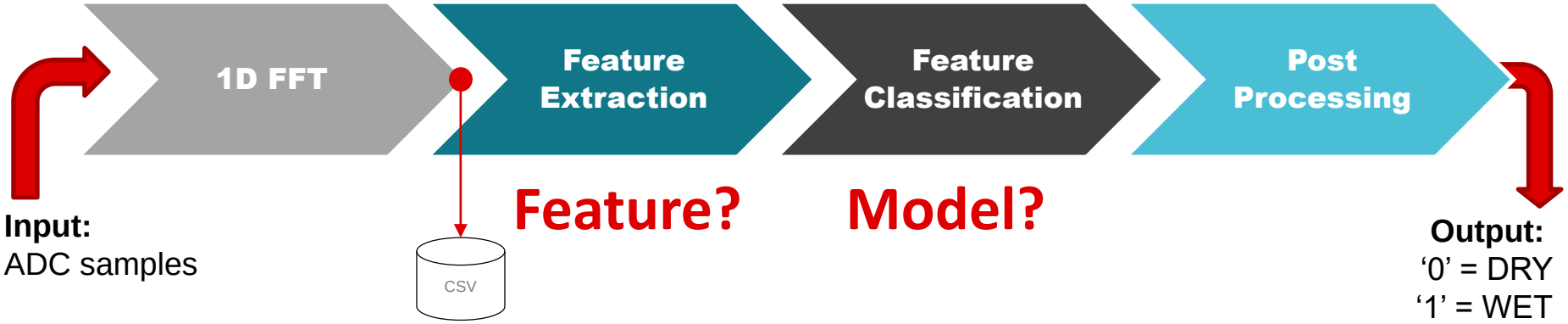


- Python based computing platform that provides interactive documentation and source code combined
- Formatted into blocks of code that can be ran one step at a time with each block's output displayed, or all at once.
- Implements every step needed to do for data manipulation (step 1) and model generation (step 2)
- Data files can be changed and notebook code blocks tweaked to create a model for any radar application

ML Flow | Major steps



Data | Decisions, decisions …



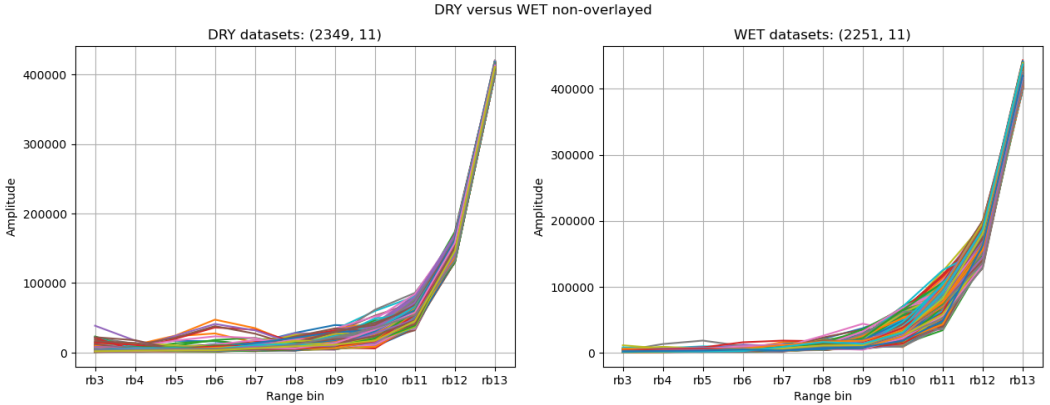
What I know

- Memory usage and processor load
- Input data types available from radar, and output classification needed.

What I don't know

- What is the best format of my data to extract helpful features ?
- Which model will be appropriate ?

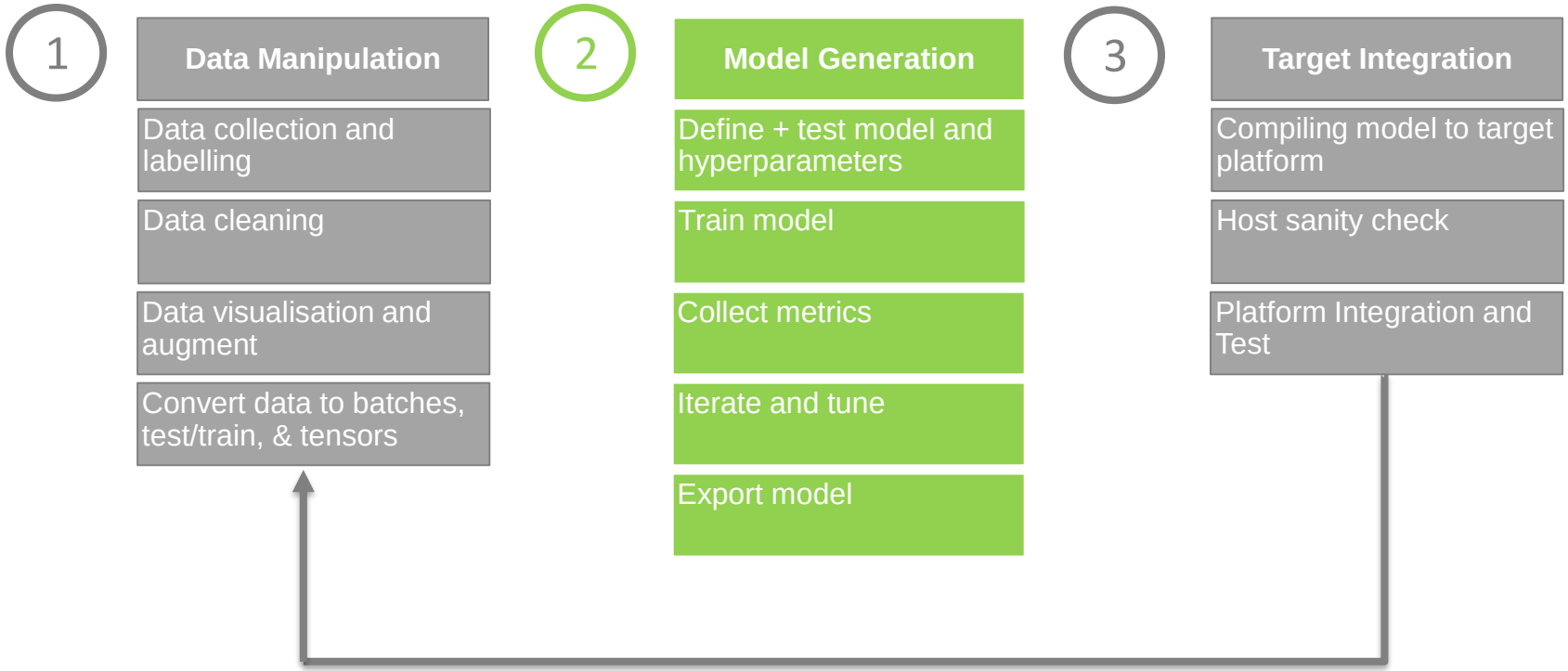
Data | Data manipulation



- Label saved data using provided separate python script
- Visualizes data to ensure no anomalies
- Converts data to Tensor format to prepare for model generation

```
Values for first sample of X:
 tensor([0.0015, 0.0019, 0.0016, 0.0040, 0.0019, 0.0018, 0.0032, 0.0026, 0.0038,
        0.0040, 0.0026, 0.0030, 0.0030, 0.0036, 0.0050, 0.0029, 0.0027, 0.0062,
        0.0072, 0.0065, 0.0039, 0.0042, 0.0074, 0.0092, 0.0051, 0.0041, 0.0056,
        0.0078, 0.0090, 0.0142, 0.0083, 0.0107, 0.0097, 0.0144, 0.0201, 0.0240,
        0.0228, 0.0161, 0.0269, 0.0197, 0.0636, 0.0458, 0.0504, 0.0557, 0.0487,
        0.1607, 0.1416, 0.1399, 0.1411, 0.1412, 0.4294, 0.4146, 0.4164, 0.4154,
        0.4173])
and corresponding y: tensor([0.])
```

ML Flow | Major steps



Model | Start simple

Torch summary output for 3 layer linear model

```
Model parameters size: 191968 bits | 24.00 KB

==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
LinearModel                              [1, 1]                    --
├─BatchNorm1d: 1-1                       [1, 55]                   110
├─Linear: 1-2                            [1, 64]                   3,584
├─ReLU: 1-3                              [1, 64]                   --
├─BatchNorm1d: 1-4                       [1, 64]                   128
├─Linear: 1-5                            [1, 32]                   2,080
├─ReLU: 1-6                              [1, 32]                   --
├─BatchNorm1d: 1-7                       [1, 32]                   64
├─Linear: 1-8                            [1, 1]                    33
==========================================================================================
Total params: 5,999
Trainable params: 5,999
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.01
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.02
Estimated Total Size (MB): 0.03
==========================================================================================
```

- Provided notebook has an implementation of a simple linear model to start.
- Memory required and relative loading of the model should be known before training starts.
- Training and testing loops provided

Model | Training loop - hyperparameters

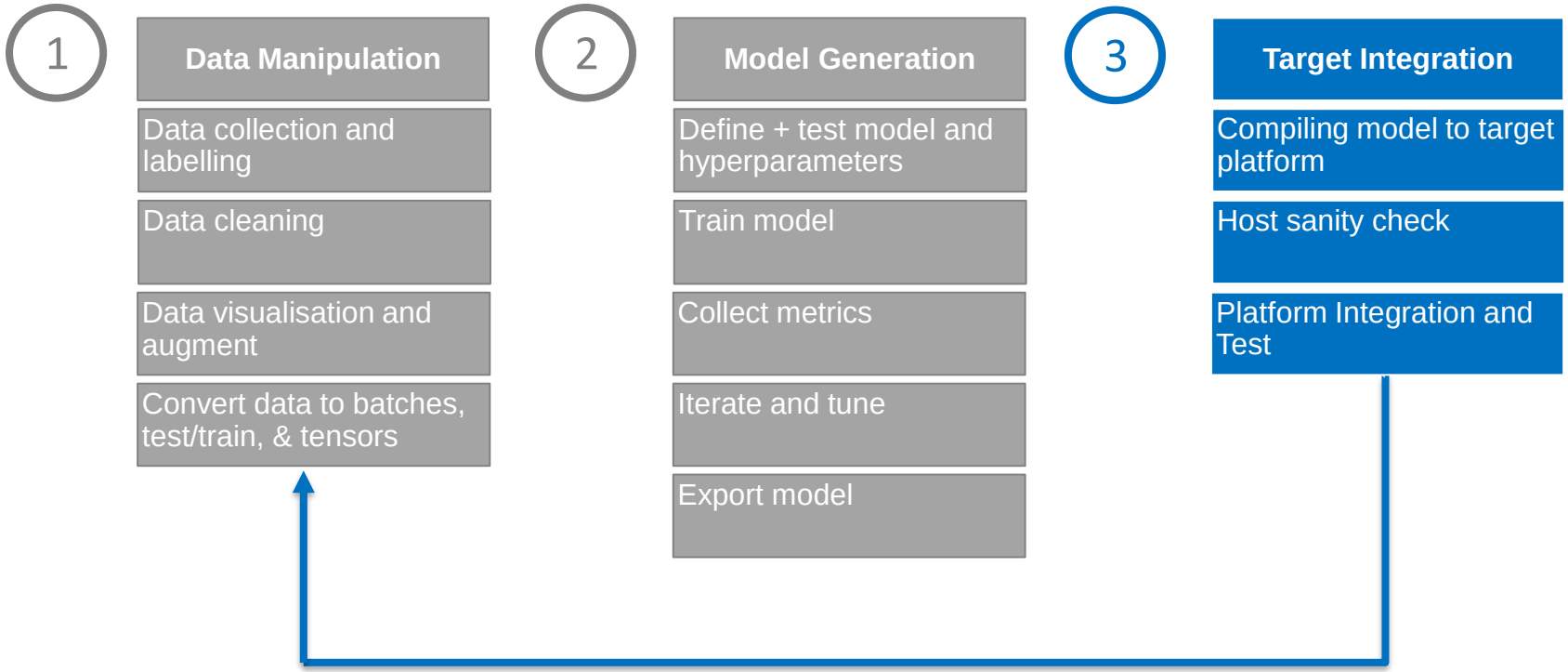
```
## All Hyper parameters

VISUALISE = False           # plot accumulated datasets or not ? (True/False)

# Dataset sample extraction
START_BIN = 3               # first range bin to consider.
END_BIN = 14                # last range bin to consider.
NUM_BINS = END_BIN-START_BIN    # number of bins in total

# Dataset feature formatting
WINDOW_SIZE = 5             # window size of frame data: 1,2,3,4,5,6 ... for example.
WINDOW_CONCATENATE = False      # concatenate each frame within the window or not (ie merge)
WINDOW_SORT = False         # Sort the window data or not (True or False)

# Training topics
NUM_EPOCHS = 500            # TRAINING: number of epochs to run
BATCH_SIZE = 64             # TRAINING: batch size of datasets defining the iterative model update, 64 is OK result.
TEST_SIZE_PERCENT = 20      # split dataset % between test(TEST_SIZE_PERCENT) and train(100-TEST_SIZE_PERCENT)
LEARNING_RATE = .001        # Learning rate of the optimizer which determines the step size at each iteration while moving toward a minimum of a loss function
F1_SCORE_THRESH = 0.5       # Threshold for converting input into predicted labels for each sample.
                            # Compute binary f1 score, which is defined as the harmonic mean of precision and recall.
```

Model | Metrics from testing loop

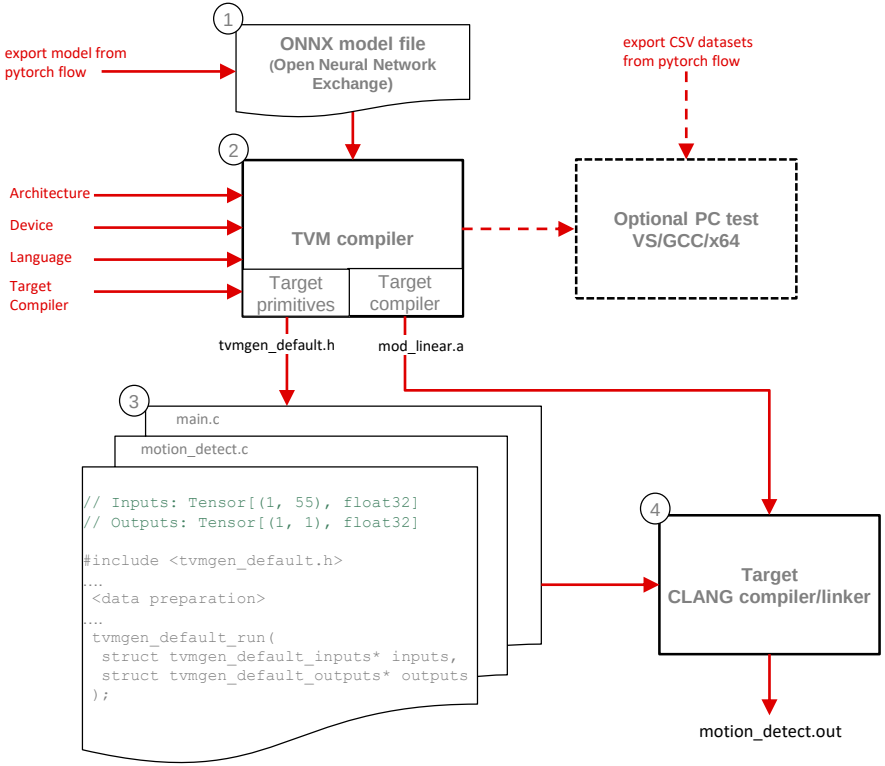
```
EPOCH: 0 | F1: 0.00000
Train loss: 0.28705 | Test loss: 0.28195, Test acc: 43.64%
EPOCH: 50 | F1: 0.93617
Train loss: 0.07864 | Test loss: 0.07505, Test acc: 92.08%
EPOCH: 100 | F1: 0.97872
Train loss: 0.06157 | Test loss: 0.05919, Test acc: 92.63%
EPOCH: 150 | F1: 0.97872
Train loss: 0.05330 | Test loss: 0.05018, Test acc: 94.75%
EPOCH: 200 | F1: 0.97872
Train loss: 0.04944 | Test loss: 0.04616, Test acc: 94.98%
EPOCH: 250 | F1: 0.97872
Train loss: 0.04698 | Test loss: 0.04263, Test acc: 95.31%
EPOCH: 300 | F1: 0.97872
Train loss: 0.04127 | Test loss: 0.04128, Test acc: 95.09%
EPOCH: 350 | F1: 0.97872
Train loss: 0.04023 | Test loss: 0.03998, Test acc: 95.42%
EPOCH: 400 | F1: 0.97872
Train loss: 0.03681 | Test loss: 0.03905, Test acc: 95.54%
EPOCH: 450 | F1: 0.97872
Train loss: 0.03513 | Test loss: 0.03916, Test acc: 95.54%
```

- Key metrics that are monitored during training is Epoch vs Accuracy and F1.
- **Accuracy**
  - Number of correct decisions divided by All decisions
- **F1 score**
  - Evaluation metric specifically for binary classification datasets for measuring performance

ML Flow | Major steps



Target Integration | Target code generation



1. Export model in ONNX format from Juypter notebook.
2. Convert trained ML models into C code using TI TVM and provided instructions
3. Integrate: Implement into target application and add input/output data-handling.
4. Compile and test.

Target Integration | Platform integration testing

```
DRY data
100: 100, 0, Accuracy: 100.00%
200: 200, 0, Accuracy: 100.00%
300: 292, 8, Accuracy: 97.33%
400: 392, 8, Accuracy: 98.00%
<..>
1800: 1753, 47, Accuracy: 97.39%
1900: 1852, 48, Accuracy: 97.47%
2000: 1949, 51, Accuracy: 97.45%
2100: 2049, 51, Accuracy: 97.57%
2200: 2149, 51, Accuracy: 97.68%
2300: 2249, 51, Accuracy: 97.78%
DRY data complete
WET data
2400: 2343, 57, Accuracy: 97.63%
2500: 2435, 65, Accuracy: 97.40%
2600: 2528, 72, Accuracy: 97.23%
2700: 2628, 72, Accuracy: 97.33%
<..>
4100: 3906, 194, Accuracy: 95.27%
4200: 4006, 194, Accuracy: 95.38%
4300: 4106, 194, Accuracy: 95.49%
4400: 4206, 194, Accuracy: 95.59%
4500: 4306, 194, Accuracy: 95.69%
4600: 4406, 194, Accuracy: 95.78%   (pytorch = 94.75%)
WET data complete
```

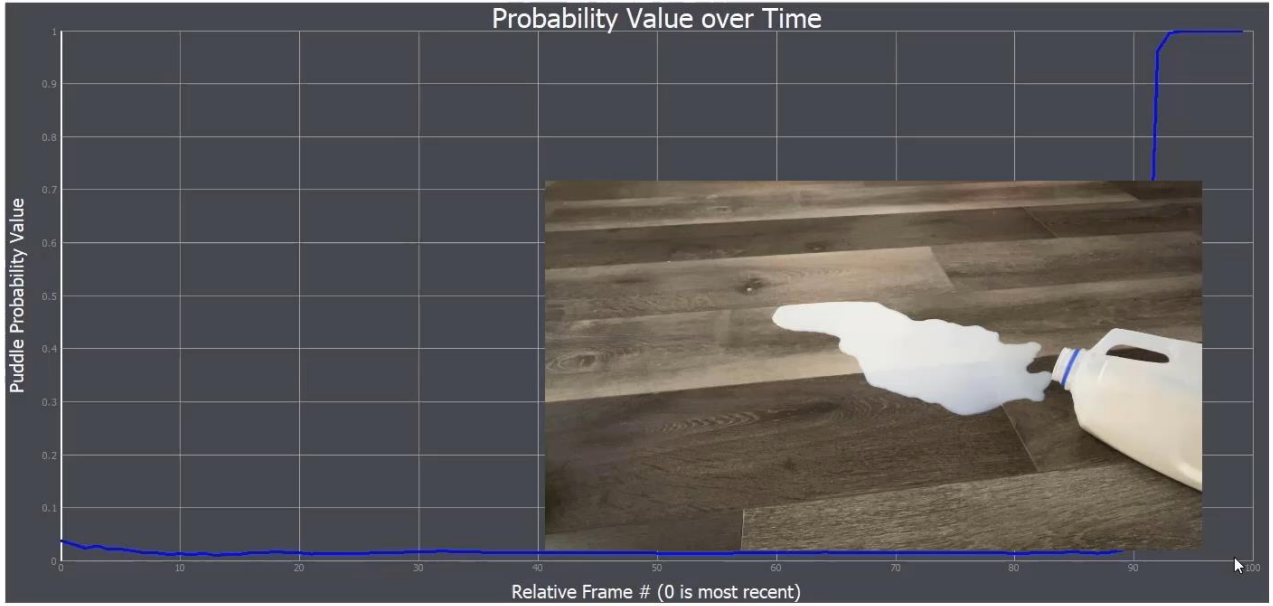
1. Use provided testing radar binary firmware to read training data as input rather than the sensor front end
2. Load datasets used to train the mode onto the device
3. Validate and compare accuracy with the Jupyter notebook model's metrics

Results | Functional demo

Puddle Classification
No Puddle

Puddle Classification Value
3.77266%

Puddle Detection



Thank You ☺

Questions ??

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265