EE P 596: TinyML

Lecture 7: TinyML for Motion Classification

Dept. of Electrical and Computer Engineering

University of Washington

Instructor: Dinuka Sahabandu (sdinuka@uw.edu)

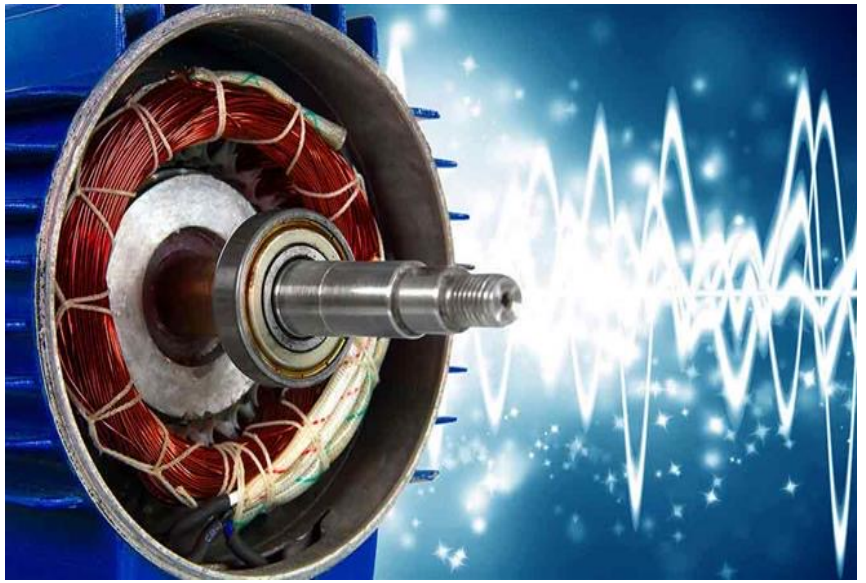
ELECTRICAL & COMPUTER ENGINEERING

UNIVERSITY of WASHINGTON

NSL

Topics Covered

- TinyML Final Projects
- Recent Advancements in ML and Their Relation to TinyML
- Background on Motion Classification and different types
- Supervised learning for motion classification
- Sensor interface for Mechanical Stress in Transport
- Model Training, Evaluation Metrics, and Deployment
- As with the past classes, today we will move away from Tensor flow and use Edge Impulse



❖Relevant reading from the Textbook [Warden and Situnayake; O'REILLY Publisher] is Chapter 17

TinyML Final Projects

1. Elderly Fall Detection Using IMU Sensor
2. SmartGuard: TinyML-Based Fall Detection System for Elderly Care
3. Elderly health alert system
4. TinyML-Based Human Activity Recognition Using Wearable Sensor Data

Four Groups !!!

5. Speech to Text Sensor
6. Self-Checkout with Vegetable Recognition
7. Canine Acoustic Detection with TinyML
8. Safe Driver – Drowsiness Detection
9. Basic Hand Gesture Control
10. Gesture Controlled IoT Device

Two Groups

11. Using Shapley Values for Detecting Adversarial Attacks in TinyML Settings
12. Musical Instrument Classification
13. Recognizing Birds via Their Sounds

TinyML Final Projects

- Project presentations are **on May 29th – from 6pm – 10pm PT**
- What type of hardware to use?

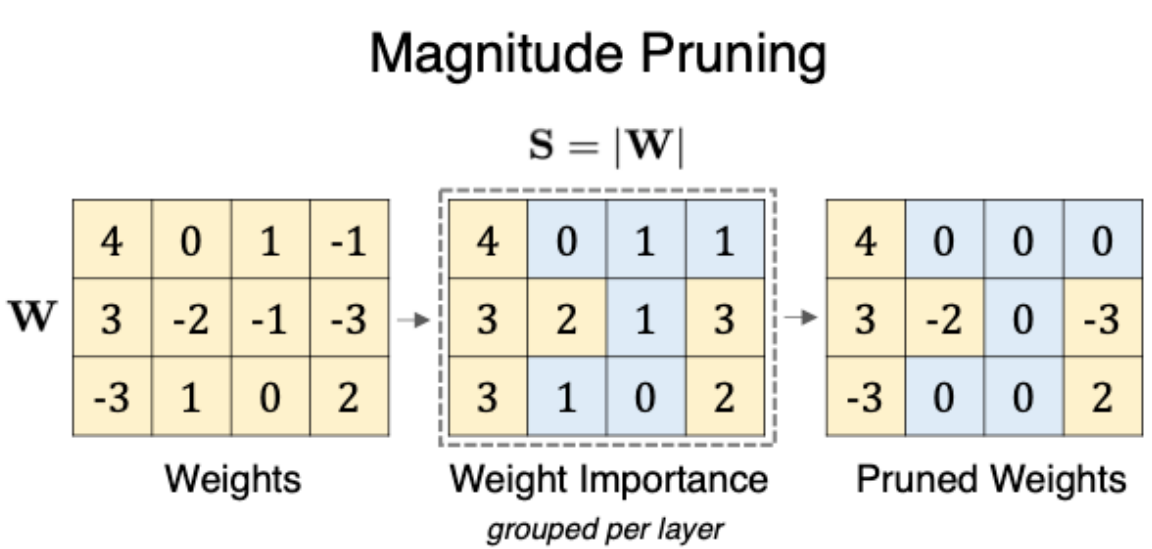
	Embedded Device	Tiny Device
Compute 	1 GHz – 4 GHz	**1 MHz – 400 MHz**
Memory 	512 MB – 64 GB	**2 KB – 512 KB**
Storage 	64 GB – 4 TB	**32 KB – 2 MB**
Power 	30 W – 100 W	**150 µW – 23.5 mW**

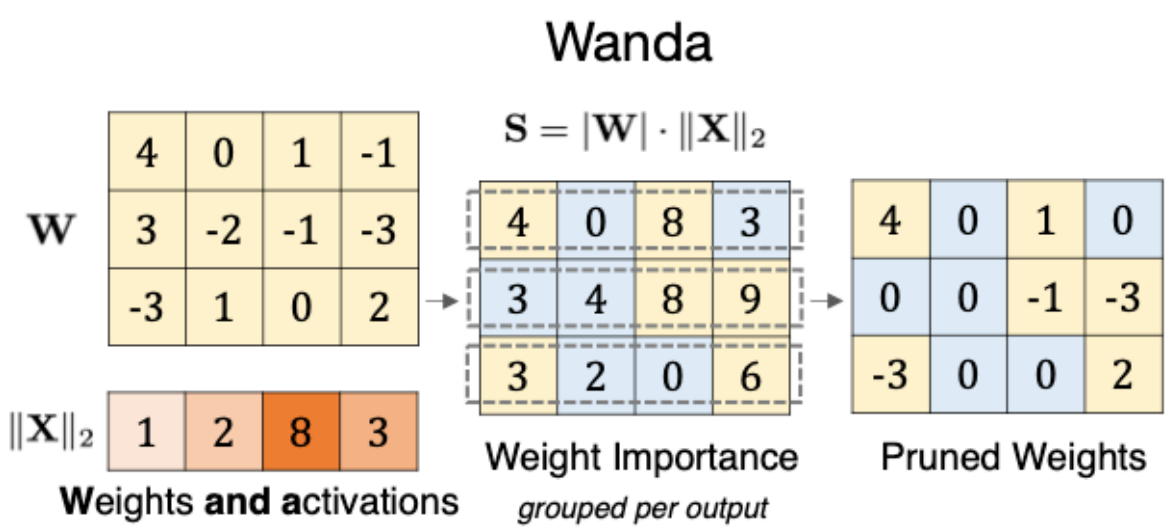
Up to 128 MB Memory is Okay

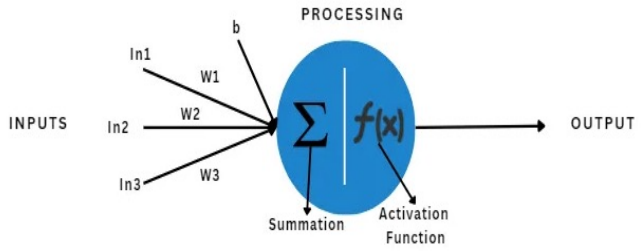
Explore your hardware (if needed): https://www.arduino.cc/en/hardware#nano-family

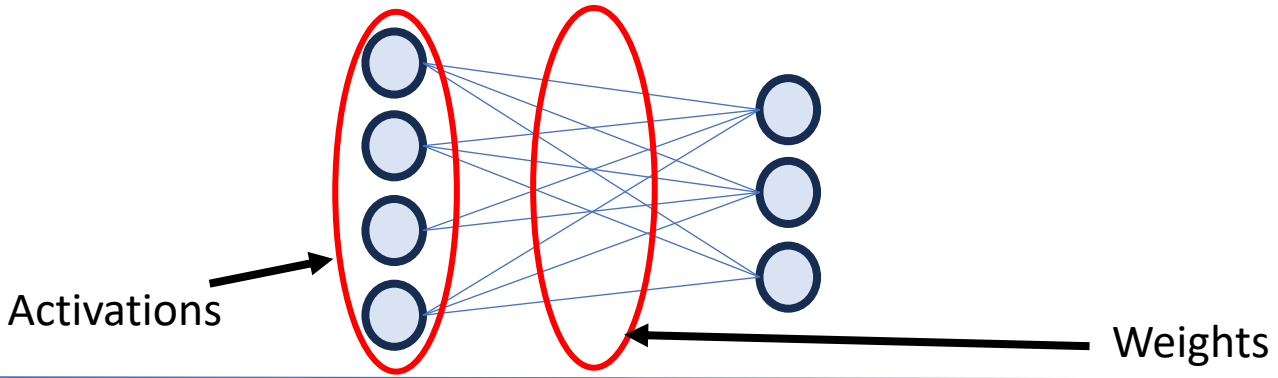
WANDA Pruning for Large Language Models

- Sun, Mingjie, Zhuang Liu, Anna Bair, and J. Zico Kolter. "A simple and effective pruning approach for large language models." arXiv preprint arXiv:2306.11695 (2023). – ICLR 2024









KAN: Kolmogorov-Arnold Networks

- Liu, Ziming, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y. Hou, and Max Tegmark. "KAN: Kolmogorov-Arnold Networks." arXiv preprint arXiv:2404.19756 (2024).

Model	**Multi-Layer Perceptron (MLP)**	**Kolmogorov-Arnold Network (KAN)**
Theorem	**Universal Approximation Theorem**	**Kolmogorov-Arnold Representation Theorem**
Formula (Shallow)	$f(\mathbf{x}) \approx \sum_{i=1}^{N(\epsilon)} a_i \sigma(\mathbf{w}_i \cdot \mathbf{x} + b_i)$	$f(\mathbf{x}) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$
Model (Shallow)	(a) *fixed* activation functions on *nodes*; *learnable* weights on *edges*	(b) *learnable* activation functions on *edges*; sum operation on *nodes*
Formula (Deep)	$\mathrm{MLP}(\mathbf{x}) = (\mathbf{W}_3 \circ \sigma_2 \circ \mathbf{W}_2 \circ \sigma_1 \circ \mathbf{W}_1)(\mathbf{x})$	$\mathrm{KAN}(\mathbf{x}) = (\mathbf{\Phi}_3 \circ \mathbf{\Phi}_2 \circ \mathbf{\Phi}_1)(\mathbf{x})$
Model (Deep)	(c) MLP(x); $\mathbf{W}_3$, σ_2, $\mathbf{W}_2$, σ_1, $\mathbf{W}_1$; *nonlinear, fixed*; *linear, learnable*; x	(d) KAN(x); $\mathbf{\Phi}_3$, $\mathbf{\Phi}_2$, $\mathbf{\Phi}_1$; *nonlinear, learnable*; x

KAN: Kolmogorov-Arnold Networks

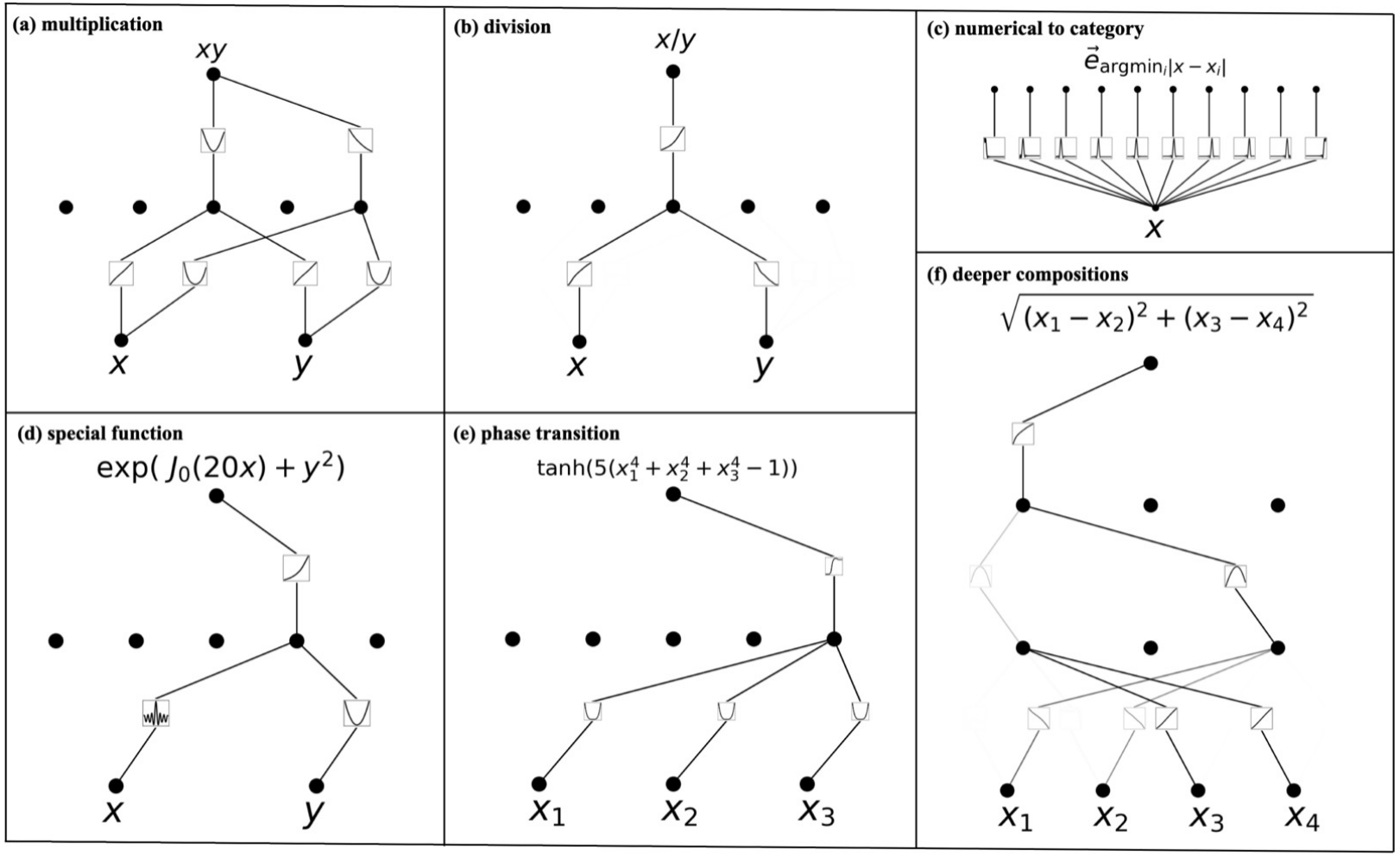
Abstract

Inspired by the Kolmogorov-Arnold representation theorem, we propose Kolmogorov-Arnold Networks (KANs) as promising alternatives to Multi-Layer Perceptrons (MLPs). While MLPs have *fixed* activation functions on *nodes* ("neurons"), KANs have *learnable* activation functions on *edges* ("weights"). KANs have no linear weights at all – every weight parameter is replaced by a univariate function parametrized as a spline. We show that this seemingly simple change makes KANs outperform MLPs in terms of accuracy and interpretability. For accuracy, much smaller KANs can achieve comparable or better accuracy than much larger MLPs in data fitting and PDE solving. Theoretically and empirically, KANs possess faster neural scaling laws than MLPs. For interpretability, KANs can be intuitively visualized and can easily interact with human users. Through two examples in mathematics and physics, KANs are shown to be useful "collaborators" helping scientists (re)discover mathematical and physical laws. In summary, KANs are promising alternatives for MLPs, opening opportunities for further improving today's deep learning models which rely heavily on MLPs.

parameter becomes KAN's spline function. Fortunately, KANs usually allow much smaller computation graphs than MLPs. For example, we show that for PDE solving, a 2-Layer width-10 KAN is **100 times more accurate** than a 4-Layer width-100 MLP (10^{-7} vs 10^{-5} MSE) and **100 times more parameter efficient** (10^2 vs 10^4 parameters).

Interpretability of KAN and Pruning

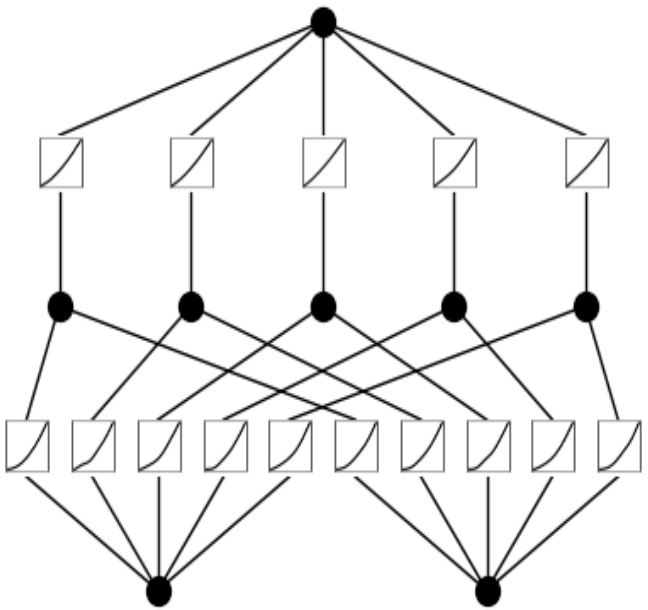
- **https://github.com/KindXiaoming/pykan**

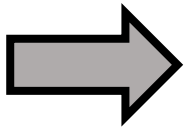


Pruning in KAN

- **https://github.com/KindXiaoming/pykan**

```
# plot KAN at initialization
model(dataset['train_input']);
model.plot(beta=100)
```





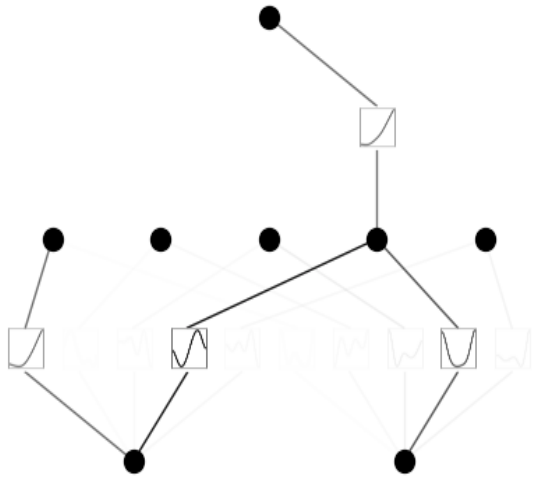
Train KAN with sparsity regularization

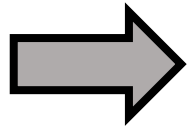
```
# train the model
model.train(dataset, opt="LBFGS", steps=20, lamb=0.01, lamb_entropy=10.);
```

```
train loss: 1.57e-01 | test loss: 1.31e-01 | reg: 2.05e+01 : 100%|██| 20/20 [
```

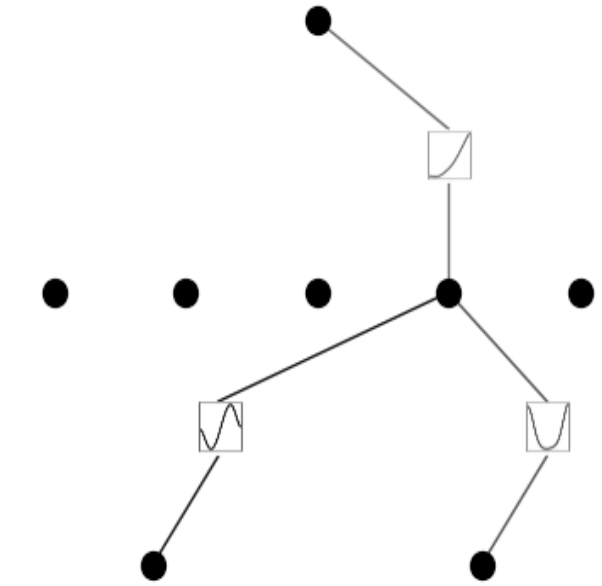
Plot trained KAN

```
model.plot()
```





```
model.prune()
model.plot(mask=True)
```



Prune KAN and replot (get a smaller shape)

Quantization Aware Training (QAT)

- **Post-Training Quantization (PTQ):** Quantization is applied after training
- **Quantization Aware Training (QAT)**: An approach that integrates quantization into the neural network training. It simulates lower precision arithmetic during the training phase
- **Goal of QAT**: Ensure that the model is robust to the effects of quantization, maintaining high accuracy when deployed with reduced precision.
- QAT is particularly important for deploying models on edge devices with limited computational resources, yet require high accuracy (e.g., Fall Detection)

How QAT Works

- **Training with Quantization:**
  - Both weights and activations are quantized to lower bit-widths during the forward and backward passes of training.
  - Allows the model to "learn" the quantization noise/error and account for it during training

- **Fine-tuning Process**:
  - Initially, the model is trained using floating point weights to a certain accuracy level
  - It is then fine-tuned with quantization enabled, which adjusts the model parameters to compensate for the loss of precision due to quantization.

PTQ Vs. QAT

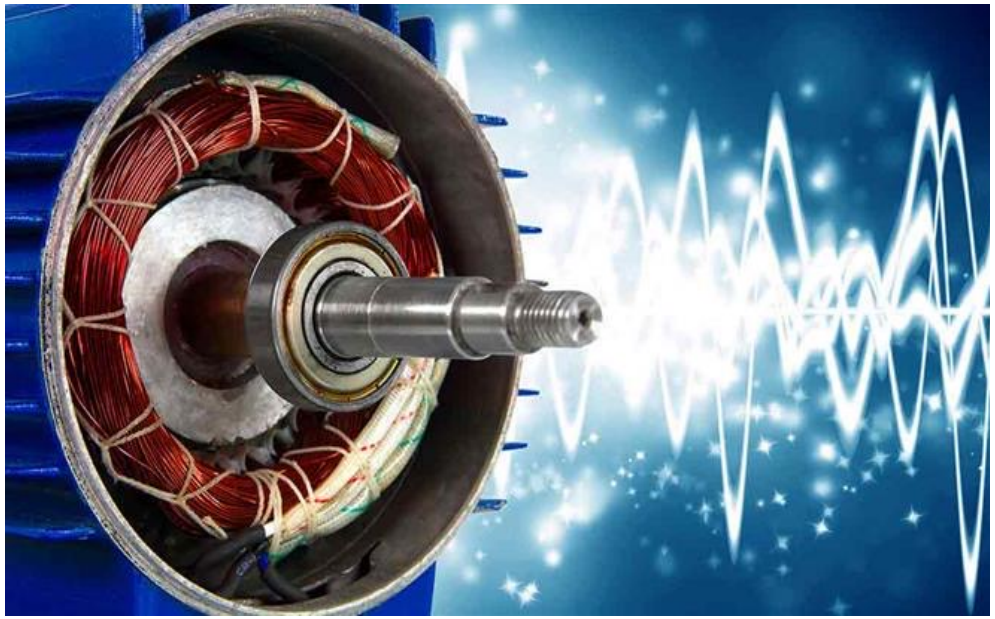
Quantization Type	Best Suited For	Complexity and Time	Performance Comparison
PTQ (Post-Training Quantization)	Scenarios with limited resources/time for retraining; slight accuracy degradation is acceptable.	Low complexity; can be applied quickly without retraining the model.	May cause significant accuracy drops, especially in sensitive models.
QAT (Quantization-Aware Training)	High accuracy is crucial; performance-critical applications.	High complexity; integrates quantization into the training cycle, requiring more resources and time.	Maintains higher accuracy by accounting for quantization during training.

Motion Classification

- Why should we classify motion?
- What are the application domains where motion classification can be useful?
- What are the input data types that can effectively help to classify motion data?
- How is motion classification similar or different to gesture recognition?

Input Video Clip	Motion Type Predicted
Cart-wheel 	***Projectile***
Jogging 	***Linear***
Diving 	***Projectile***
Drinking 	***Local***
Fencing 	***Random***

Classification of Different Data



Audio Classification

Image Classification

Motion Classification

What do we choose as input for motion classification?

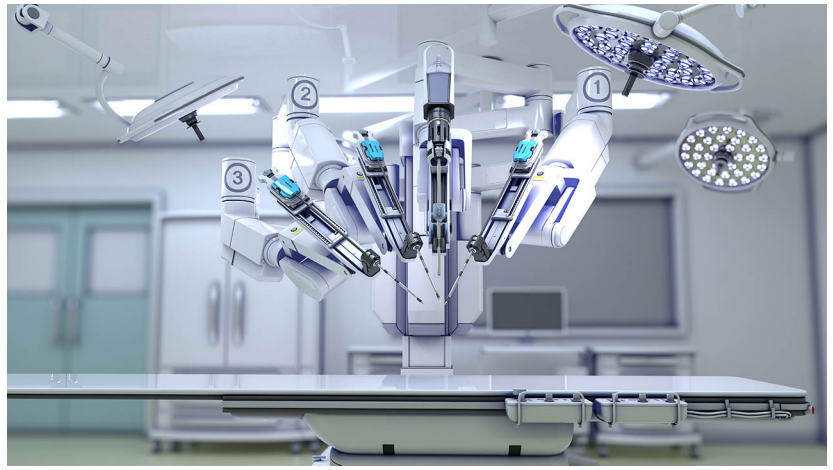
Depends on the application!

Motion Classification

Classify Washing Machine cycles

Classify poses for games/ yoga/ martial arts

Applications of Motion Classification

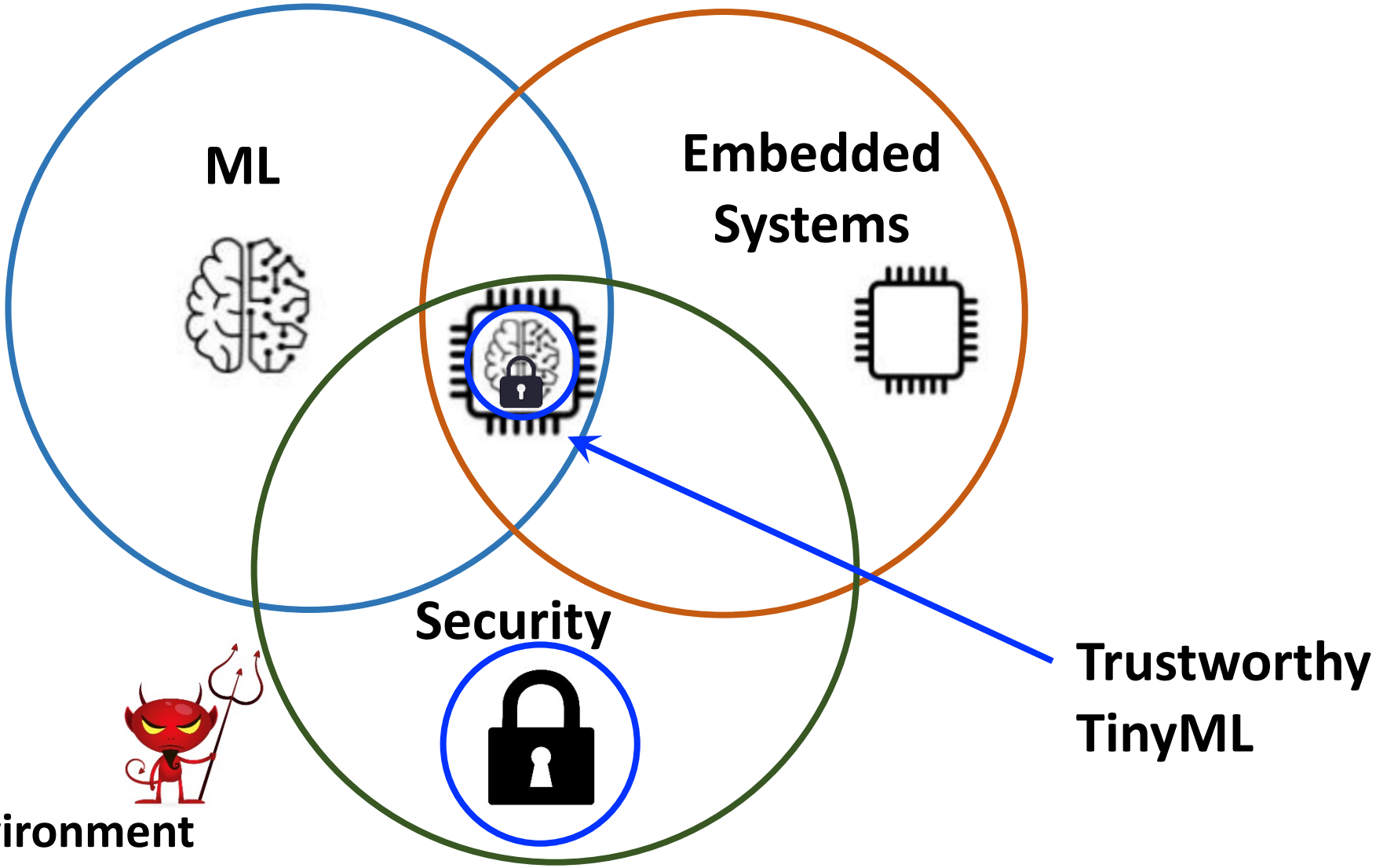


Surgical Robotics

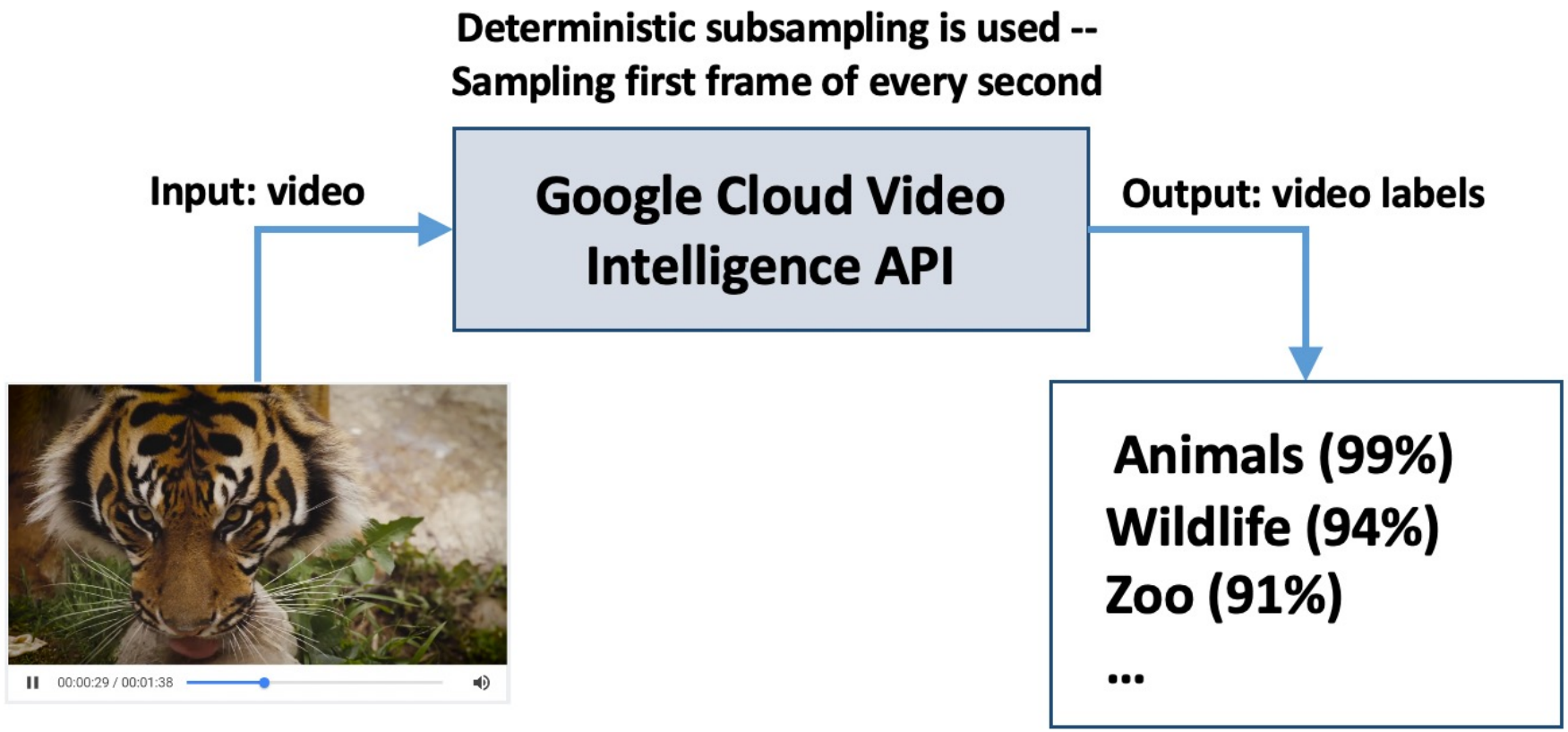
Transportation of Freight

Motion Capture Technology

Can we attack motion classifier applications?

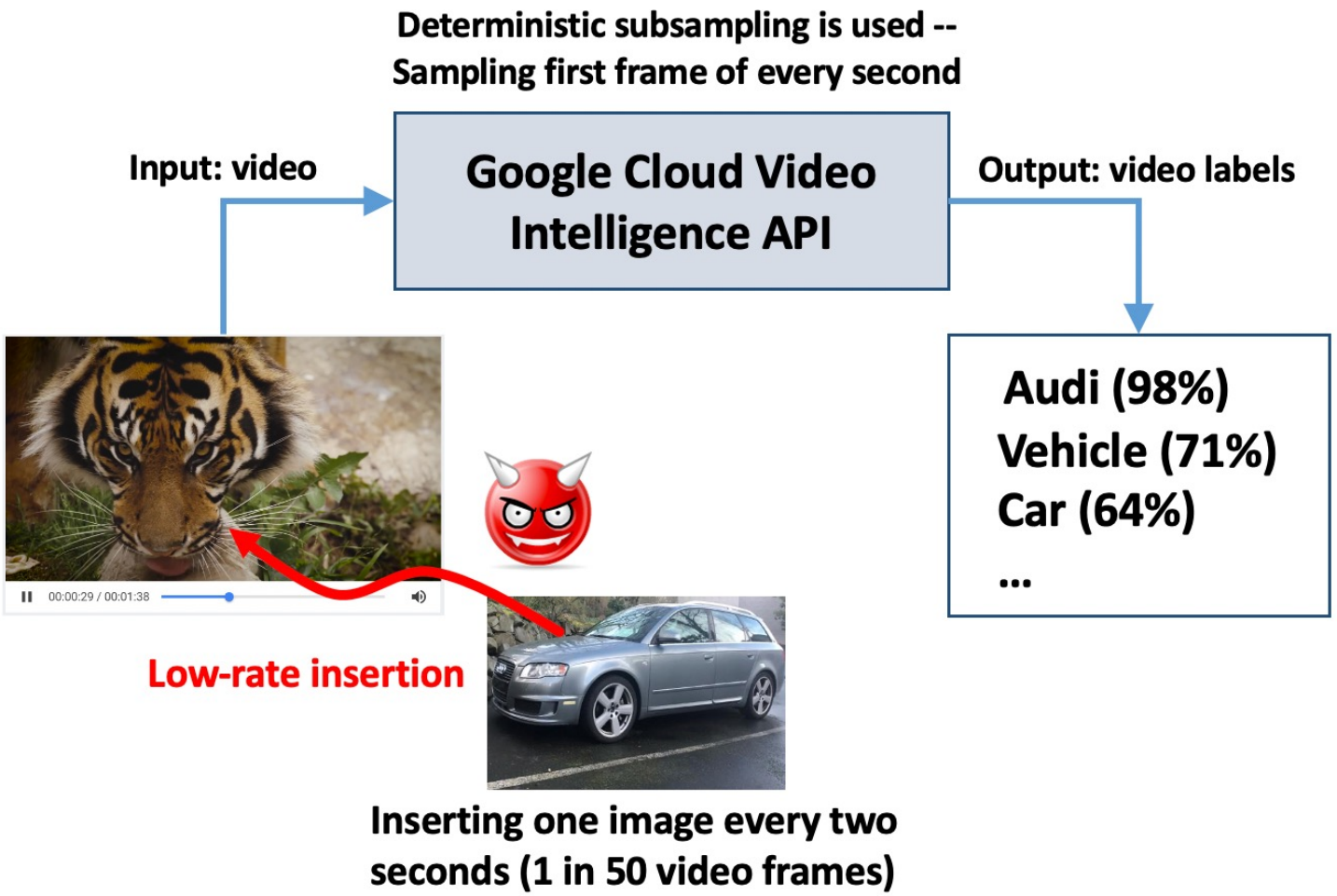


Google Video Analysis with DNN



Hosseini, H., Xiao, B. and Poovendran, R., "Deceiving Google's Cloud Video Intelligence API Built for Summarizing Videos." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017.

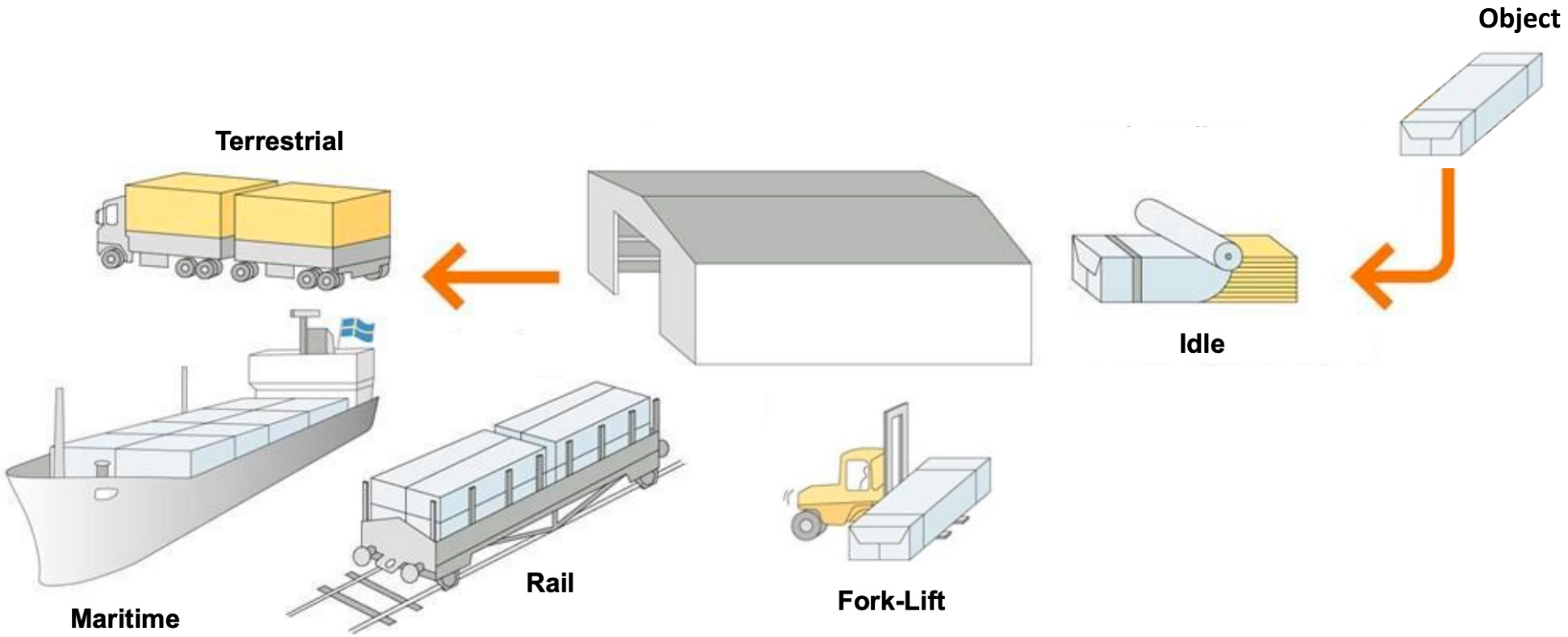
Decision Time Attack---With Domain Knowledge



Hosseini, H., Xiao, B. and Poovendran, R., "Deceiving Google's Cloud Video Intelligence API Built for Summarizing Videos." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017.

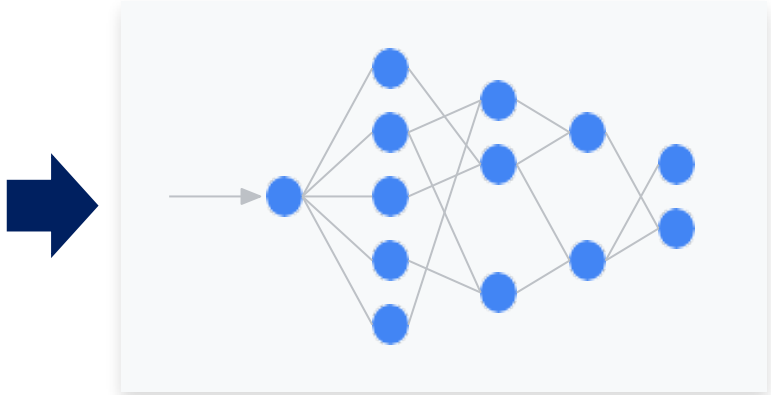
Motion Classification for Transport

- Motion classification for transport is essential to calculate the stress or force on objects in transport.
- Objects experience different types of stress based on the conditions of the route.



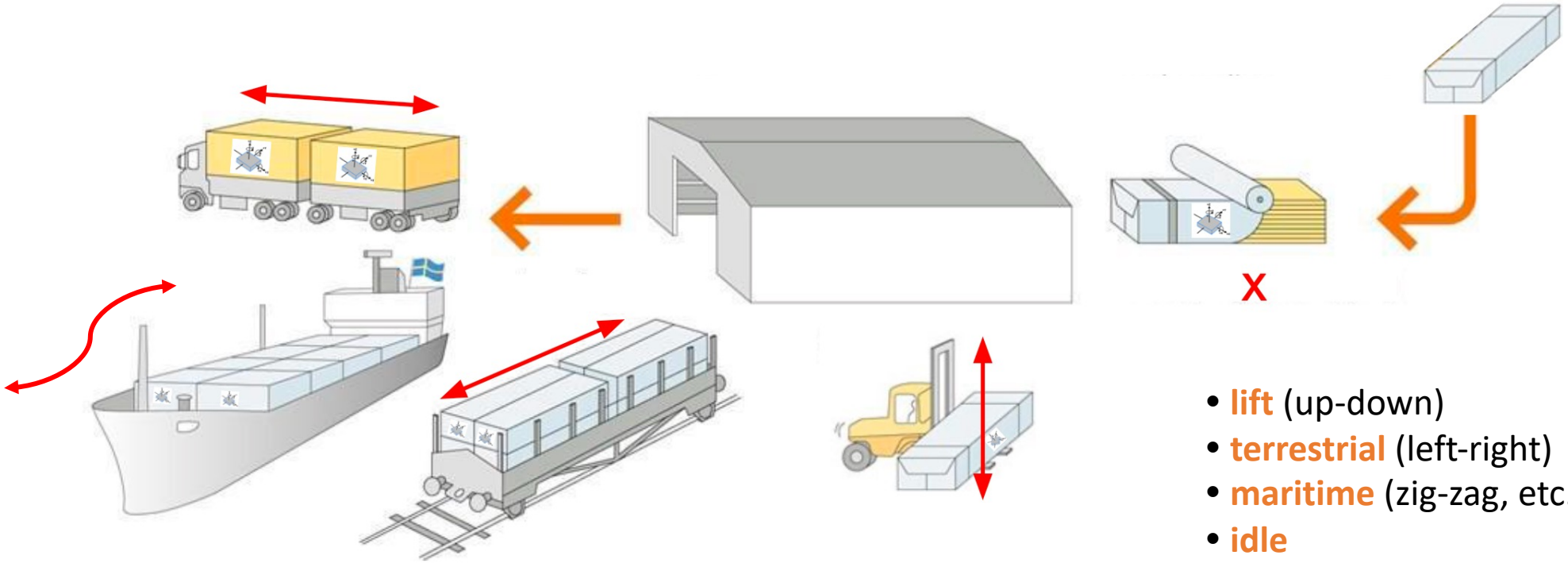
Challenges in implementation

- On-device Computing is necessary since streaming to the cloud might not be possible due to poor connection during freight transport.



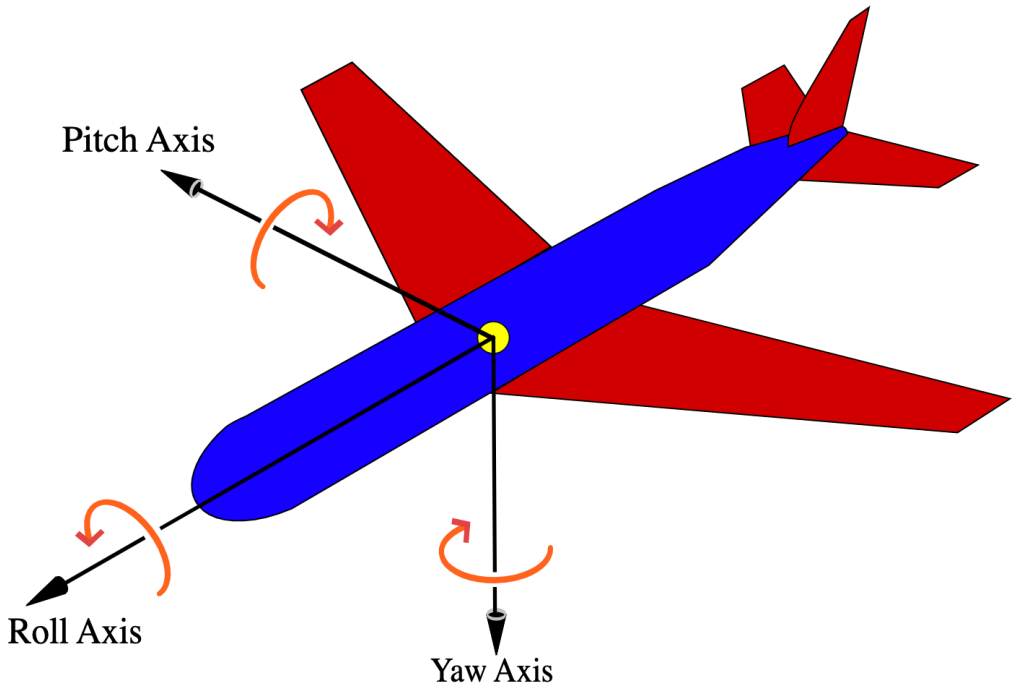
Degrees of Freedom

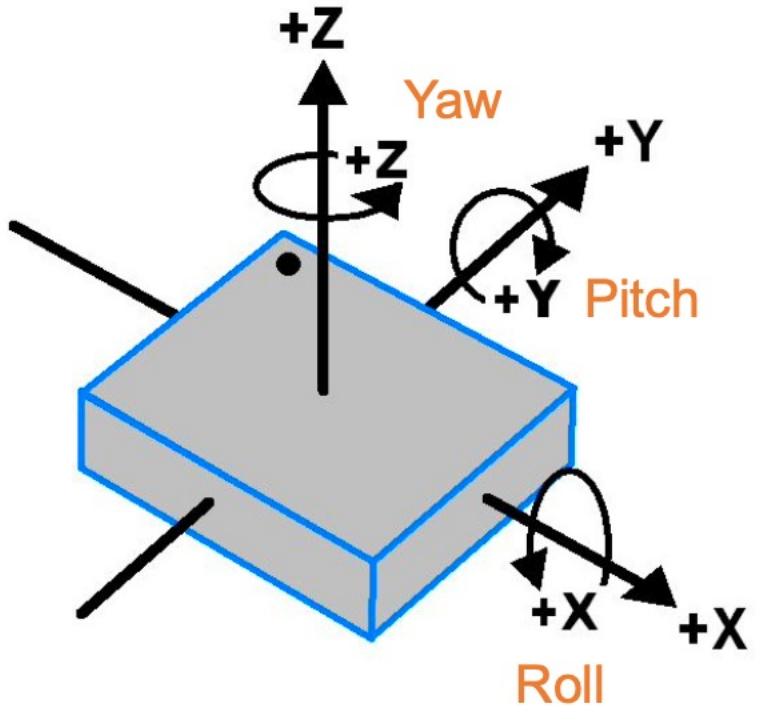
- Degrees of freedom (DoF) refer to the number of basic ways a rigid object can move through 3D space.
- We define various classes based on the transport type and the degree of freedom for object movement in each of the types.



- **lift** (up-down)
- **terrestrial** (left-right)
- **maritime** (zig-zag, etc.)
- **idle**

Inertial Measurement Units (IMU)

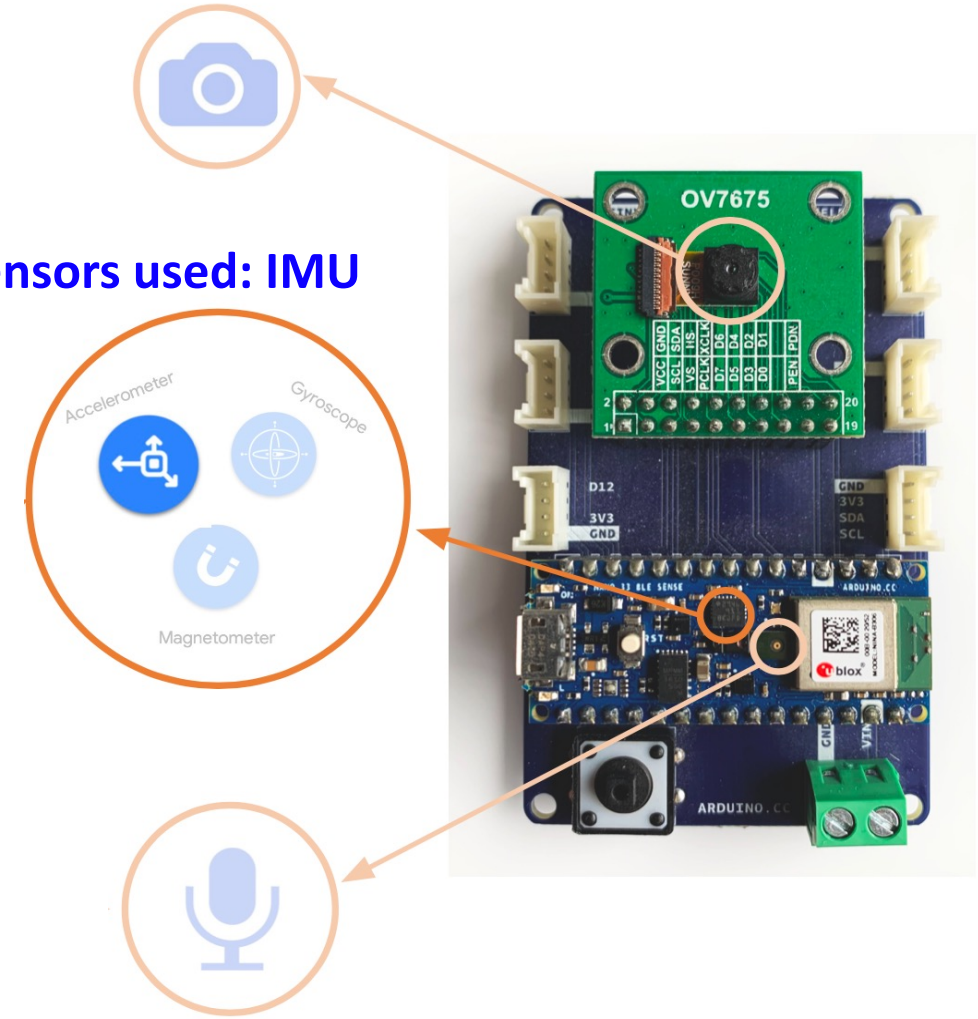




Yaw, Pitch and Roll measurements through Accelerometer and Gyroscope data

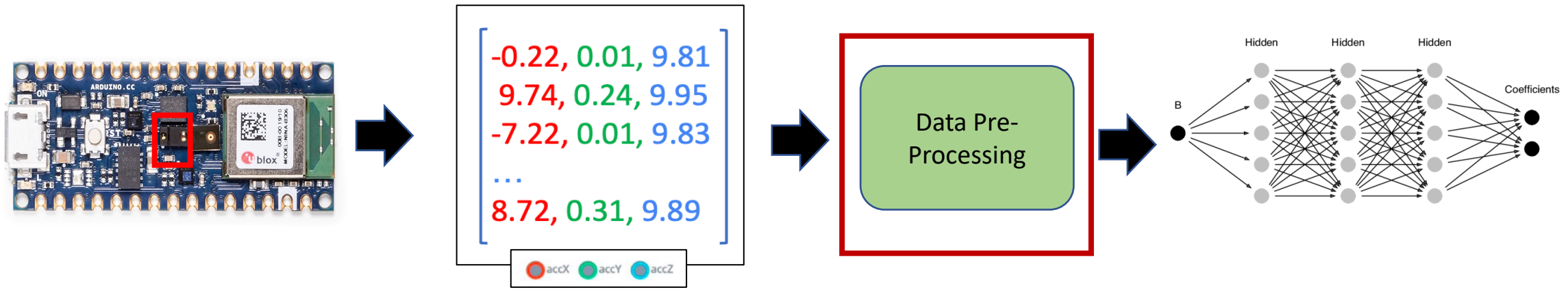
TinyML IMU Sensors

Sensors used: IMU



End-to-End flow of Motion Classifier

- End-to-End flow from capturing the motion data to classifying the different types of motion.
- We use a DNN Classifier as our model.



Input data captured by IMU

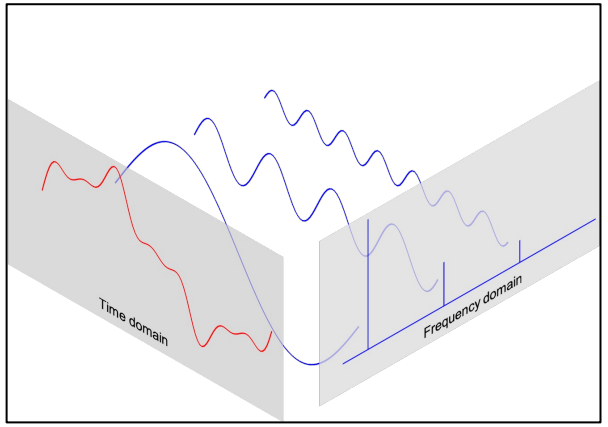
Raw Features of Accelerometer

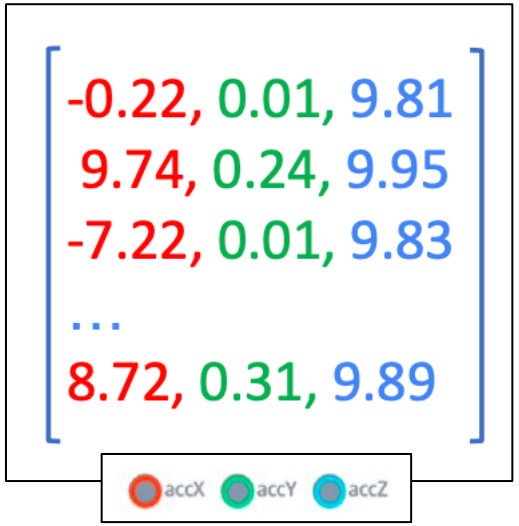
Motion Classifier (NN)

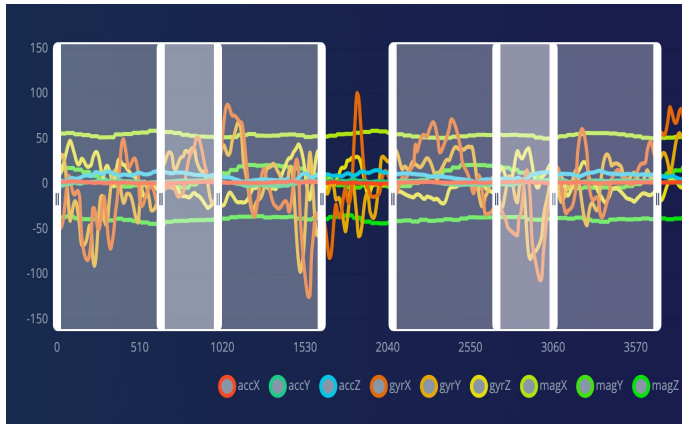
Data Pre-Processing

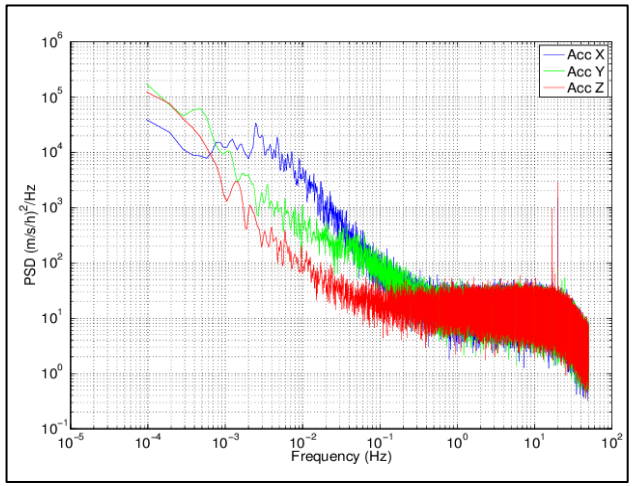
Data pre-processing steps:

Fast Fourier Transform (FFT) ← **Convert Raw Features to Time Series Data** → **Sliding window procedure for feature extraction** → **Generate Power Spectral Density (PSD)**





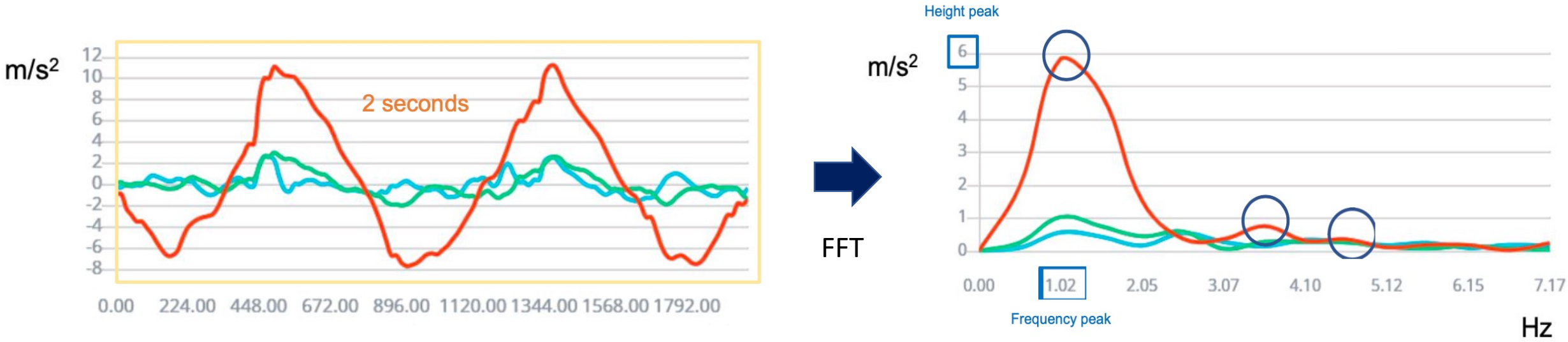




Source: https://heartbeat.comet.ml/neural-network-pruning-research-review-2020-bc21a77f0295

FFT for Accelerometer Data

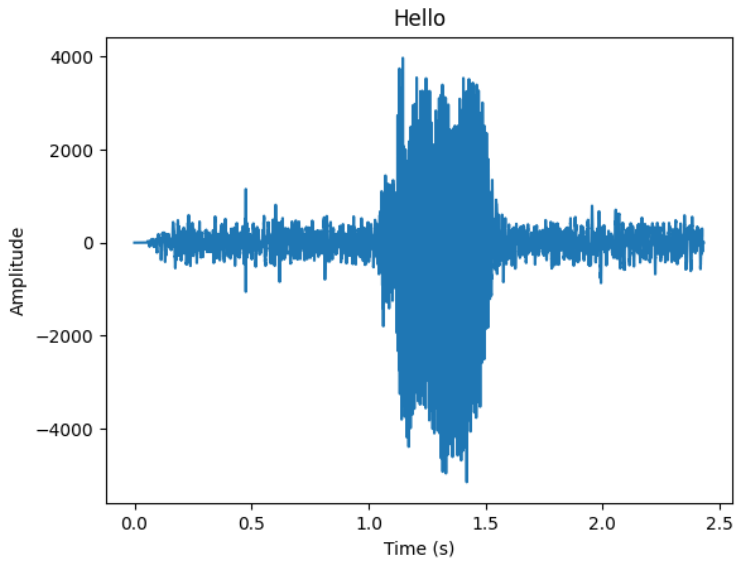
- Calculating FFT for accelerometer data extracts **Height Peak** and **Frequency Peak** parameter from the raw data which can then be fed to the NN for gaining a compressed model to implement on tiny devices.



Recall.. Fast Fourier Transform (FFT)

- The time series signal (Time v. Amplitude) is transformed into the frequency domain (Frequency v. Amplitude) using Discrete Fourier transform.
- The Fast Fourier Transform (FFT) is an efficient algorithm to calculate the DFT of a sequence.

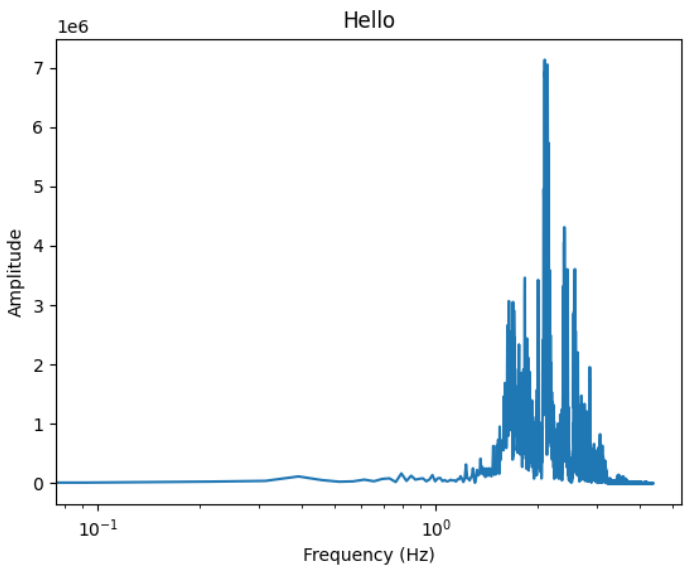
Word: Hello



Speech Signal in Time Domain

Fast Fourier Transform

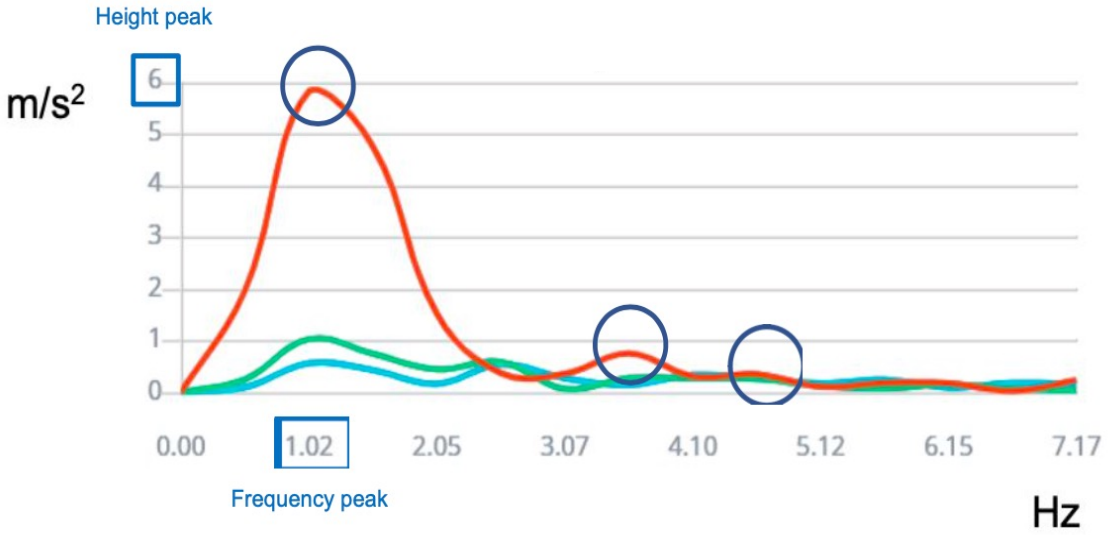
Word: Hello



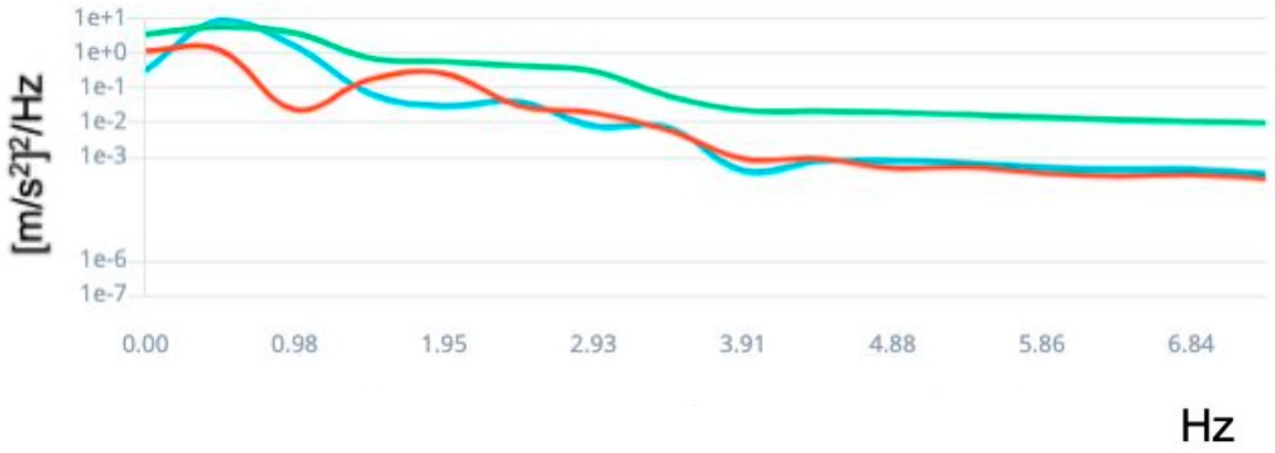
Speech Signal in Frequency Domain

Power Spectral Density (PSD)

Calculating PSD from FFT of the signal:



Fast Fourier Transform



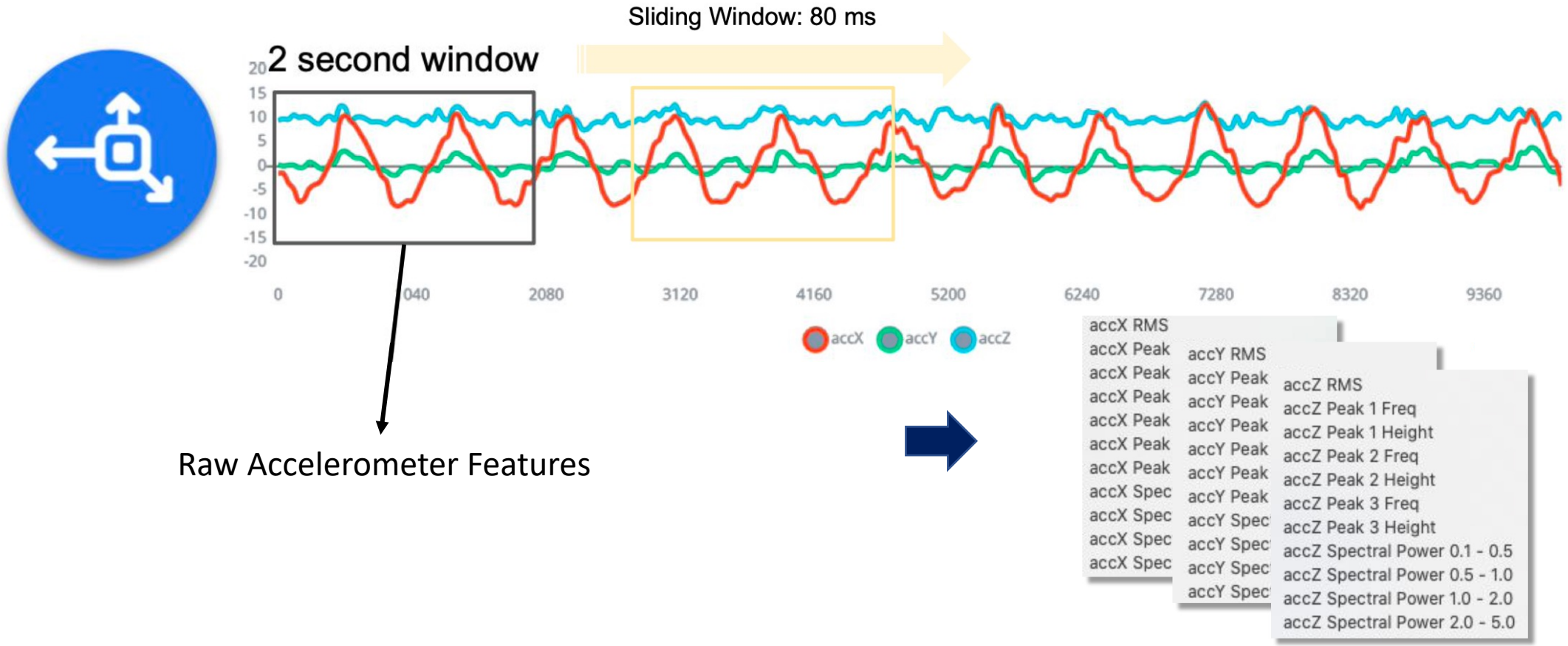
Power Spectral Density

4 frequency bins/axis

Source: https://blog.endaq.com/why-the-power-spectral-density-psd-is-the-gold-standard-of-vibration-analysis

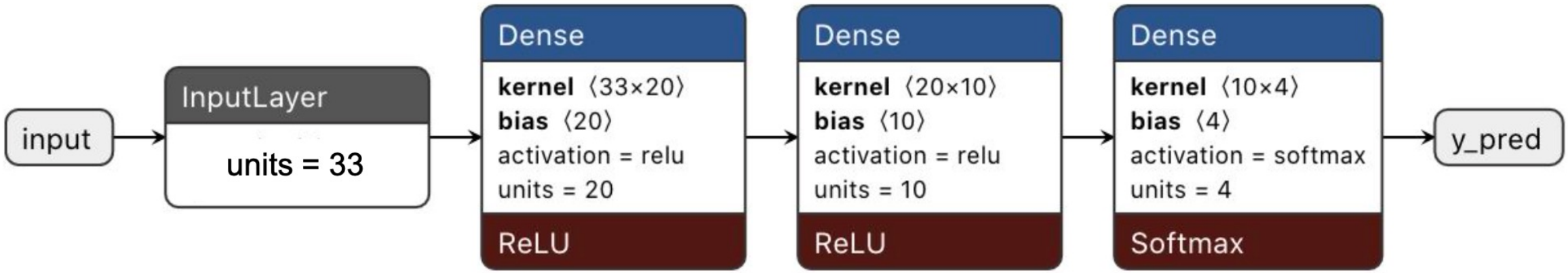
Sliding Window for Accelerometer Data

- For our motion classifier application, we will be using Sliding Window procedure for extracting important features from the raw accelerometer data.

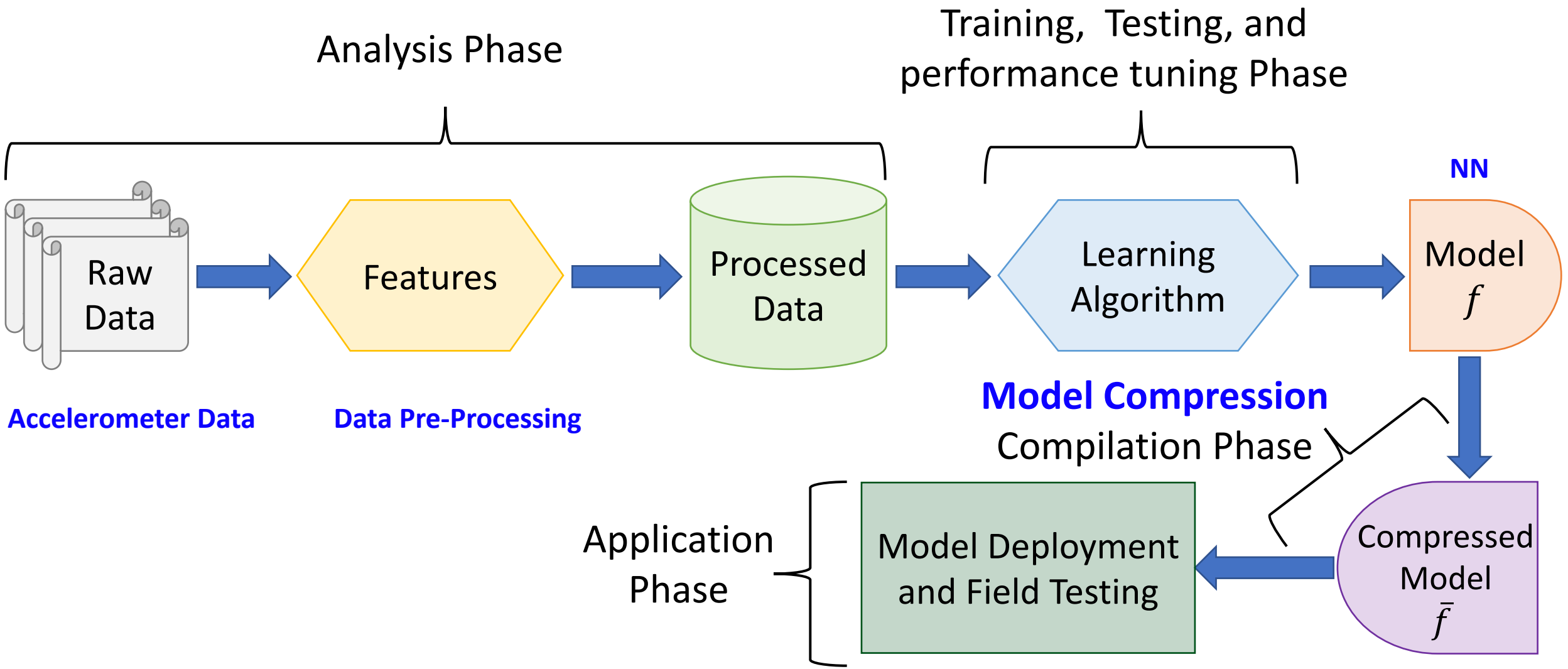


Deep Neural Network (DNN) Classifier

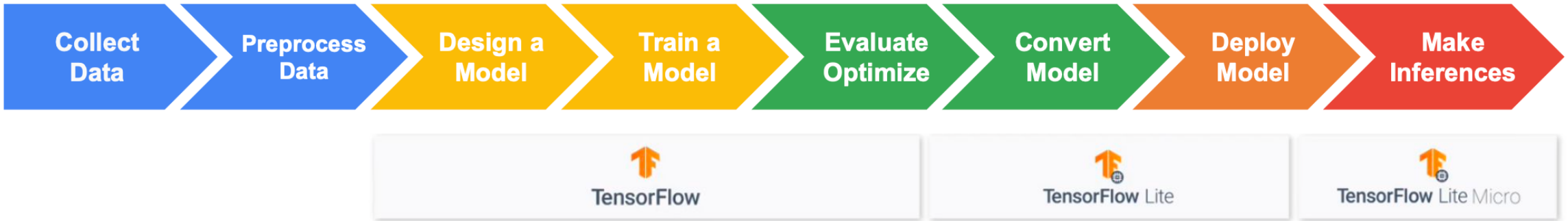
- The DNN classifies transportation data into four classes:
  - **lift**
  - **terrestrial**
  - **maritime**
  - **idle**



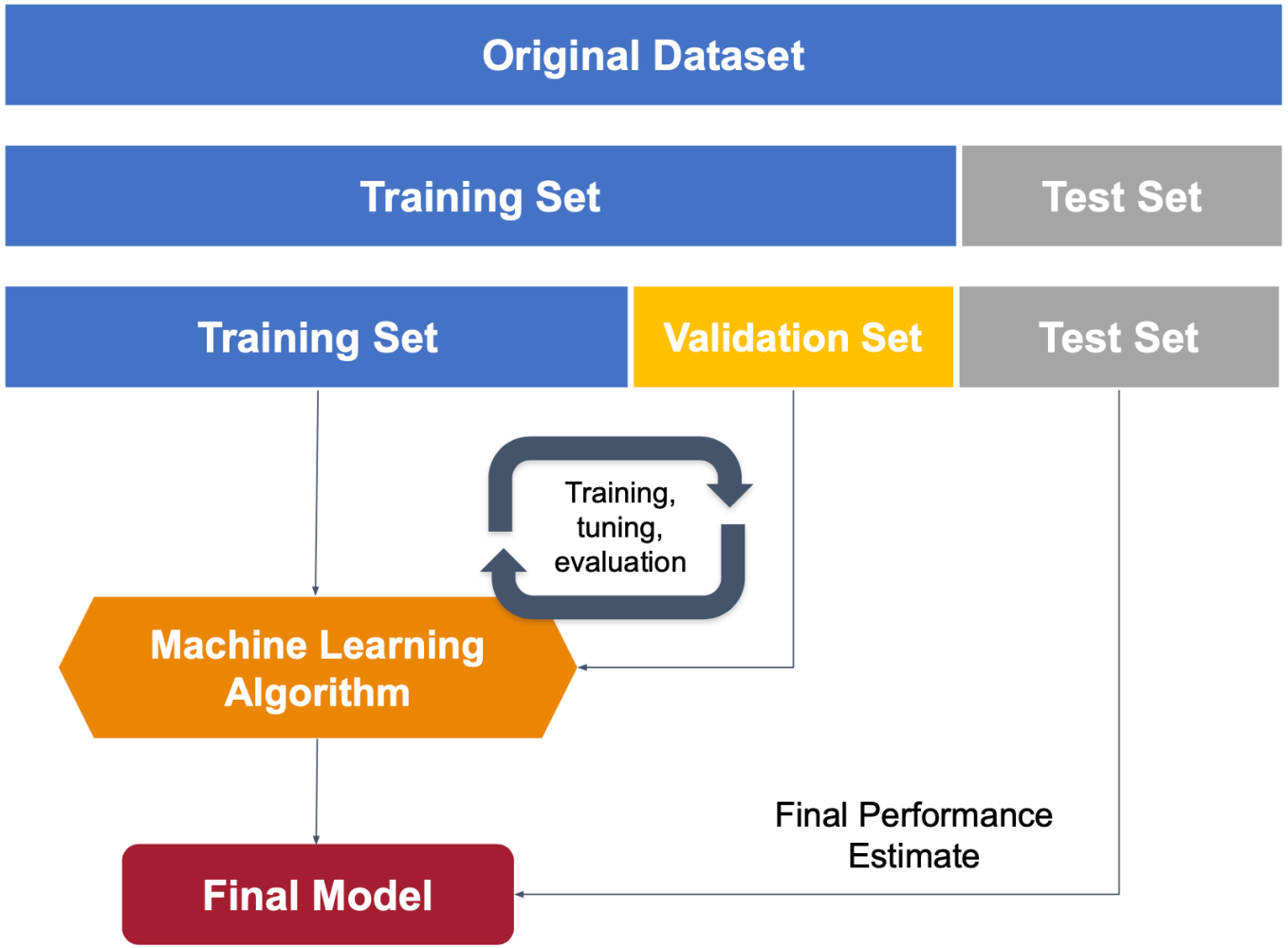
A Schematic View of TinyML Pipeline



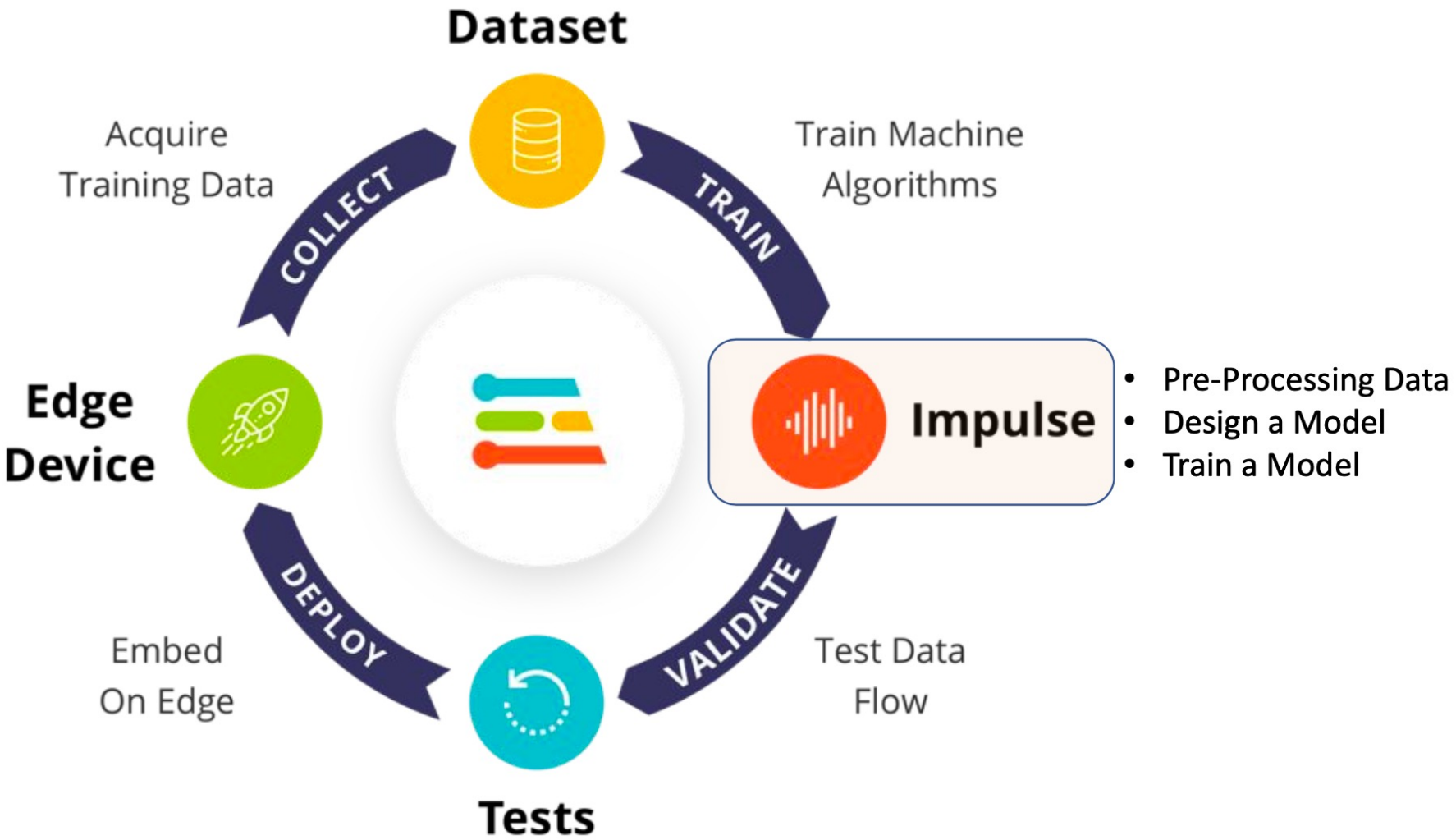
TinyML for Motion Classification



Edge Impulse - Model



Edge Impulse - Model



5 min Break

Today’s Lab

- Both Software and Hardware Lab!
- Accelerometer data collection for motion classification
- Computing FFT, Spectrogram, and PSD from time series data
- ML Models for Motion Classifier
- Implement the Model with TinyML Kits

Lab 7 - Software - Road Map

1. Installing Arduino CLI
2. Installing Edge Impulse Firmware on Arduino
3. Data Collection and Labelling
4. Splitting Data
5. Spectral Analysis
6. Training the ML model (DNN)

Lab 7 - Software - Road Map

7. Evaluation of ML Model Performance

8. Building ML model

Lab 7 – Software

We will use Edge Impulse for this Lab.
Website: https://www.edgeimpulse.com/
(You can login in using the account we created for the previous Edge Impulse lab)

Create a new Project on Edge Impulse and provide the title as “Motion Classification for Transport”.

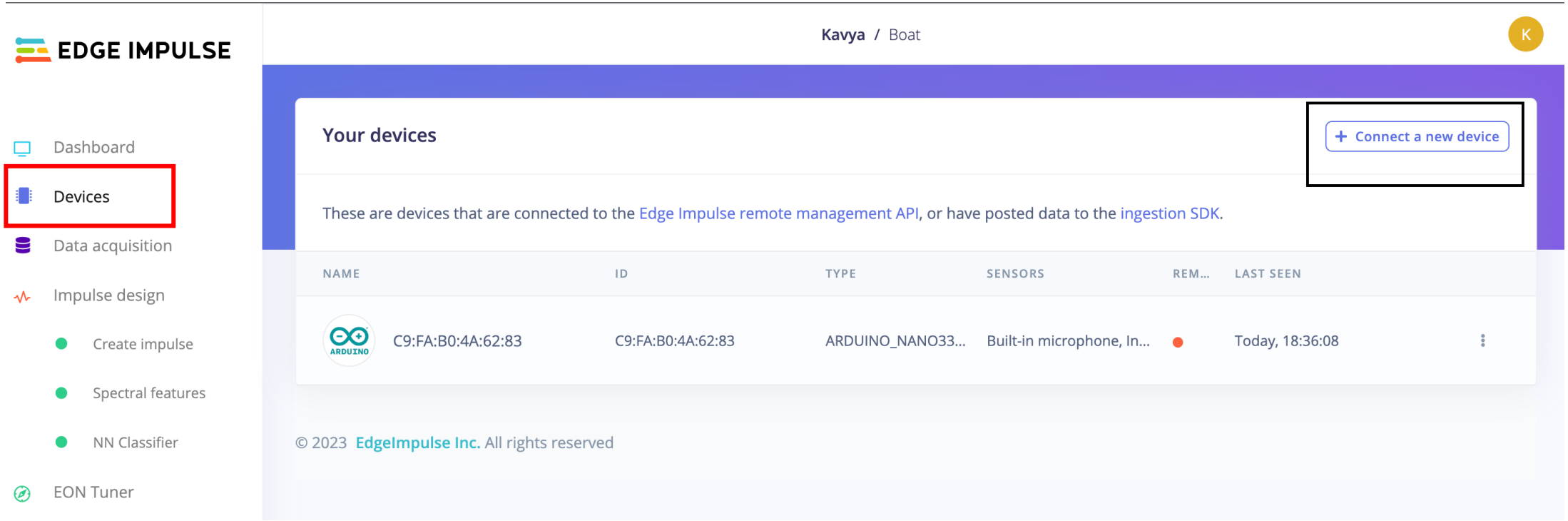
Lab 7 - Hardware - Road Map

1. Build the Arduino library using Edge Impulse.

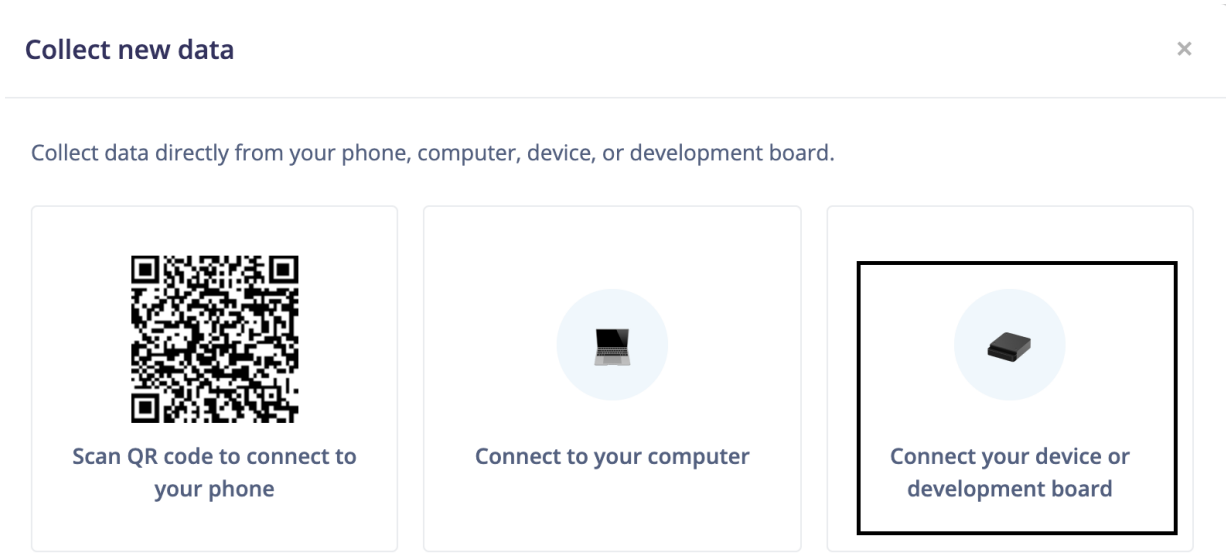
2. Compile and upload the sketch to the board.

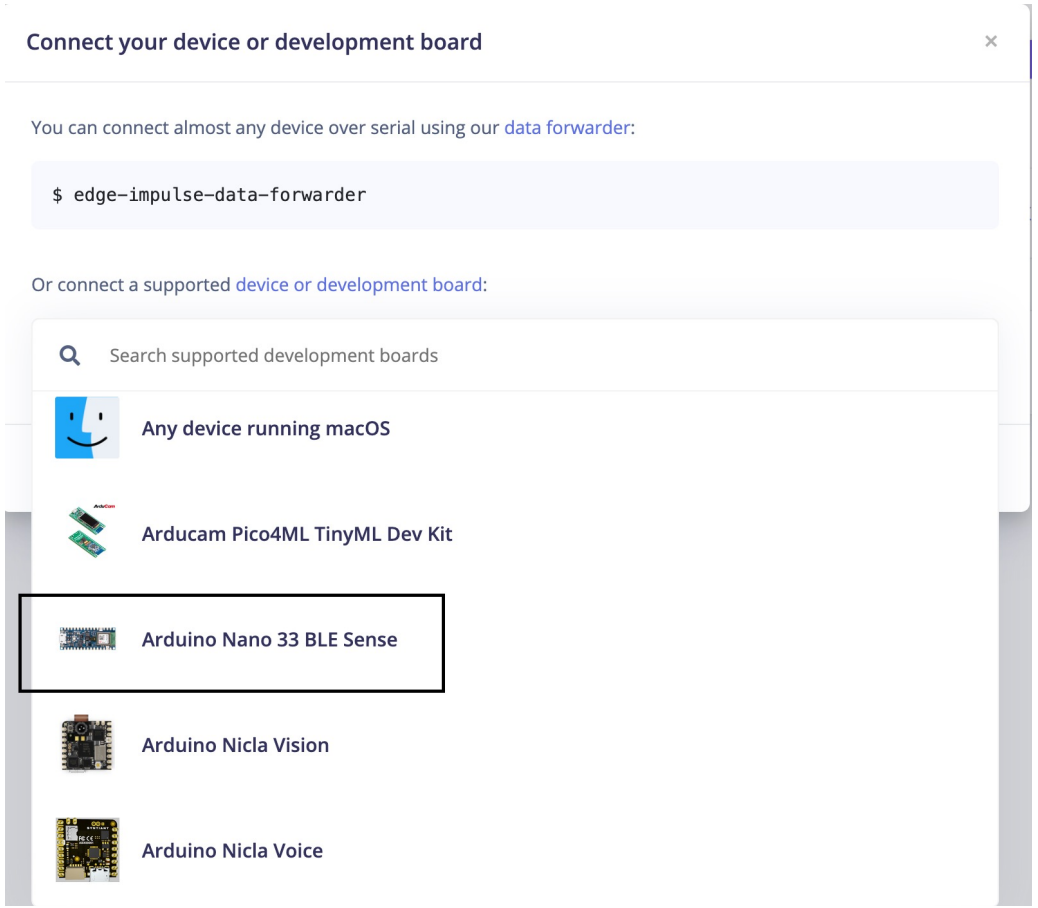
3. Open Serial Monitor and test the classification.

Software Lab – Connecting the Device



Software Lab - Firmware Installation





Software Lab – Arduino CLI

If you're on Windows:

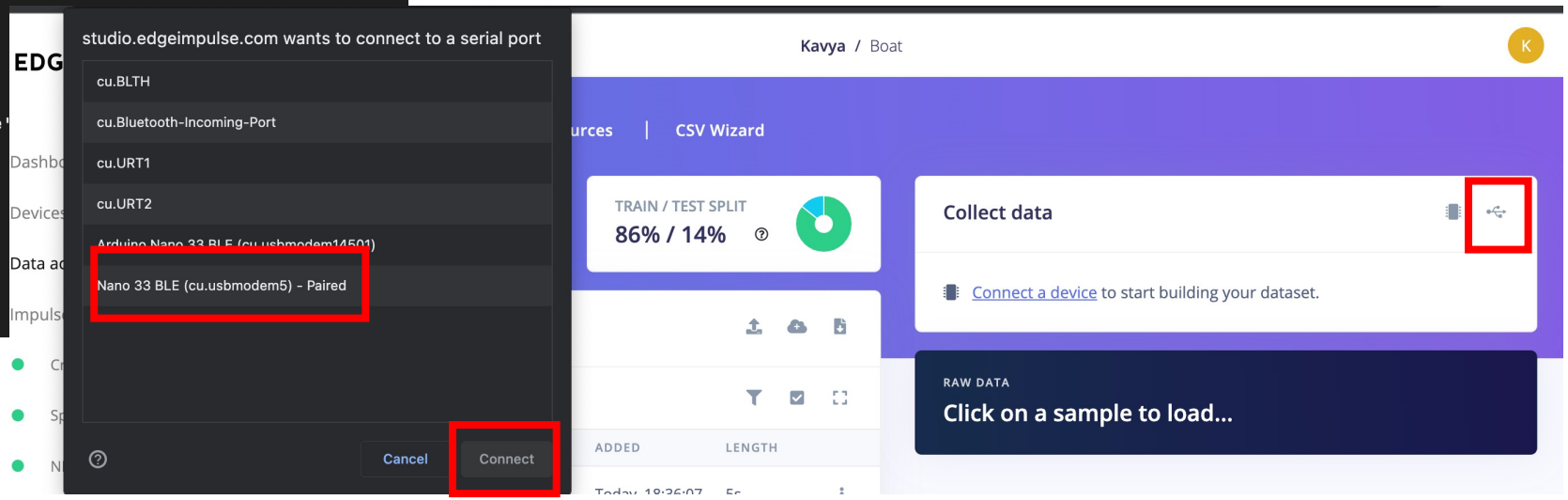
- Unzip the .zip file to C:\Program Files\arduino-cli
- Open System Properties > Advanced > Environment Variables
- Path under "user variables" > Edit
- Add C:\Program Files\arduino-cli

If you're on macOS:

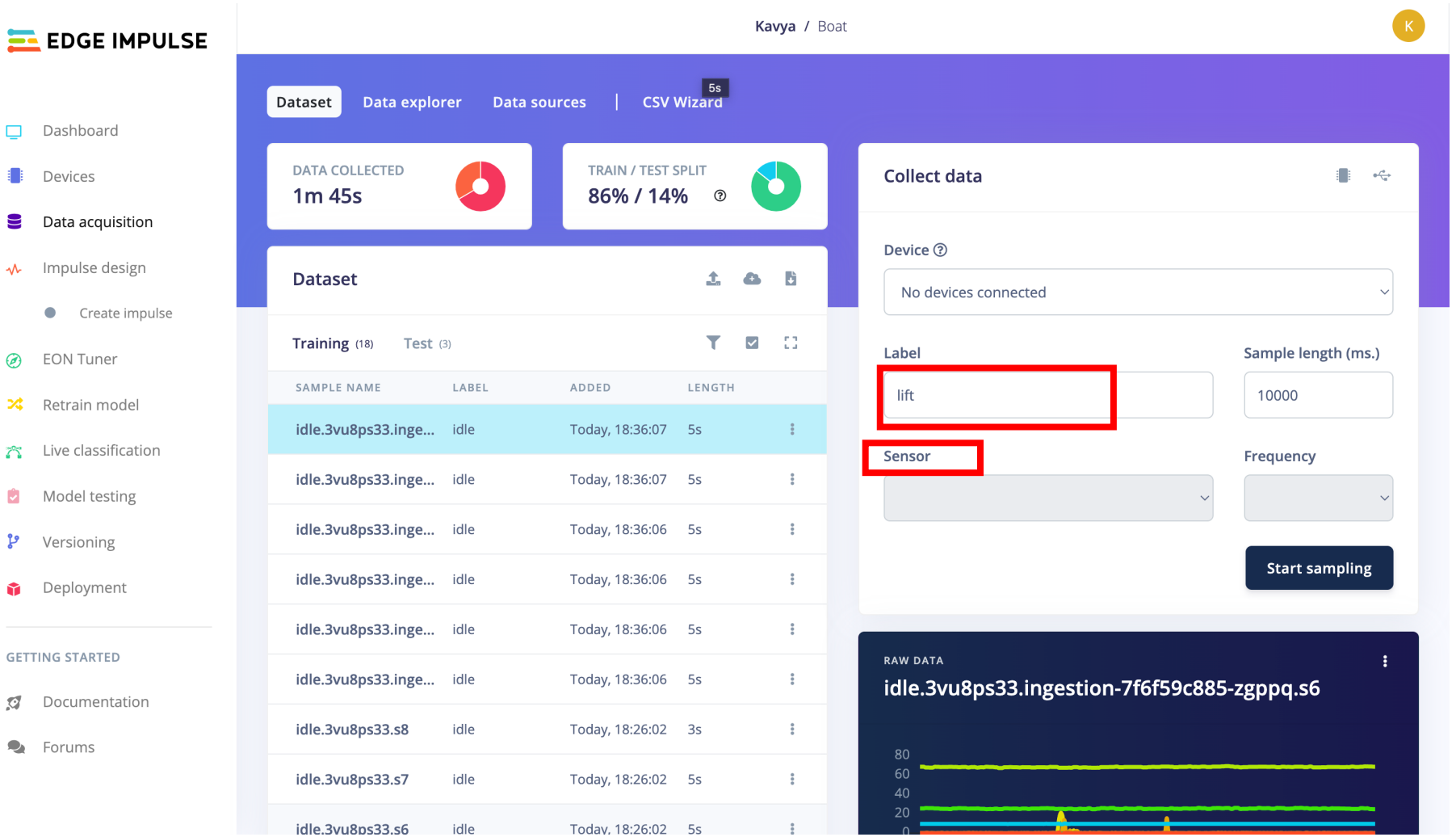
- Use the curl method or homebrew shown on the installation page: https://arduino.github.io/arduino-cli/0.21/installation/
- If you use the curl method, it likely installs arduino-cli to ~/bin. That might not be on your path. So, you might need to run: export PATH=$PATH:~/bin

Software Lab - Arduino CLI and Firmware

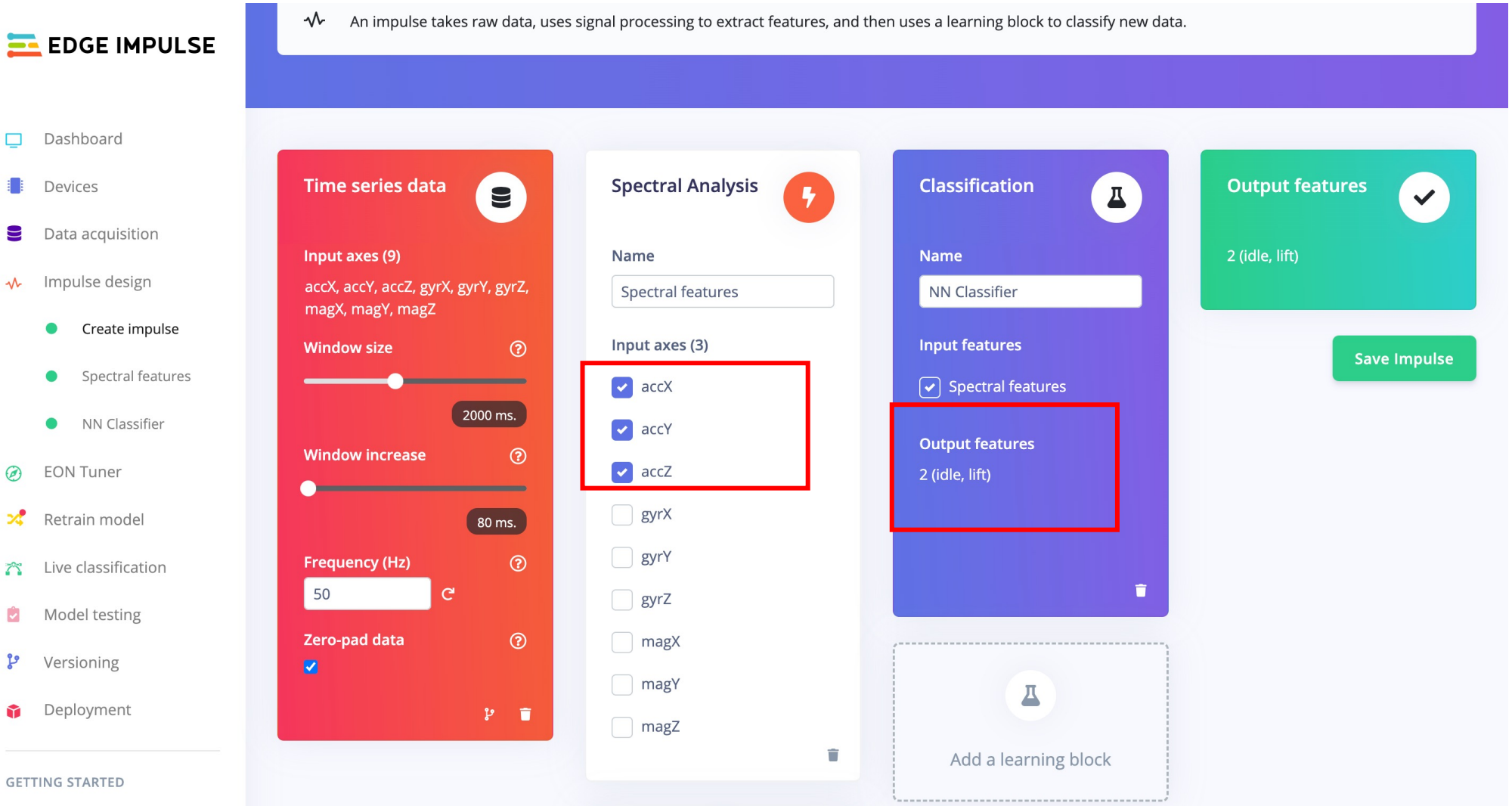
```
/Users/Admin/Downloads/arduino-nano-33-ble-sense\ \(2\)/flash_mac.command ; exit;
(base) Admin@Sarats-MacBook-Pro ~ % /Users/Admin/Downloads/arduino-nano-33-ble-sense\ \(2\)/flash_mac.command ; exit
You're using an untested version of Arduino CLI, this might cause issues (found: 0.32.2, expected: 0.18.x)
Finding Arduino Mbed core...
Finding Arduino Mbed OK
Finding Arduino Nano 33 BLE...
Finding Arduino Nano 33 BLE OK
Flashing board...
TOUCH: error during reset: opening port at 1200bps: Serial port busy
Device       : nRF52840-QIAA
Version      : Arduino Bootloader (SAM-BA extended) 2.0 [Arduino:IKXYZ]
Address      : 0x0
Pages        : 256
Page Size    : 4096 bytes
Total Size   : 1024KB
Planes       : 1
Lock Regions : 0
Locked       : none
Security     : false
Erase flash

Done in 0.001 seconds
Write 352560 bytes to flash (87 pages)
[==============================] 100% (87/87 pages)
Done in 13.914 seconds

Flashed your Arduino Nano 33 BLE development board.
To set up your development with Edge Impulse, run 'edge-impulse-daemon'
To run your impulse on your development board, run 'edge-impulse-run-impulse'

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```



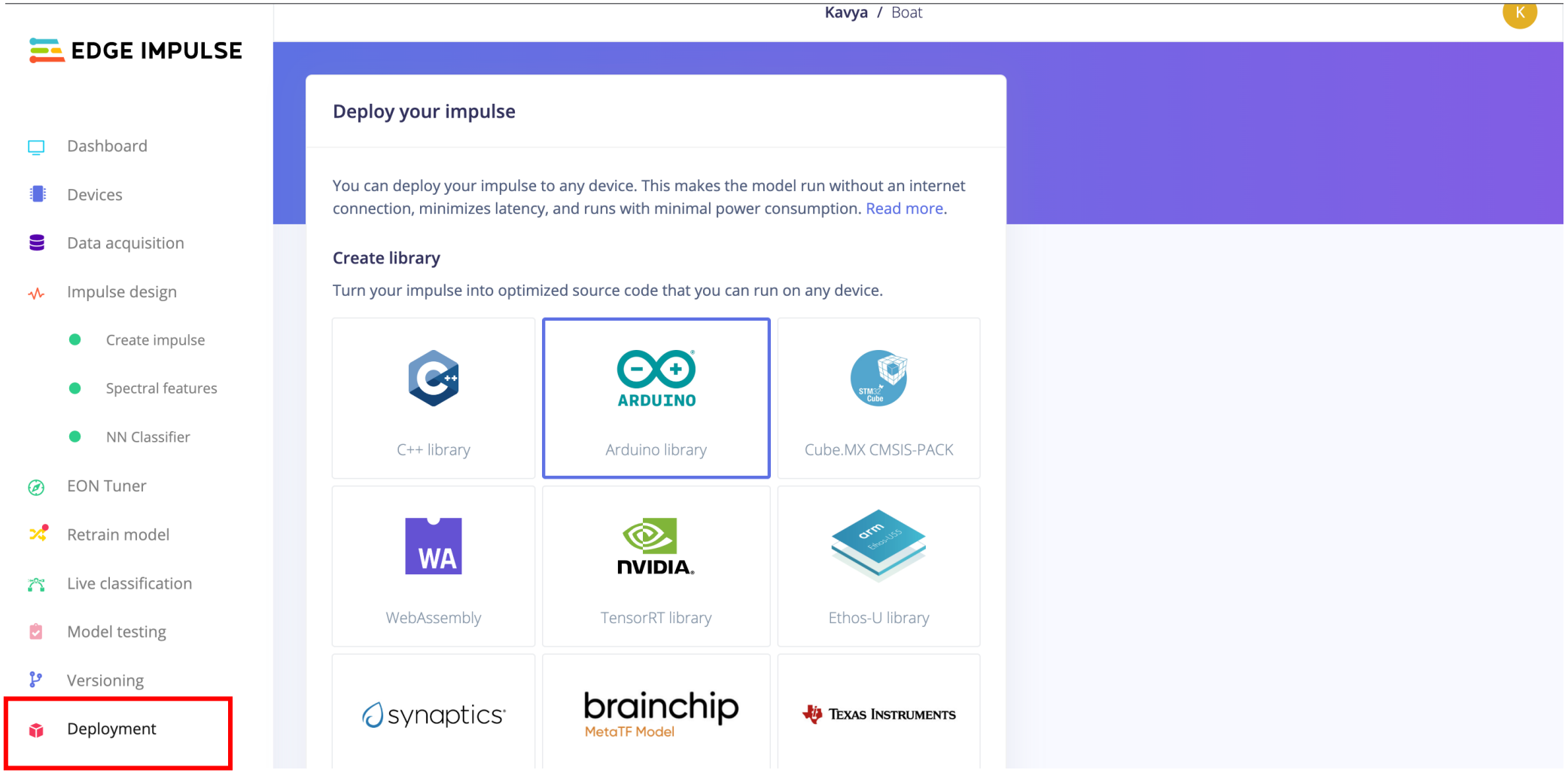
Software Lab – Data Collection



Software Lab – Impulse Design



Hardware Lab - Deployment



Hardware Lab – Arduino Testing