ISUW2020

AI/ML application to power system protection

OD Naidu, Principal Scientist

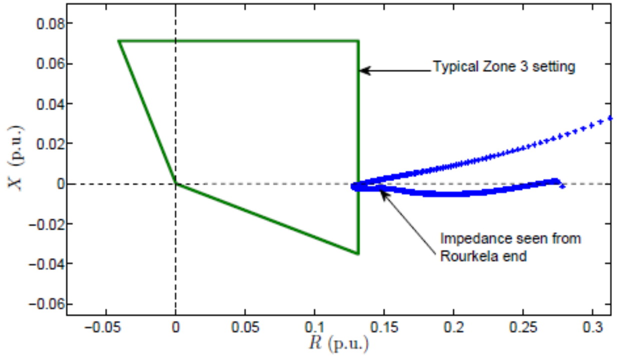
Major blackouts in Asia

Year and place of the blackout	People affected (million)
2012 India	640
2001 India	230
2014 Bangladesh	150
2015 Pakistan	140
2019 Java	120
2005 Java-Bali	100

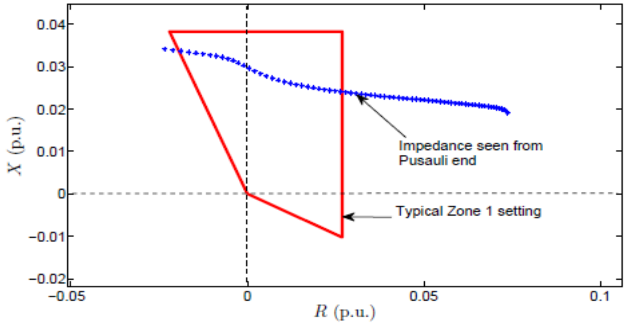
Major Causes:

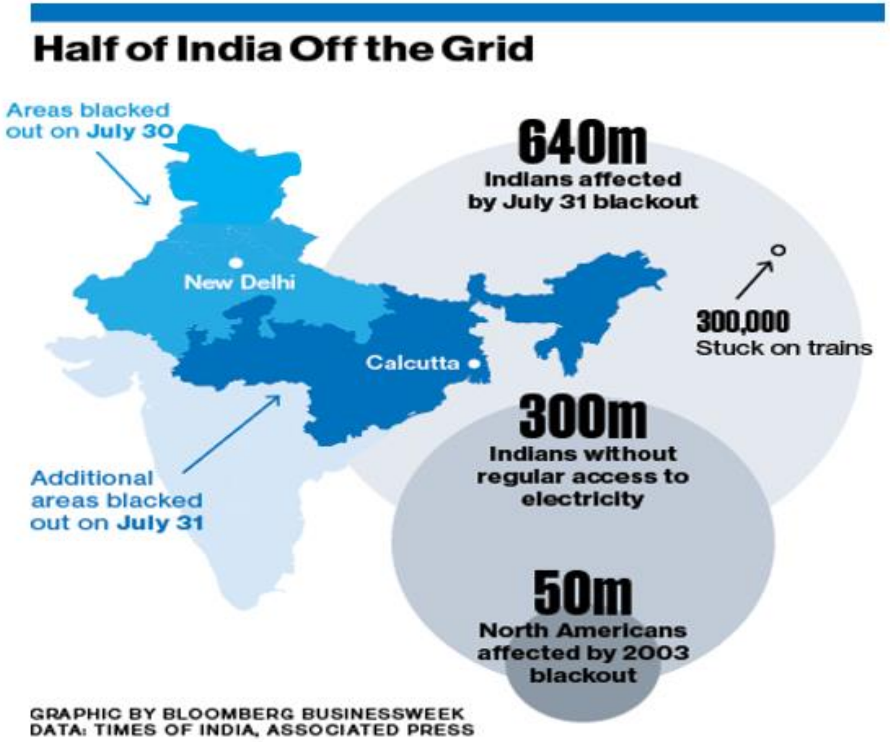
- Load encroachment
- Power swings
- Human errors in relay settings

Load encroachment



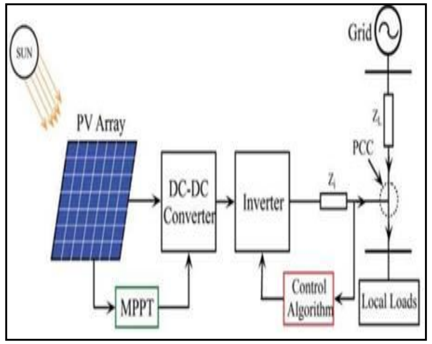
Power swing

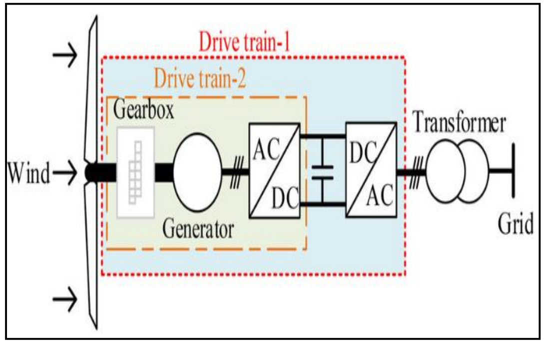




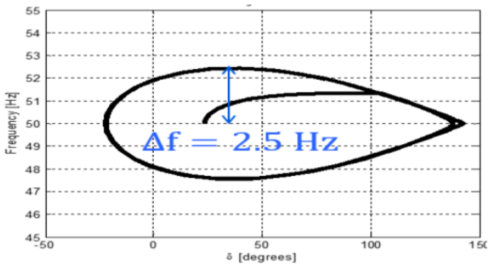
Load encroachment, power swings and human errors such as incorrect protection settings are major causes for blackouts

Can conventional protection safeguard the paradigm shift in power grids?

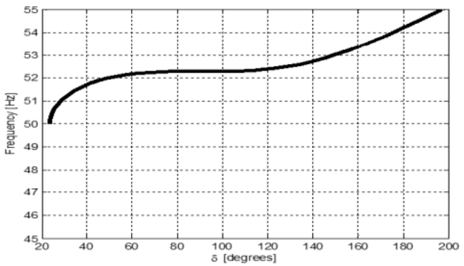




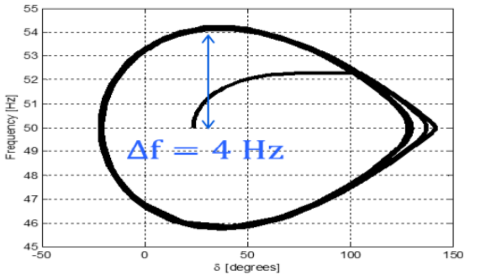
Case 1: H= 6 sec ; CCT =228 ms
System is Stable



Case 2: H= 2 sec ; CCT =228 ms
System is unstable



Case 3: H= 2 sec ; CCT =130ms
System is stable



Reduced inertia (to **33%**) demands much faster fault clearing (by **40%)** to keep system stable

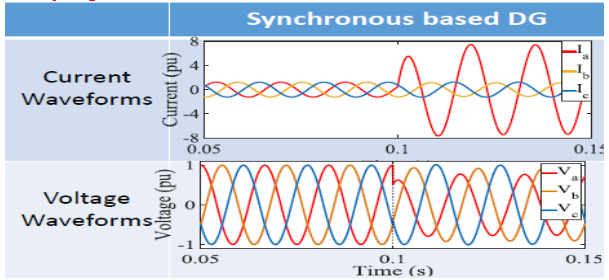
Characteristics of low inertia system:

- Low fault current (1.1-1.3 p.u)
- No zero and negative sequence current
- Less transient stability margin

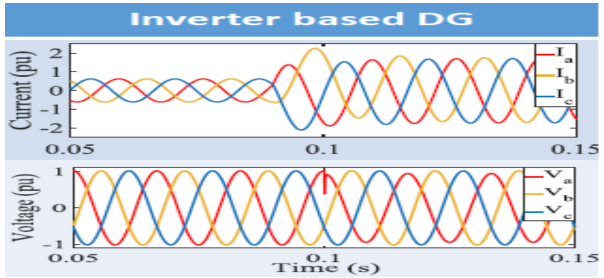
Protection challenges :

- Fault detection
- Fault classification
- Faster fault clearing

Fault response is governed by law of physics

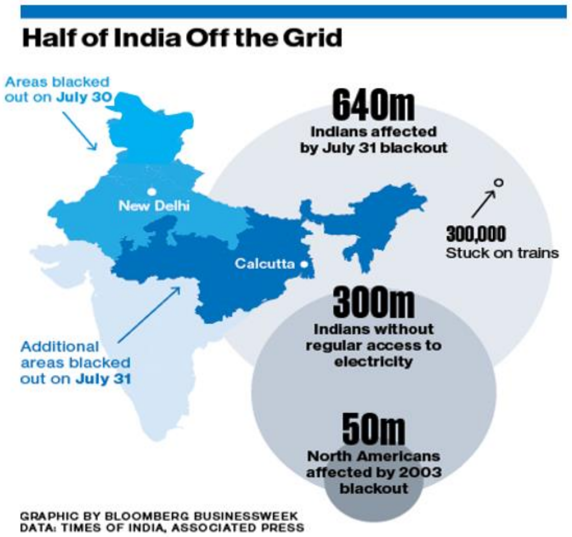


Fault response is governed by inverter /converter control systems

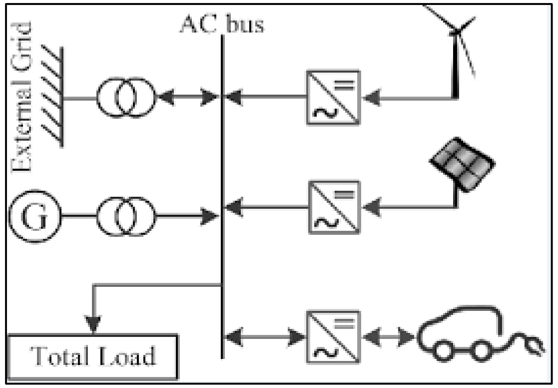


Penetration of renewable sources reduces the system stability margins and necessitates faster fault clearing. Limited fault currents due to inverter based resources (IBRs) pose challenges in fault detection, classification etc.

AI/ML based relaying concept for power grids



2012 Indian blackout



Evolving power grid

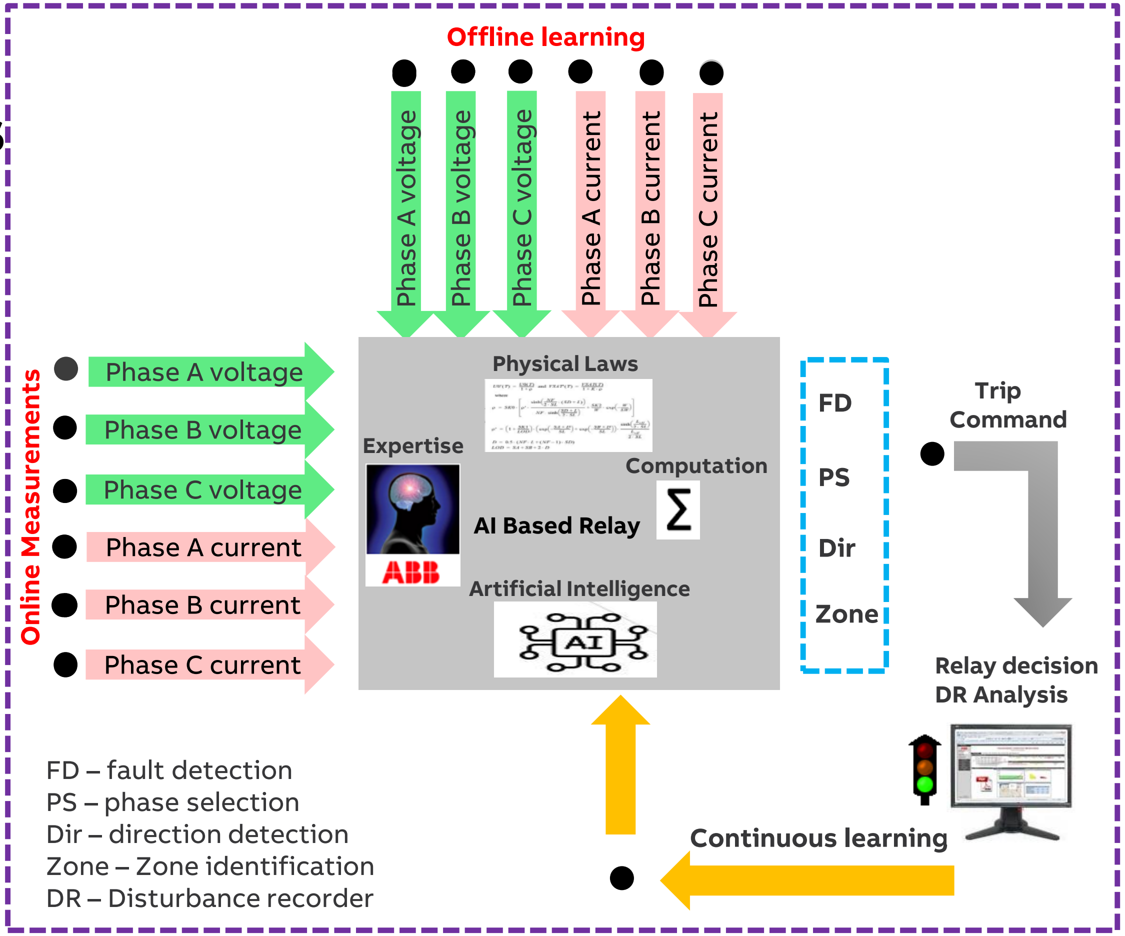
Motivation:

- Incorrect protection settings and human errors are major causes of blackouts
- Renewable integration necessitates faster fault clearing

Objective:

Line protection using **Artificial Intelligence** combined with **domain principles** aimed at

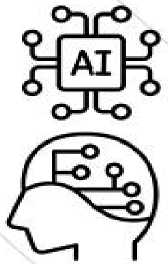
- reducing relay maloperations and improving reliability
- reducing human effort and errors in relay settings
- making relay choose its own settings and be **autonomous**
- improving the relay performance for **conventional and evolving power grid**



Results and conclusions:

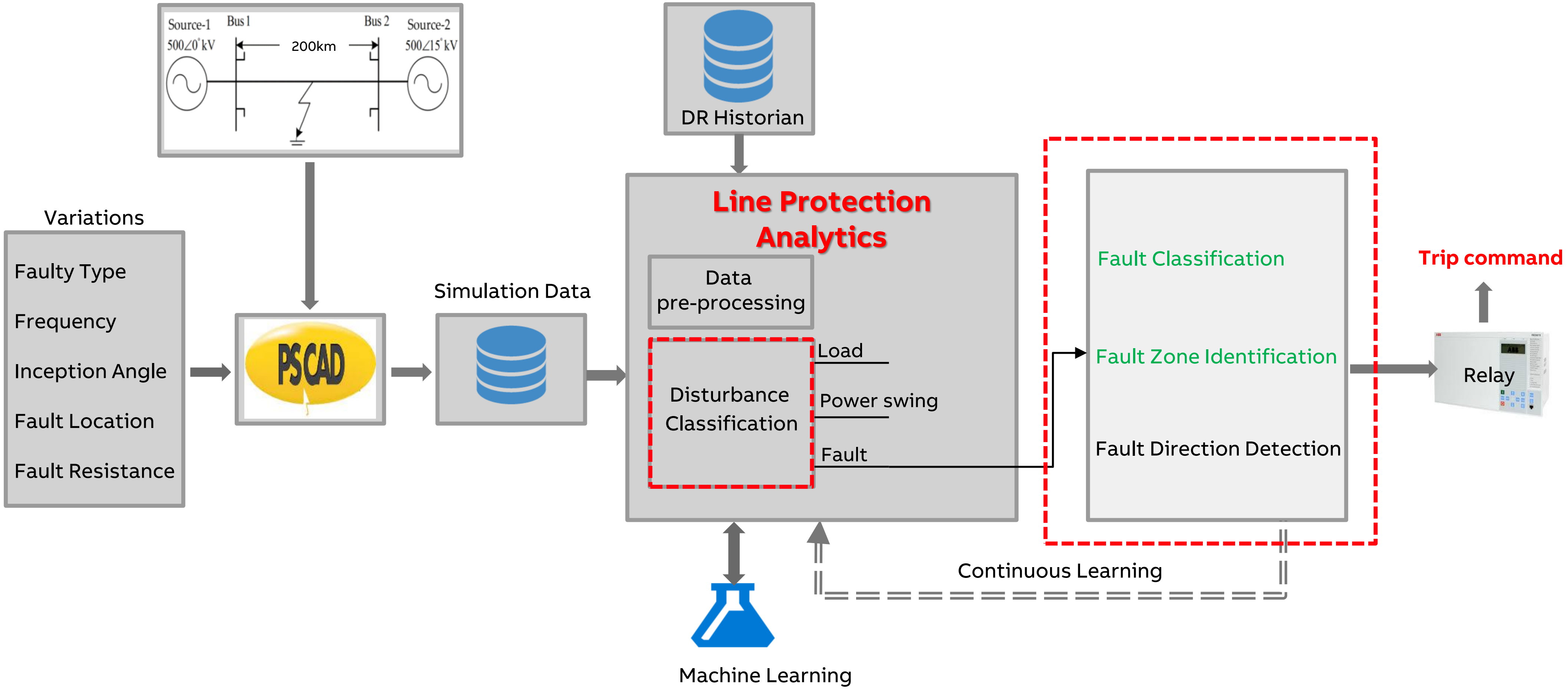
- Improved speed of relay operation by 75%
- Improved reliability to 99.76%

Automation to Autonomous



Source: Business value-Newton-Evans report (2016-18)

AI/ML based protection for transmission grids – Block diagram



AI/ML based disturbance classification

Inputs & Outputs

Input Features

Voltage of each phase

↓

RMS Voltage

↓

Moving Average Filter

↓

Rate of Change

↓

First 3 Samples*

Input:

[ΔVArms, ΔVBrms, ΔVCrms]

Output:

[Fault, Power Swing, Load Change]

Dataset Coverage

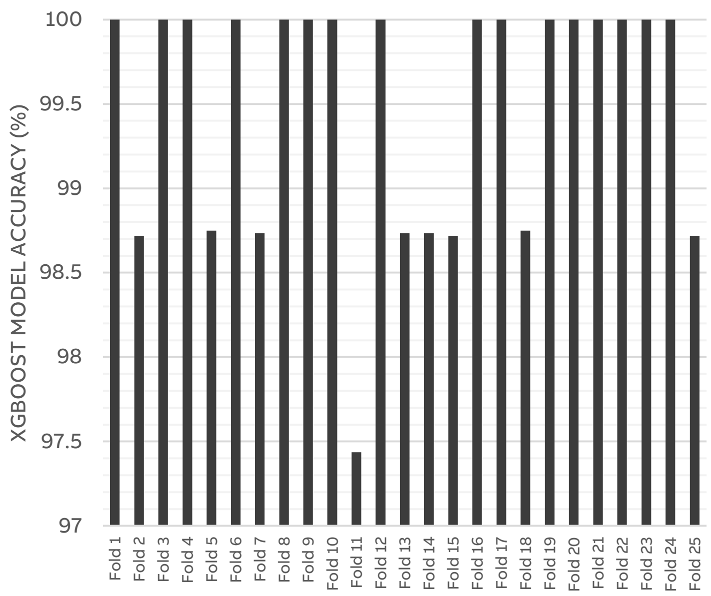
Disturbance type	Parameter	Variations	Quantity
Fault	Fault Type	ABC-g	816
	Swing Frequency (Hz)	0.5 to 10Hz with an increment of 0.1 Hz	
	Inception angle (degree)	0, 90	
	Fault location (km)	5, 10, 20, 40, 50, 60, 80 and 95	
	Fault resistance (Ω)	0.01, 10, 50	
	Source to line impedance ratios (SIR)	(0.1:1), (1:2), (2:0.5) and (5:2)	
Power Swing	Swing Frequency (Hz)	0 to 10 Hz with an interval of 0.1 Hz. Here we considered both low (50-40Hz) and high (50 to 60Hz) frequency	950
Load Change	Power flow angle (deg)	5.5 to 55.5 degree with an interval of 0.5 degrees. And few random load changes using switching on the parallel line etc.	200

*: the rate of change of RMS voltage (ΔV_{rms}) after disturbance detection. The disturbance detection can be done by using a small threshold and it is independent on the power system conditions.

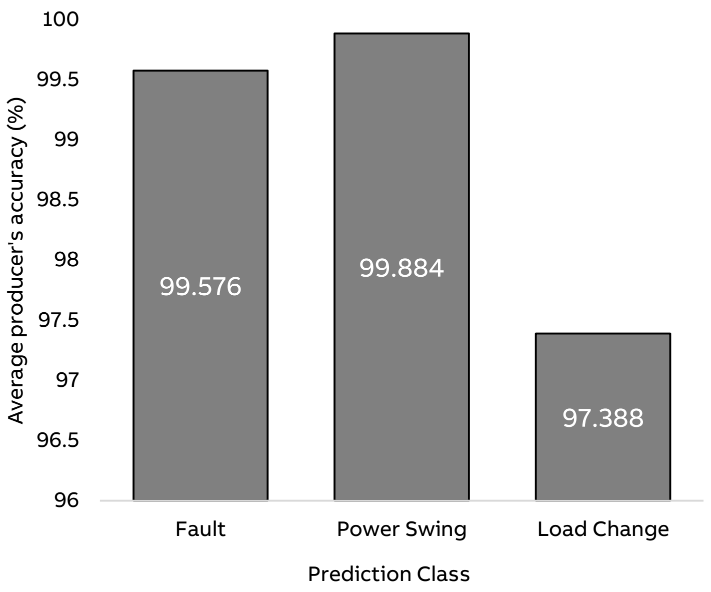
Validation of the concept

Results and conclusions

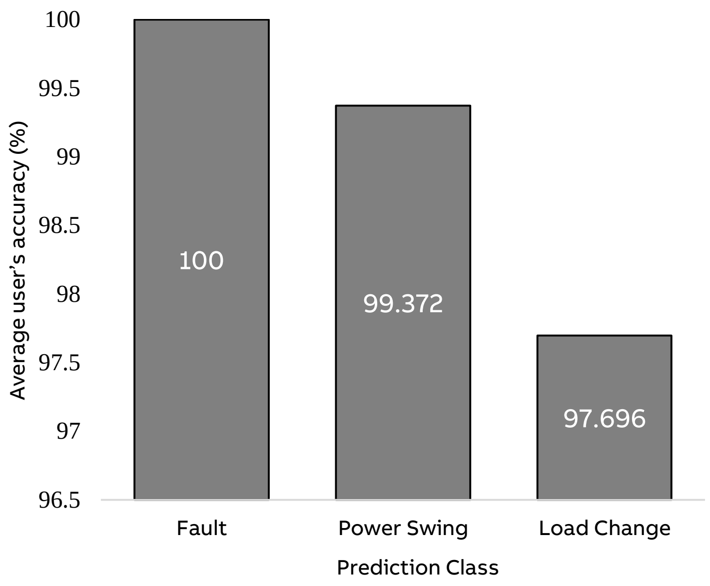
Accuracy



Producer's Accuracy

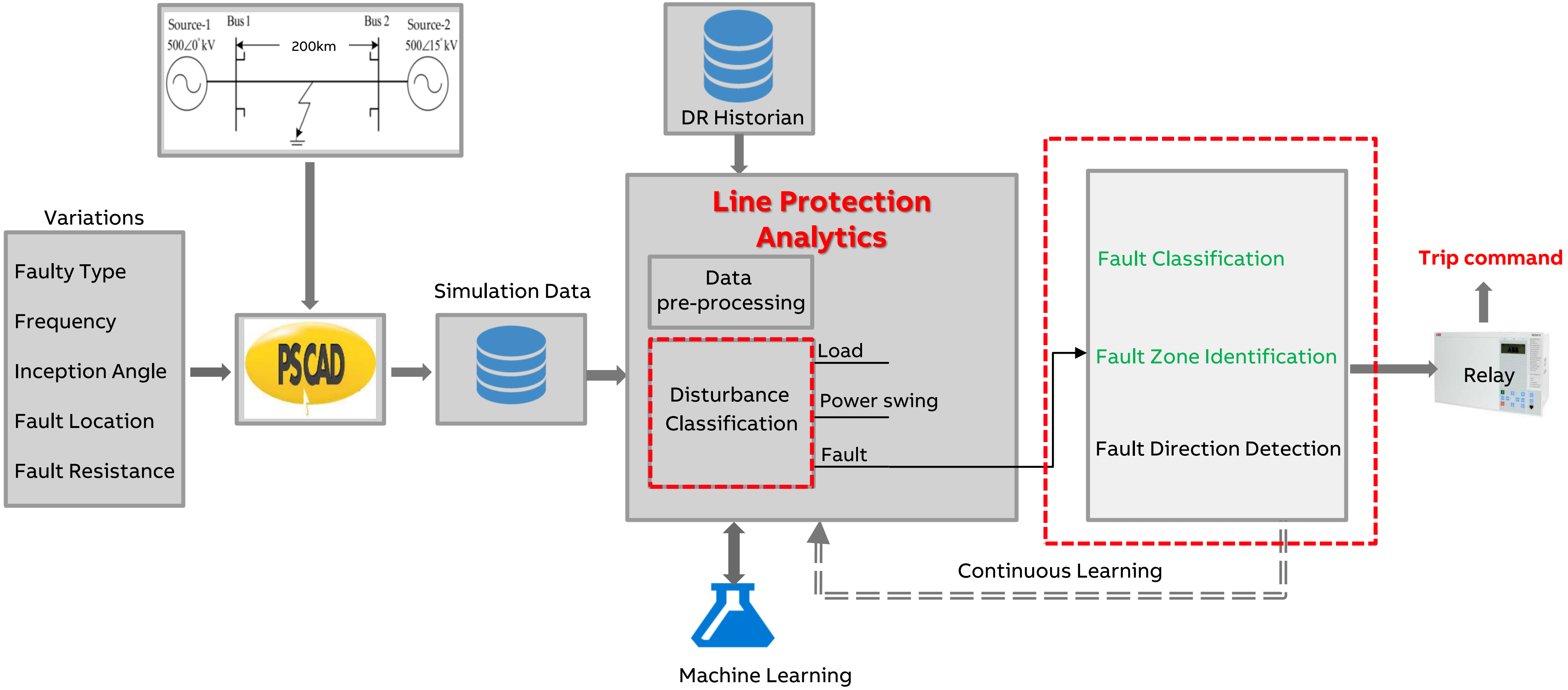


User's Accuracy



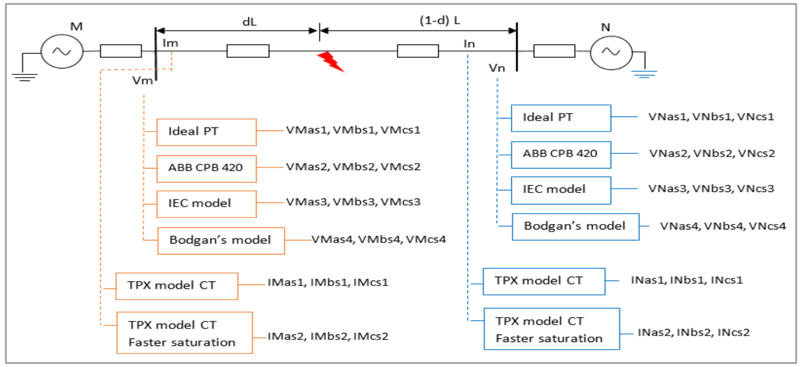
Average prediction accuracy of 99.49% within 4msec after disturbance

AI based Line Protection for transmission grids – Block diagram



AI/ML based distance protection

Data set coverage, Test system, Features



Dataset Coverage

Simulation parameters	Variation
System parameters	400 kV, 200 km
Source to line impedance ration (SIR (M:N))	(0.1:0.5), (0.5:0.5), (1:0.5)
Fault type	A-g, BC, BC-g and ABC
Fault location (%)	2, 20, 40, 64, 76, 84 and 95
Fault resistance (Ω)	0.1, 5 and 20
Fault inception angle (deg)	0, 15, 30, 60, 90, 120, 135, 210 and 270
Load as percentage of full load (%)	0, 40, 80 and 125

Total number of test cases for one load case =2268 and for all load cases =9072

Input Features

3-phase voltages and currents

Calculation of incremental quantities

Calculation of slope of incremental quantities

12 samples after fault detection

Input:

[ΔVa, ΔVb, ΔVc

ΔIa, ΔIb, ΔIc]

AI Trained Model

Output:

[Fault classification,

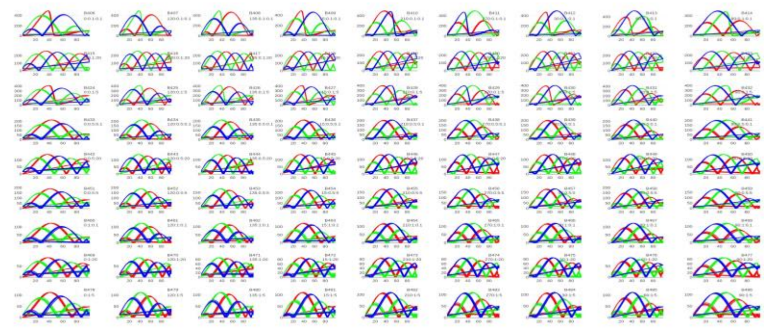
Fault reach identification]

Fault Classification

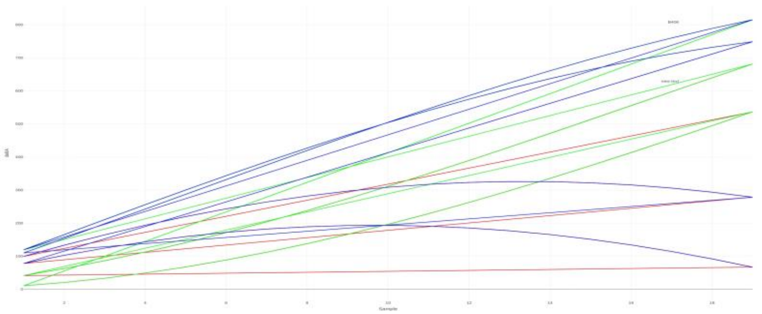
Analytical Workflow

Data → Exploratory Analysis → Feature Selection → Model Building → Model Validation

Data



Exploratory Analysis

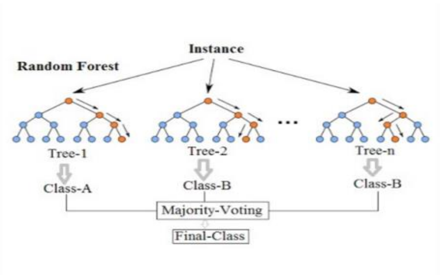


Feature Selection

Selected 4 features

- × Rise Time
- × Peak Value
- ✓ **Slope**
- × Area

Model Building



5 Trees for fault classification

25 Trees for zone identification

Model

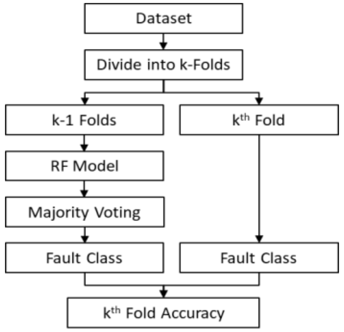
```
training_set = data_slope[!data_slope$Batch %in% Folds[[F]],]
test_set = data_slope[data_slope$Batch %in% Folds[[F]],]
dt_model = rpart(Zone~ slopeVA + slopeVB + slopeVC + slopeIA + slopeIB + slopeIC , training_set)
y_predict = predict(dt_model, test_set[,c("slopeVA", "slopeVB", "slopeVC", "slopeIA", "slopeIB", "slopeIC" )], type='class',
shape = TRUE)
test_set_predict = cbind(test_set, y_predict)
```

```
training_set = data_slope[!data_slope$Batch %in% Folds[[F]],]
test_set = data_slope[data_slope$Batch %in% Folds[[F]],]
rf_model = randomForest(Zone~ slopeVA + slopeVB + slopeVC + slopeIA + slopeIB + slopeIC , training_set, ntree = 25, mtry=3)
y_predict = predict(rf_model, test_set[,c("slopeVA", "slopeVB", "slopeVC", "slopeIA", "slopeIB", "slopeIC" )], type='class',
shape = TRUE)
test_set_predict = cbind(test_set, y_predict)
```

Model Validation

K-Fold Cross Validation

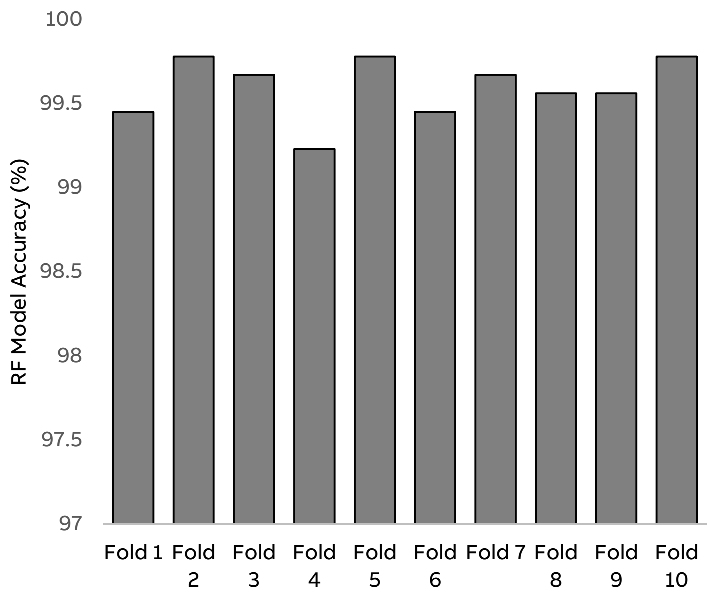
2 out of 3 voting



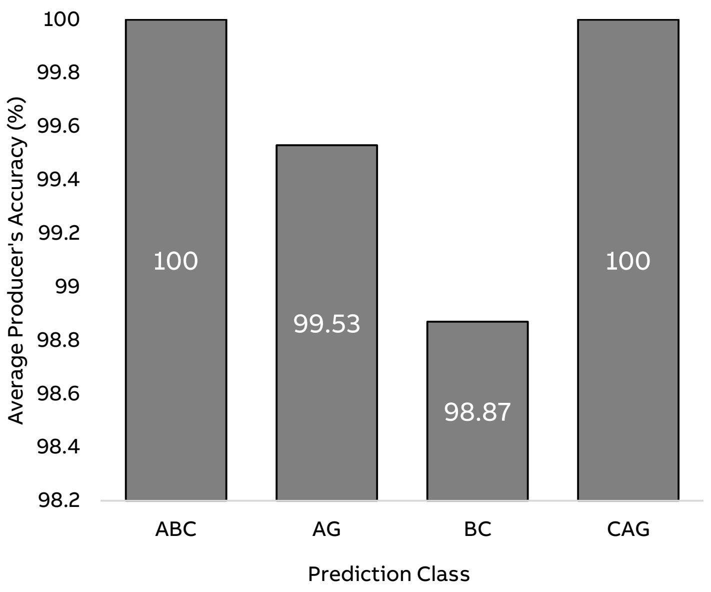
Fault Classification

Random Forest Model Validation

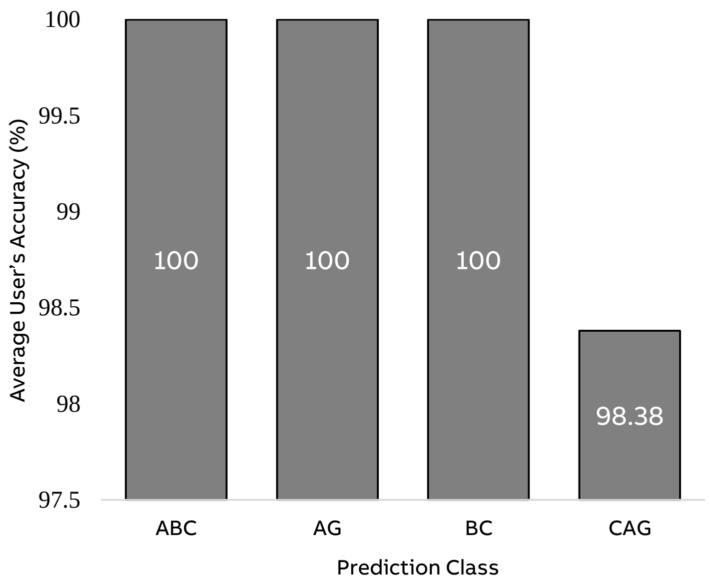
Accuracy



Producer's Accuracy



User's Accuracy

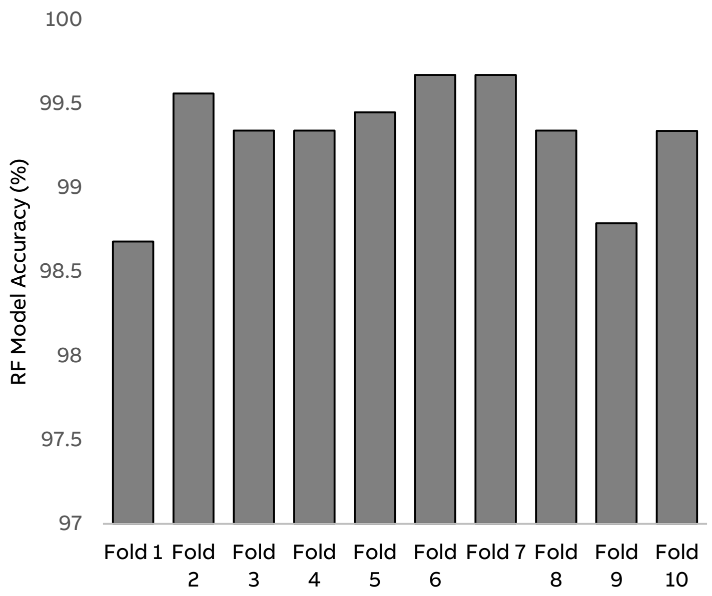


Average Fault Classification accuracy of 99.59% within 3-5ms after disturbance

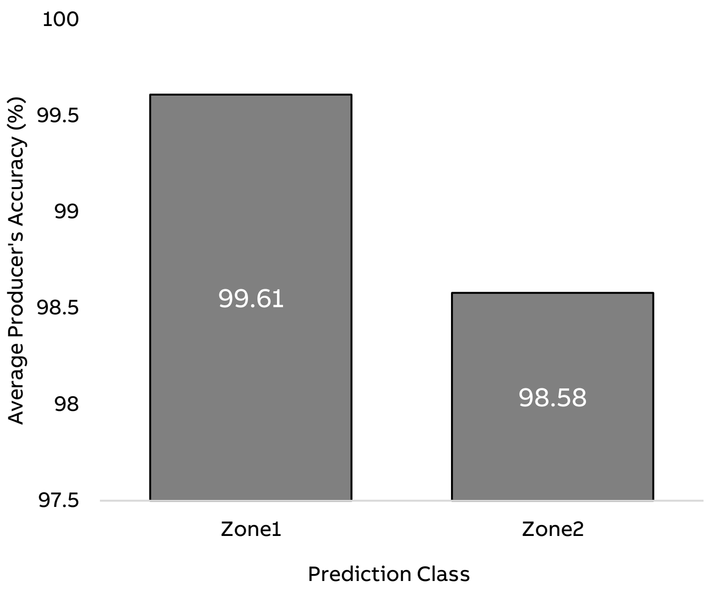
Fault Zone/Reach Identification

Random Forest Model Validation

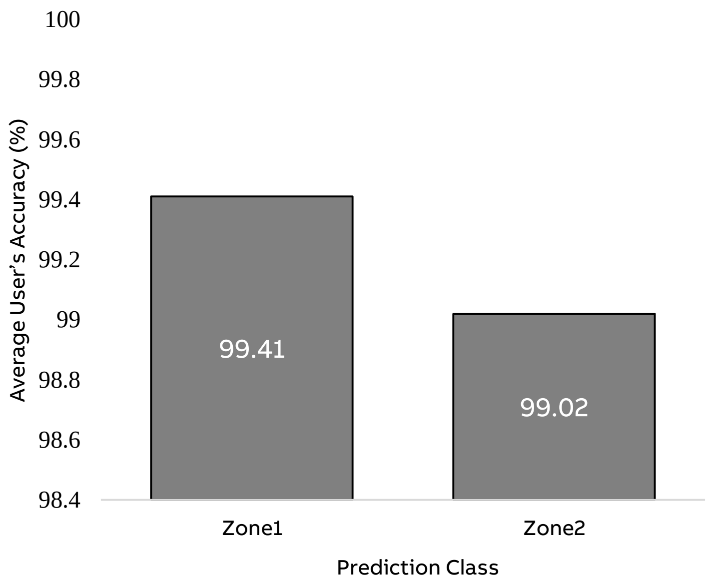
Accuracy



Producer's Accuracy



User's Accuracy



Average Fault Zone identification accuracy of 99.3% within 4ms after disturbance

AI Algorithm – Practical Deployment

Sample Use case

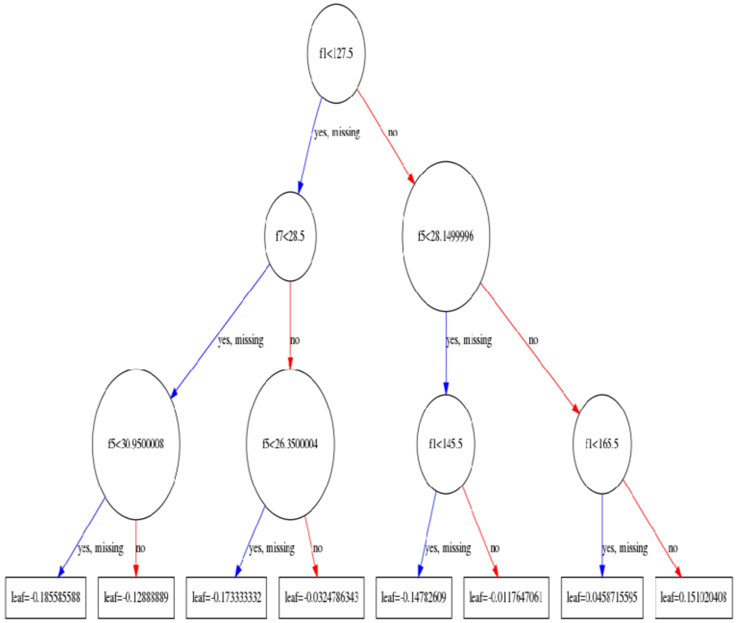
Machine Learning Model

```python
# First XGBoost model for Pima Indians dataset
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import m2cgen as m2c
# load data
dataset = loadtxt('pima-indians-diabetes.csv', delimiter=",")
# split data into X and y
X = dataset[:,0:8]
Y = dataset[:,8]
# split data into train and test sets
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size, random_state=seed)
# fit model no training data
model = XGBClassifier()
model.fit(X_train, y_train)
# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]
# evaluate predictions
accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))


code = m2c.export_to_c(model)

print(code,  file=open('Code.txt', 'w'))
```

Interpretation



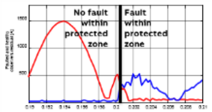
Deployable Code

```c
#include <math.h>
#include <string.h>
void score(double * input, double * output) {
    double var0;
    if ((input[1]) >= (127.5)) {
        if ((input[5]) >= (28.1499996)) {
            if ((input[1]) >= (165.5)) {
                var0 = 0.151020408;
            } else {
                var0 = 0.0458715595;|
            }
        } else {
            if ((input[1]) >= (145.5)) {
                var0 = -0.0117647061;
            } else {
                var0 = -0.14782609;
            }
        }
    } else {
        if ((input[7]) >= (28.5)) {
            if ((input[5]) >= (26.3500004)) {
                var0 = -0.0324786343;
            } else {
                var0 = -0.173333332;
            }
        } else {
            if ((input[5]) >= (30.9500008)) {
                var0 = -0.12888889;
            } else {
                var0 = -0.185585588;
            }
        }
    }
    double var1;
    if ((input[1]) >= (127.5)) {
        if ((input[5]) >= (28.1499996)) {
            if ((input[6]) >= (0.434000015)) {
                var1 = 0.124070883;
            } else {
                var1 = 0.0292990096;
            }
        } else {
            if ((input[1]) >= (145.5)) {
                var1 = -0.010865353;
            } else {
                var1 = -0.136249229;
            }
        }
```

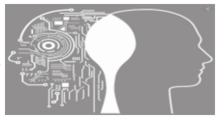
Risk identified: Challenges in practical deployment of AI based solutions
Mitigation: Generated practically deployable deterministic code

Results and Conclusions

Proposed Method



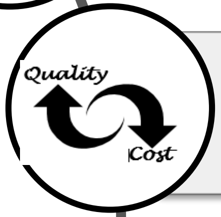
Time domain



ML

Performance index \ Technology	Proposed Method (Time domain + ML)
Dependability	99.3%
Security	98.6%
Operating time	**~2.5ms**
Sampling rate	4.8 kHz
Product cost	Low
Algorithm complexity	Medium
Setting complexity	Low

Outstanding dependability and security - close to 100% - with decision time in range of 2-3 milliseconds !!



Achieve the excellent performance without the additional cost/ hardware burden and reduced engineering costs !!

Boost digital substation business by encouraging deployment of applications at process bus level !!

Self-learning solution - No more manuals !! Less dependent on human domain expertise !!

ABB