Deepwave Digital

making sense of signals

John D. Ferguson, Ph.D.
President / CEO
info@deepwavedigital.com

March 28, 2018

Deep Learning and Radio Frequency (RF) Systems

Deep Learning is Emerging

Cyber

- Intrusion Detection
- Threat classification
- Facial recognition
- Imagery analysis

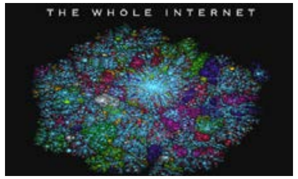
Medicine

- Tumor Detection
- Medical data analysis
- Diagnosis
- Drug discovery

Autonomy

- Pedestrian / obstacle detection
- Navigation
- Street sign reading
- Speech recognition

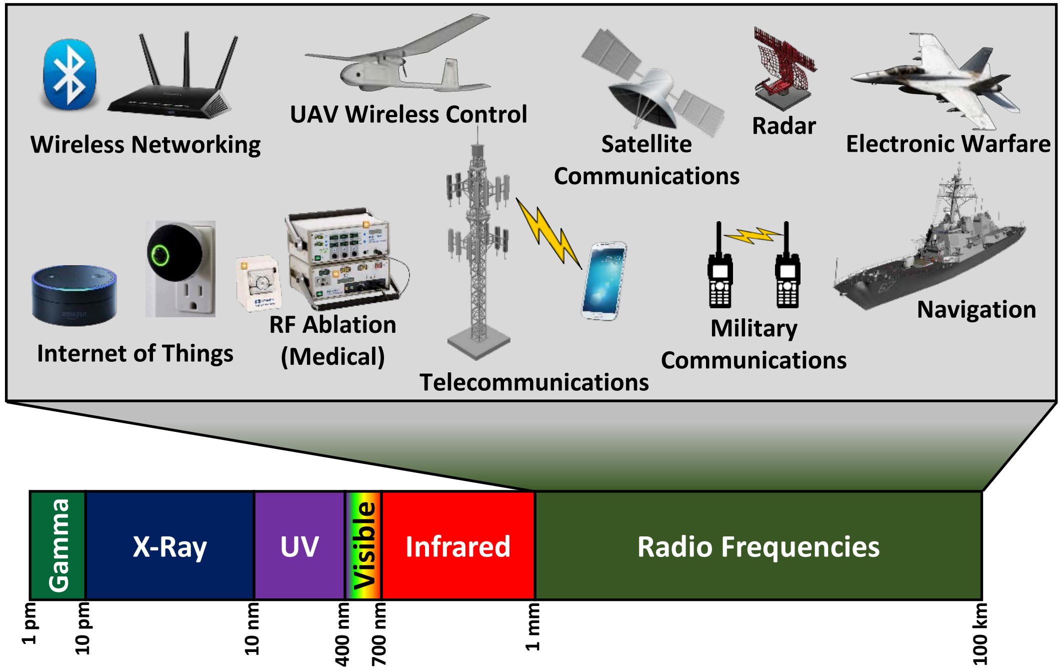
Internet



- Image classification
- Speech recognition
- Language translation
- Document / database searching

Enabled by low-cost, highly capable general purpose graphics processing units (GPUs)

Radio Frequency Technology is Pervasive



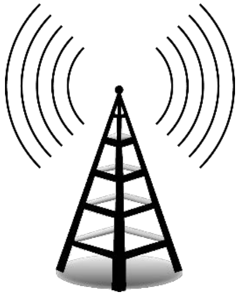
Deep learning technology enabled and accelerated by GPU processors

- Has yet to impact design and applications in wireless and radio frequency systems

Why Use AI in Wireless Technology

Reduce Human Capital

with AI analytics



Increase Platform Reliability

with reduced down-time

Increase Cyber Security

with wireless network monitoring

AI will increase wireless system reliability and security while simultaneously reducing human capital and cost

Why Has It Not Been Addressed

Bandwidth Limitations

remote processing not possible

- AI requires large data sets
- Insufficient bandwidth to send to remote data center

Limited Compute Resources

at field site

- No RF systems exist with integrated AI computational processors

Complicated Software

for RF and AI independently

- Disjointed software
- Difficult to program and understand

Datalink Bandwidth Limitations

- Raw RF signal must be digitized to apply deep learning
  - Nyquist says two samples per Hertz required to reconstruct the signal
  - Each sample is 16 bits (unpacked)

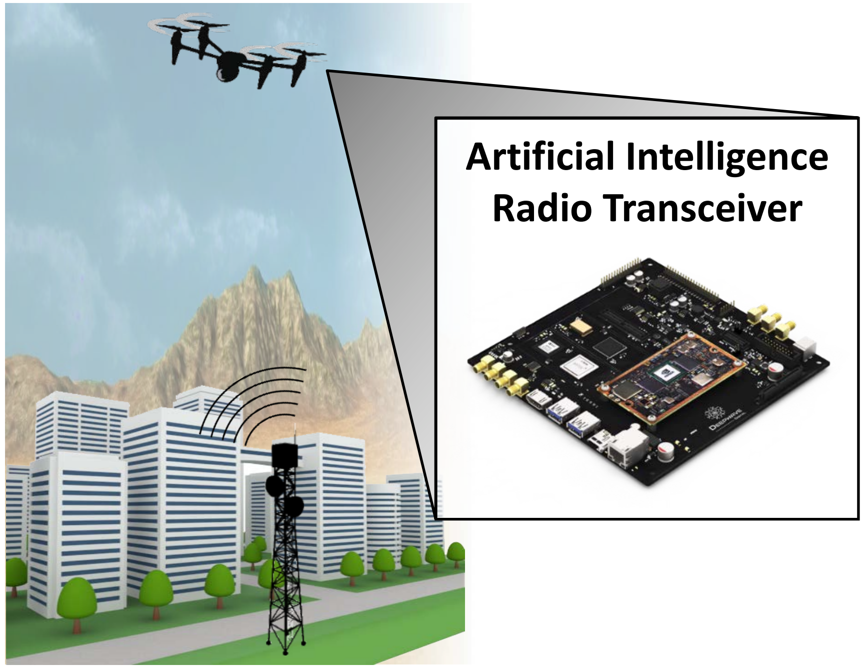
Frequency Band	Bandwidth (MHz)	Data Rate (Mbps)
FM Radio (1 channel)	0.2	6
FM Radio (all channels)	20	640
ISM (915)	26	832
ISM (2.4)	100	3,200
Automotive Radar (64)	500	16,000

- Sending raw digital signal to remote data center is unfeasible
  - Datalinks and fiber optic connections already primary resources for signal's data content
  - Remote locations have limited connectivity
- Reasonable solution is to move the computational engine to the edge

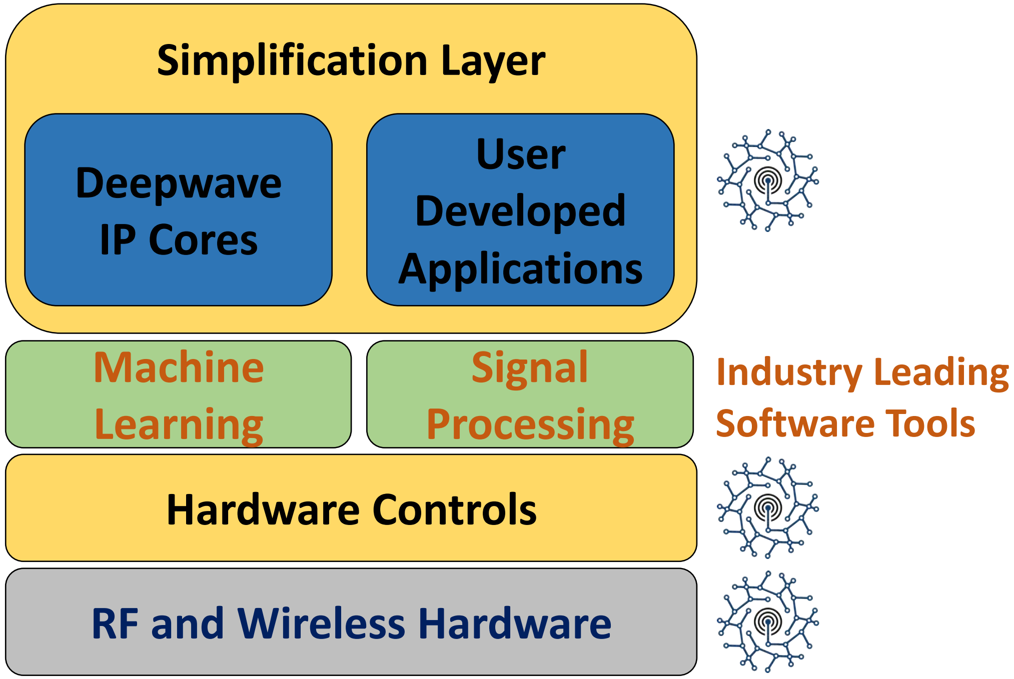
Our Solution and Platform

Approach - Enable the wide adoption of AI within wireless technology with our integrated hardware and software platform

Hardware for Real-world Applications



Easy to Program Software


Simplification Layer

Deepwave IP Cores

User Developed Applications

Machine Learning

Signal Processing

Hardware Controls

RF and Wireless Hardware

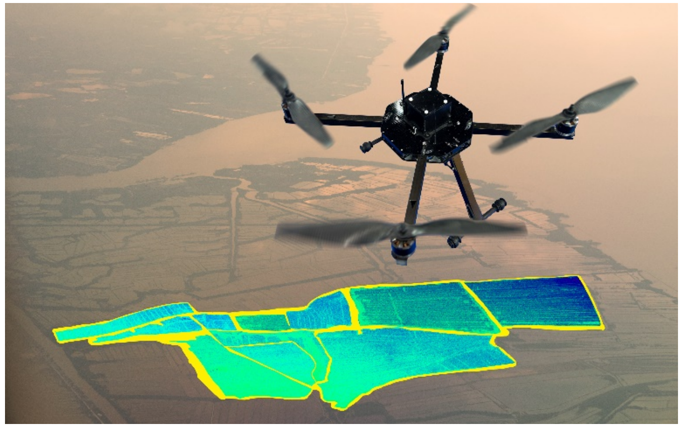
Industry Leading Software Tools

Outline

- Introduction
- ➡ Deep Learning in RF Systems
- Deepwave Digital Technology
- Example Signal Detection and Classification
- Training and Inference of Classifier
- Deploying a Deep Learning RF Systems

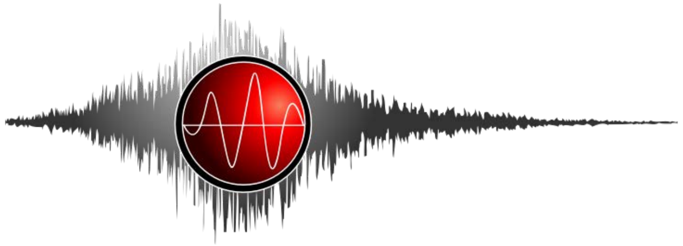
Deep Learning Comparison

Image and Video



- Multiple channels (RGB)
- x, y spatial dependence
- Temporal dependence (video)

Audio and Language



- Single channel
- Frequency, phase, amplitude
- Temporal dependence

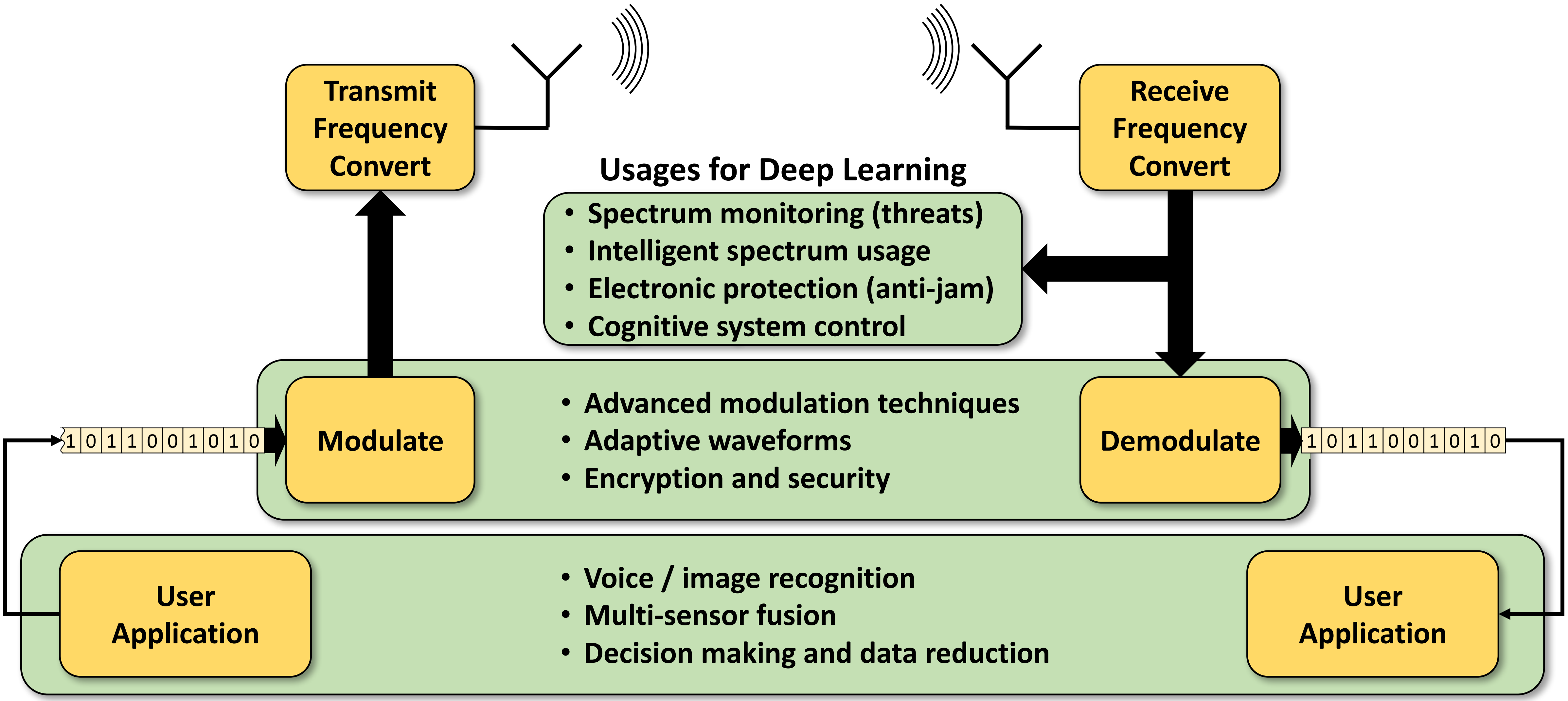
Systems and Signals

- Multiple channels
- Frequency, phase, amplitude
- Temporal dependence
- Human engineered
- Complex data (I/Q)

Existing deep learning potentially adaptable to systems and signals

- **Must contend with complex data types**

Intelligent Radio Frequency (IRF) Systems



Usages for Deep Learning

- Spectrum monitoring (threats)
- Intelligent spectrum usage
- Electronic protection (anti-jam)
- Cognitive system control

- Advanced modulation techniques
- Adaptive waveforms
- Encryption and security

- Voice / image recognition
- Multi-sensor fusion
- Decision making and data reduction

Deep Learning Processors for RF Systems: Training

	Pros	Cons
CPU	• Supported by frameworks	• Slower than GPU • Fewer software architectures
GPU	• Most utilized • Highly parallel and adaptable • Good throughput vs. power req.	• Overall power consumption
FPGA	Not widely utilized, not well suited (yet)	
ASIC	Not widely utilized or well suited	

Deep Learning Processors for RF Systems: Inference

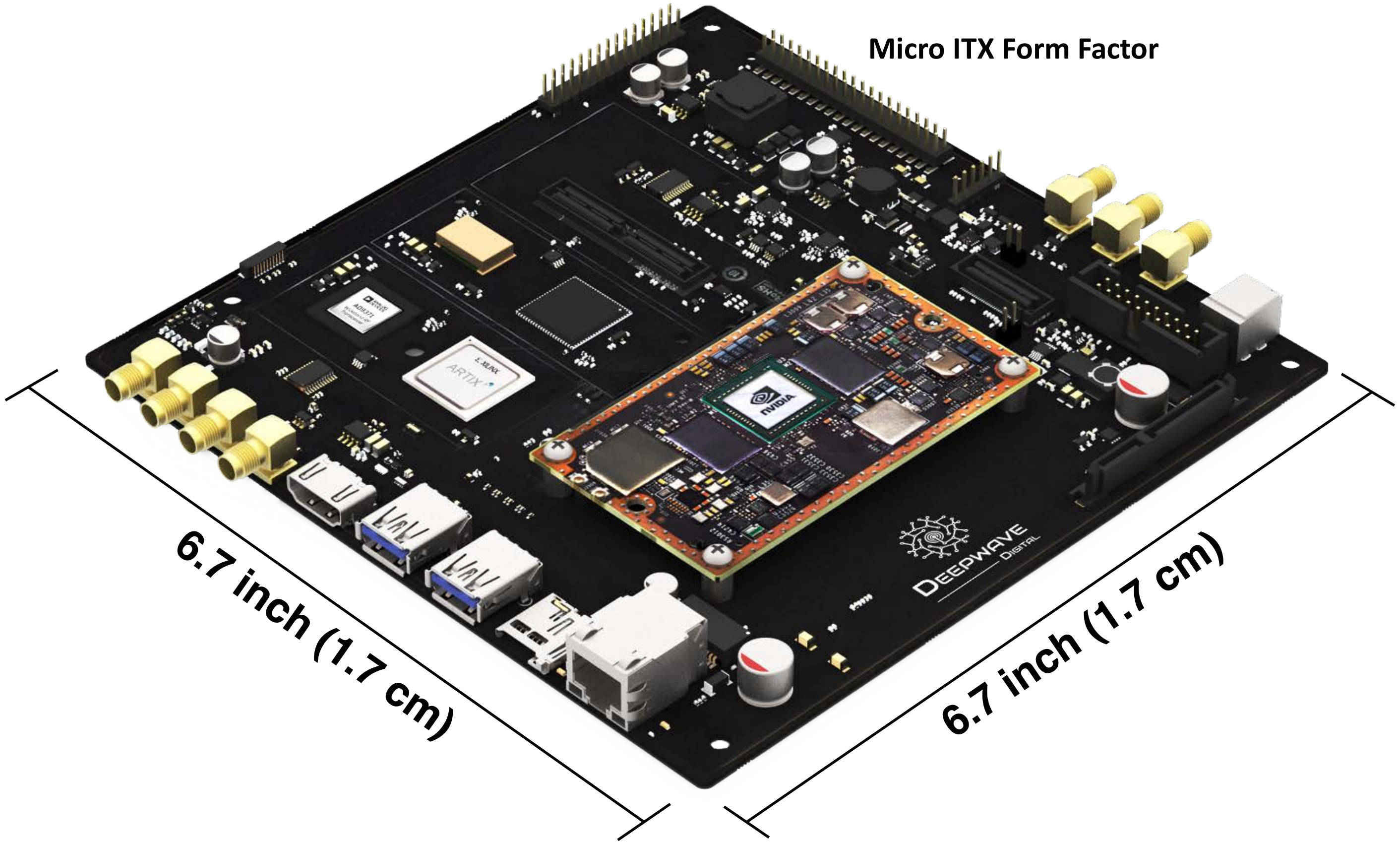
	Pros	Cons
CPU	• Adaptable architecture • Software programmable • Widely utilized in software defined radios	• Low parallelism • Medium power requirements
GPU	• Adaptable architecture • High throughput • Software programmable	• Medium to high power requirements • Not well integrated into RF systems
FPGA	• Power efficient • Somewhat reprogrammable	• Long development time • Specialty expertise required
ASIC	• Extremely power efficient • Highly reliable	• Not adaptable • Long development time • Specialty expertise required

Outline

- Introduction
- Deep Learning in RF Systems
- ➡ Deepwave Digital Technology
- Example Signal Detection and Classification
- Training and Inference of Classifier
- Deploying a Deep Learning RF Systems

Artificial Intelligence Radio Transceiver (AIR-T)

AIR-T

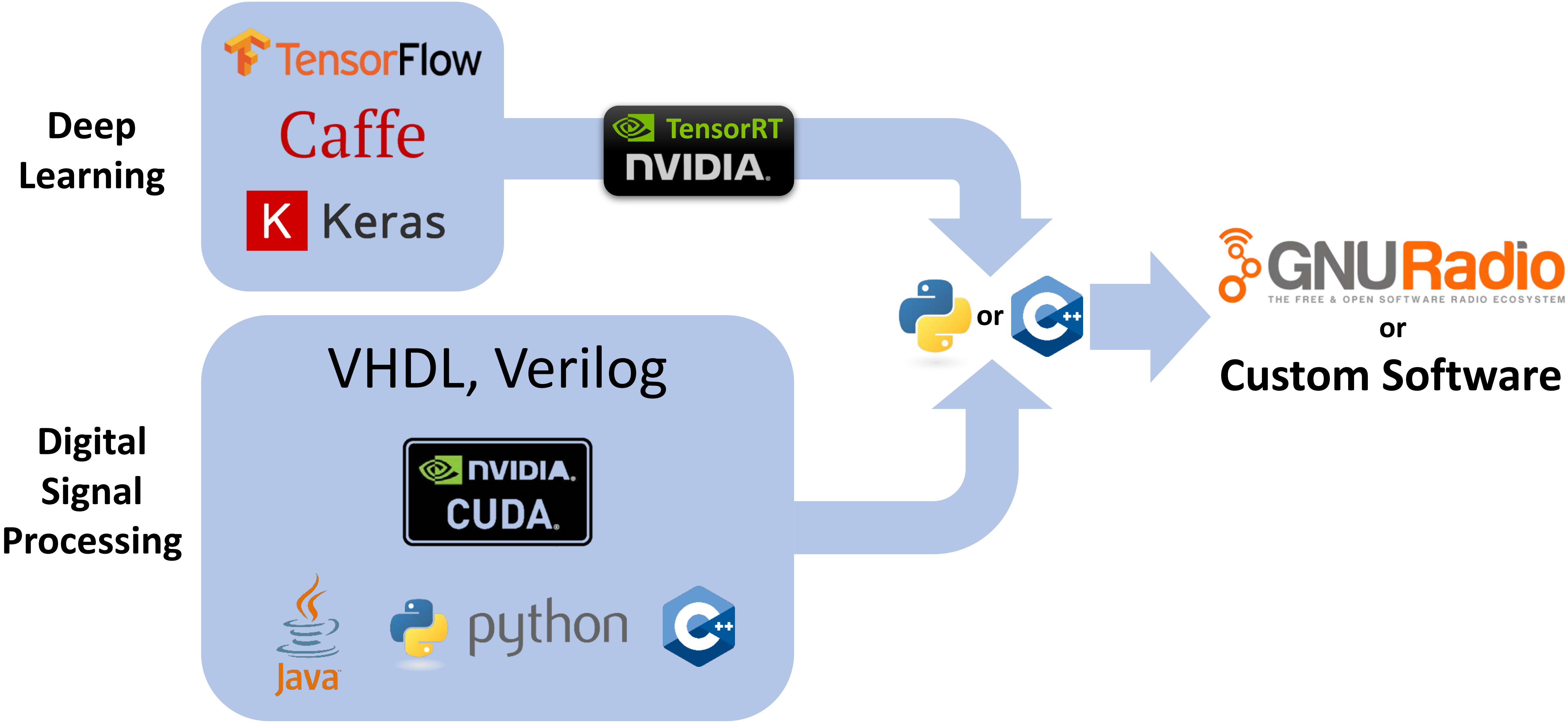


Specifications

- **Dual Channel Transceiver**
  - 300 MHz to 6 GHz
  - 100 MHz bandwidth per Rx channel
  - 250 MHz bandwidth per Tx channel
- **Digital Signal / Deep Learning Processors**
  - Xilinx Artix 7 FPGA
  - ARM Cortex-A57 (quad-core)
  - Denver2 (dual core)
  - Nvidia Pascal 256 Core GPU
  - Shared CPU/CPU memory
- **Connectivity**
  - 1 PPS / 10 MHz for GPS Synchronization
  - HDMI, USB 2.0/3.0, SATA
  - Ethernet, WiFi, Bluetooth
- **Dual Power Mode (22 / 14W)**

Artistic rendering. Exact design may slightly differ

Simplified Programming



Deep Learning

TensorFlow

Caffe

Keras

TensorRT NVIDIA

Digital Signal Processing

VHDL, Verilog

NVIDIA CUDA

Java

python

or

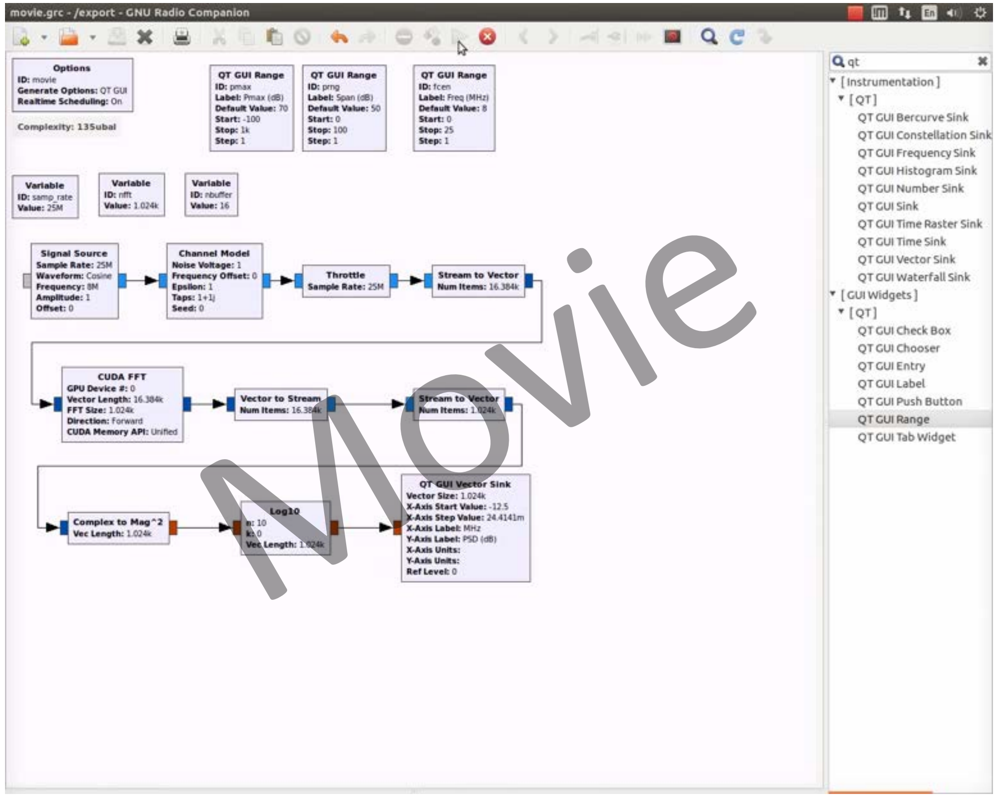
GNURadio THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

or

Custom Software

GNU Radio – Software Defined Radio (SDR) Framework

- **Popular open source SDR toolkit:**
  - RF Hardware optional
  - Can run full software simulations
- **Python API**
  - C++ under the hood
- **Easily create DSP algorithms**
  - Custom user blocks
- **Primarily uses CPU**
  - Advanced parallel instructions
  - Recent development: RFNoC for FPGA processing
- **Deepwave working on integrating GPU support for both DSP and ML**



movie.grc - /export - GNU Radio Companion

Options
ID: movie
Generate Options: QT GUI
Realtime Scheduling: On

Complexity: 135ubal

QT GUI Range
ID: pmax
Label: Pmax (dB)
Default Value: 70
Start: -100
Stop: 1k
Step: 1

QT GUI Range
ID: prng
Label: Span (dB)
Default Value: 50
Start: 0
Stop: 100
Step: 1

QT GUI Range
ID: fcen
Label: Freq (MHz)
Default Value: 8
Start: 0
Stop: 25
Step: 1

Variable
ID: samp_rate
Value: 25M

Variable
ID: nfft
Value: 1.024k

Variable
ID: nbuffer
Value: 16

Signal Source
Sample Rate: 25M
Waveform: Cosine
Frequency: 8M
Amplitude: 1
Offset: 0

Channel Model
Noise Voltage: 1
Frequency Offset: 0
Epsilon: 1
Taps: 1+1j
Seed: 0

Throttle
Sample Rate: 25M

Stream to Vector
Num Items: 16.384k

CUDA FFT
GPU Device #: 0
Vector Length: 16.384k
FFT Size: 1.024k
Direction: Forward
CUDA Memory API: Unified

Vector to Stream
Num Items: 16.384k

Stream to Vector
Num Items: 1.024k

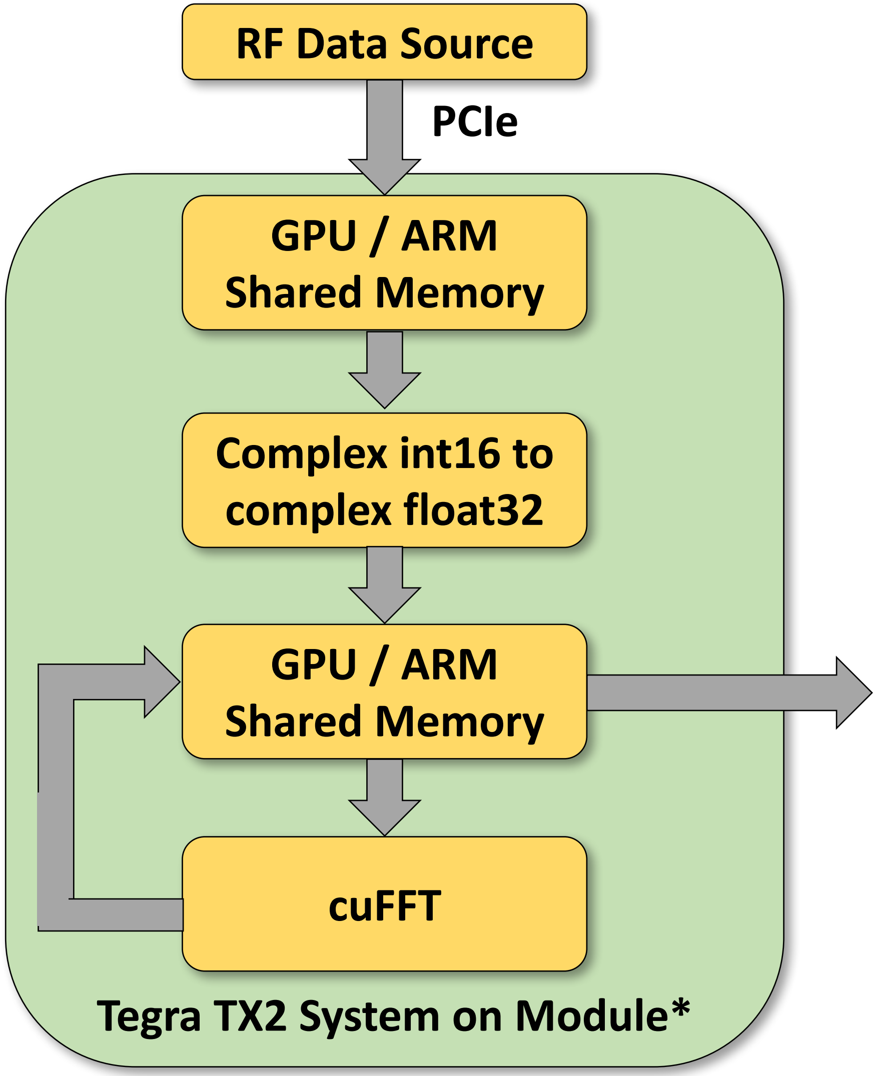
Complex to Mag^2
Vec Length: 1.024k

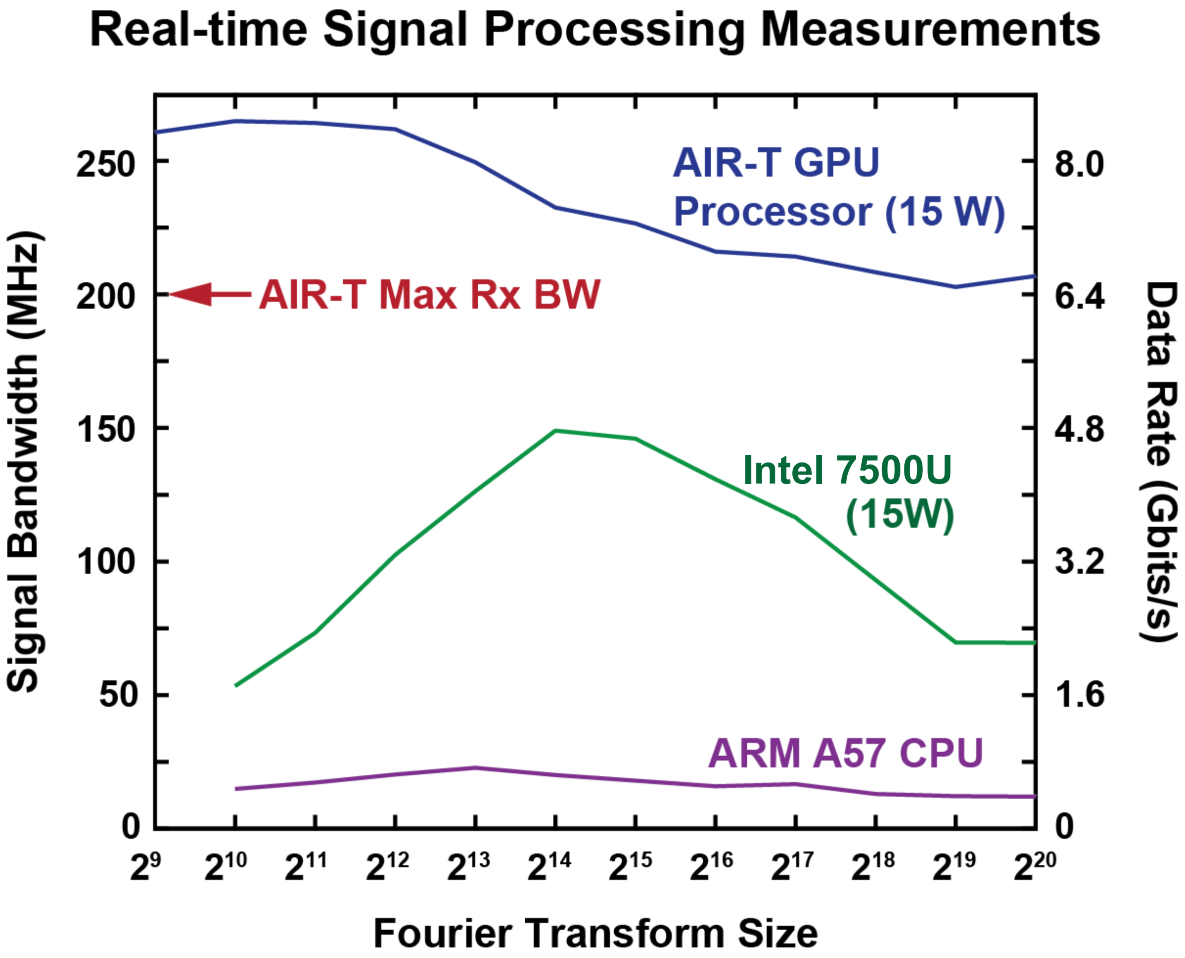
Log10
n: 10
k: 0
Vec Length: 1.024k

QT GUI Vector Sink
Vector Size: 1.024k
X-Axis Start Value: -12.5
X-Axis Step Value: 24.4141m
X-Axis Label: MHz
Y-Axis Label: PSD (dB)
X-Axis Units:
Y-Axis Units:
Ref Level: 0

GNURadio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

AIR-T Initial Performance Testing





* GNU Radio excluded from cuFFT benchmark

Current GNU Radio Limitations for GPU Processing

- **GNU Radio handles memory management**
  - Cannot currently tell it about existing memory buffers
  - Must allocate special CUDA memory (even on Jetson TX2)
  - Requires memcpy() each time GNU Radio block operates
    - Extra copy each direction

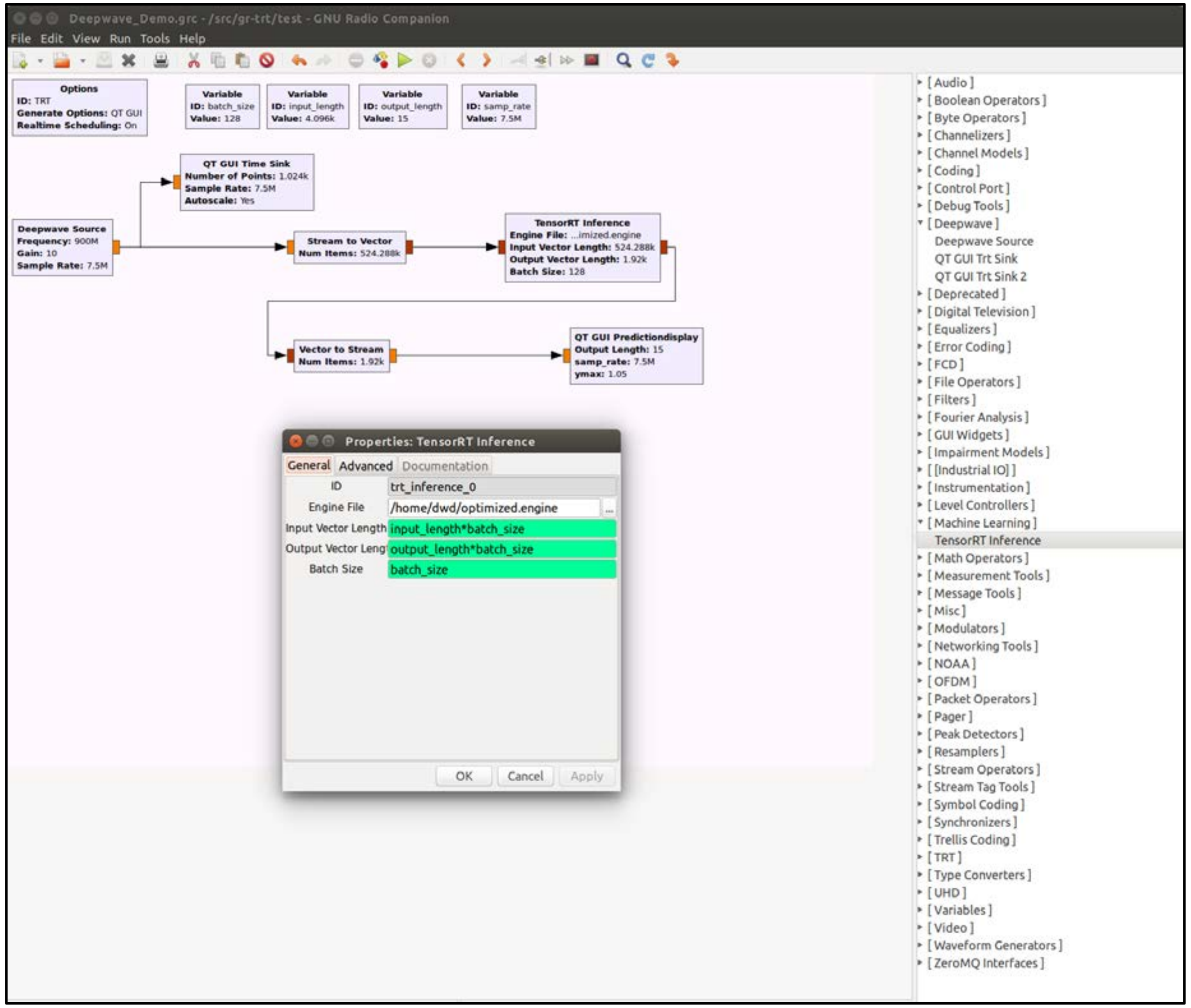
- **GNU Radio has open feature request to support custom buffer allocators***
  - Deepwave willing to collaborate with GNU Radio open source project to advance issue

* Issue #950 on Github

Inference using GNU Radio and TensorRT

- TensorRT chosen as initial inference library for AIR-T
  - Optimized inference for Nvidia based hardware
  - Significant speedups over TensorFlow on Jetson TX2 for image processing
  - Native support for:
    - TensorFlow, Caffe, other frameworks

- Deepwave working on GNU Radio OOT Module (gr-trt) to execute TensorRT inference within GNU Radio



Deepwave_Demo.grc - /src/gr-trt/test - GNU Radio Companion

Options
ID: TRT
Generate Options: QT GUI
Realtime Scheduling: On

Variable
ID: batch_size
Value: 128

Variable
ID: input_length
Value: 4.096k

Variable
ID: output_length
Value: 15

Variable
ID: samp_rate
Value: 7.5M

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 7.5M
Autoscale: Yes

Deepwave Source
Frequency: 900M
Gain: 10
Sample Rate: 7.5M

Stream to Vector
Num Items: 524.288k

TensorRT Inference
Engine File: ...imized.engine
Input Vector Length: 524.288k
Output Vector Length: 1.92k
Batch Size: 128

Vector to Stream
Num Items: 1.92k

QT GUI Predictiondisplay
Output Length: 15
samp_rate: 7.5M
ymax: 1.05

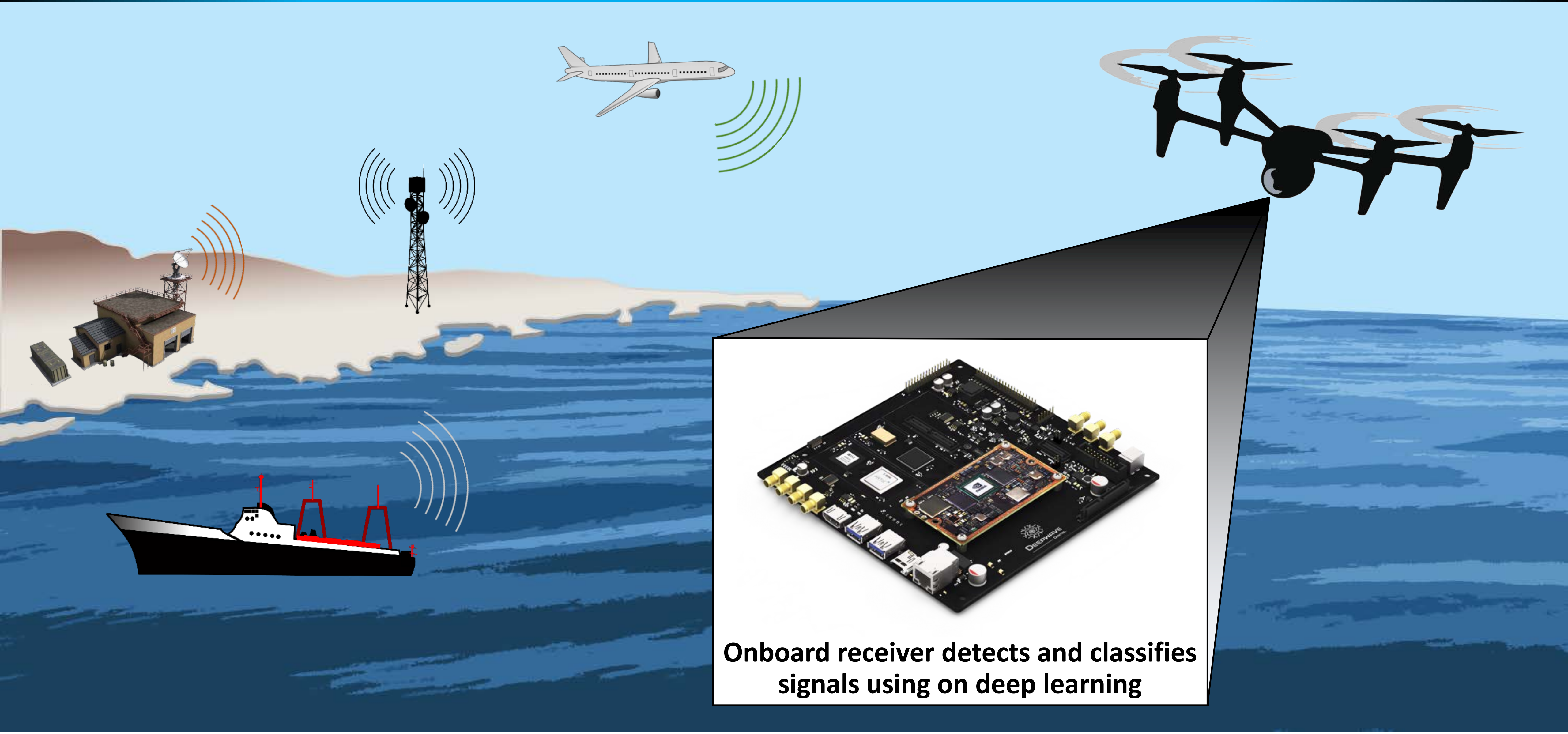
Properties: TensorRT Inference

ID: trt_inference_0
Engine File: /home/dwd/optimized.engine
Input Vector Length: input_length*batch_size
Output Vector Leng: output_length*batch_size
Batch Size: batch_size

Outline

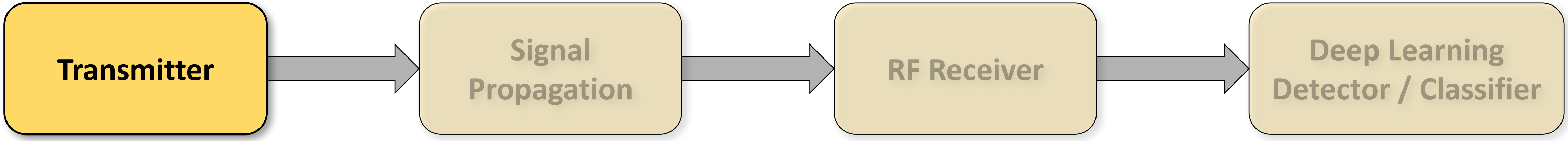
- Introduction
- Deep Learning in RF Systems
- Deepwave Digital Technology
- ➡ Example Signal Detection and Classification
- Training and Inference of Classifier
- Deploying a Deep Learning RF Systems

Multi-transmitter Environmental Scenario



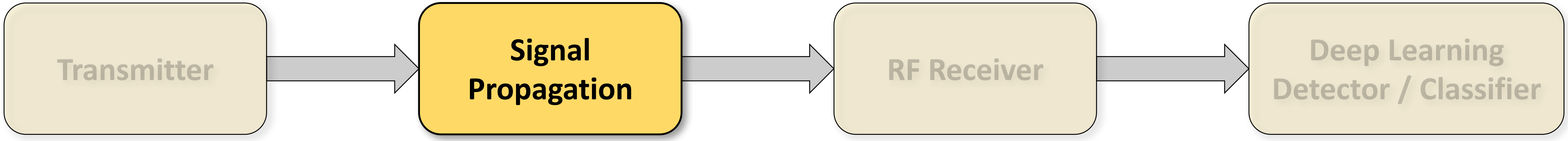
Onboard receiver detects and classifies signals using on deep learning

Radar Signal Detector Model: Transmitted Signals



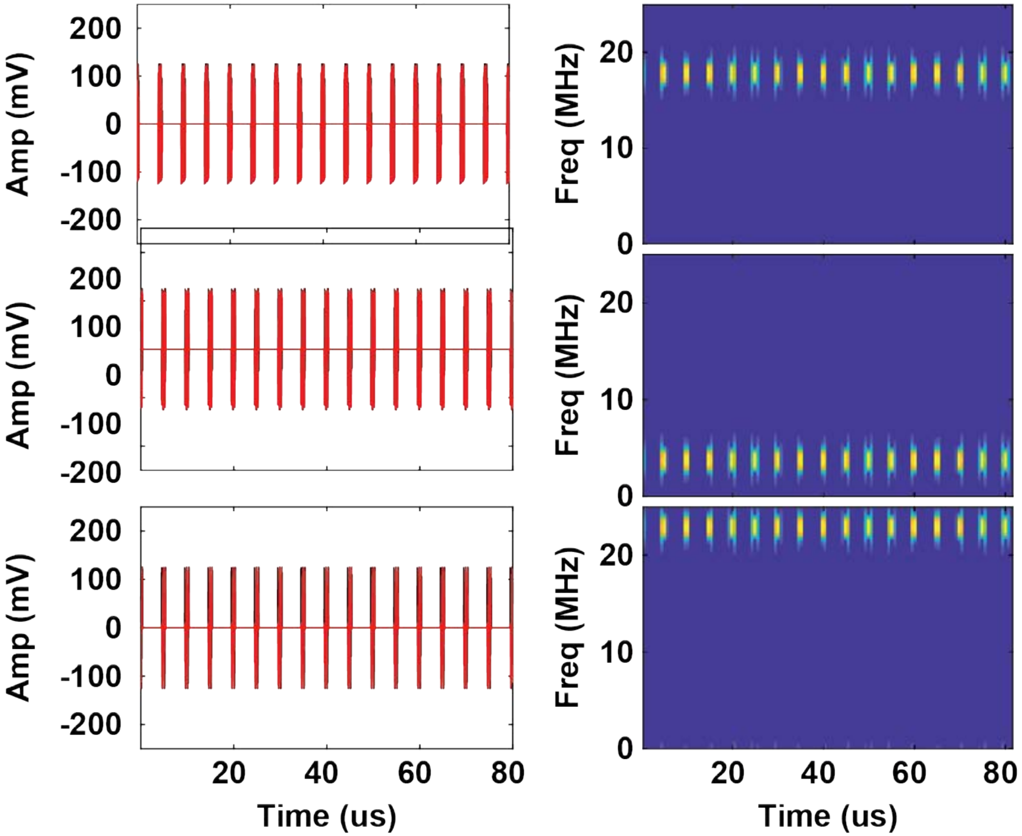
Radar Waveform	Nothing	Interference	Surveillance	Ground (LFM1)	Ground (LFM2)	MTI	Airborne (Med PRF)	Airborne (High PRF)	Ground (Frank Code)	Nautical (Short Range)	Nautical (Long Range)	Nautical (Long Range)	Ground (NLFM1)	Ground (NLFM2)	Ground (NLFM3)
Linear Pulse			X	X	X					X	X	X			
Non-Linear Pulse													X	X	X
Phase Coded Pulse									X						
Pulsed Doppler						X	X	X							

Radar Signal Detector Model: Propagation

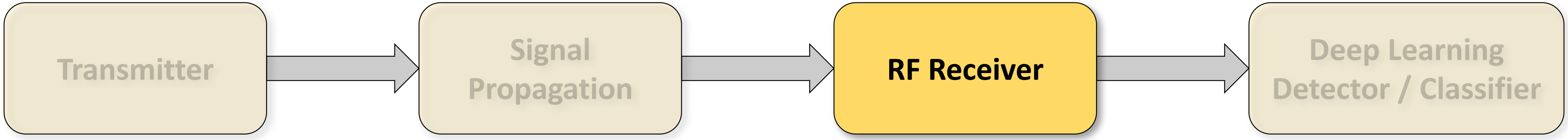


Data Set Generation:

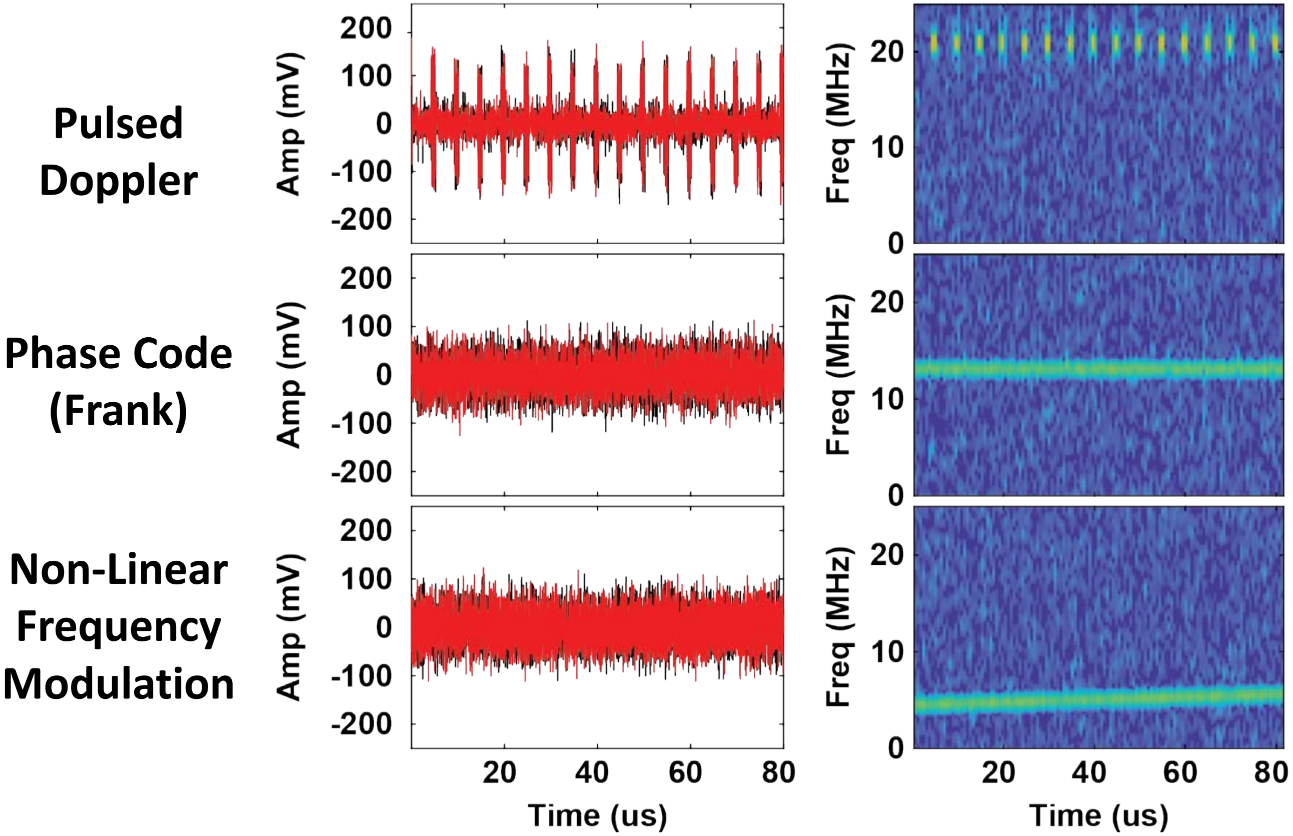
- Random phase shift applied to each signal segment
- Signal frequency changes between coherent processing intervals
- Transmitter range modeled as received signal to noise radios



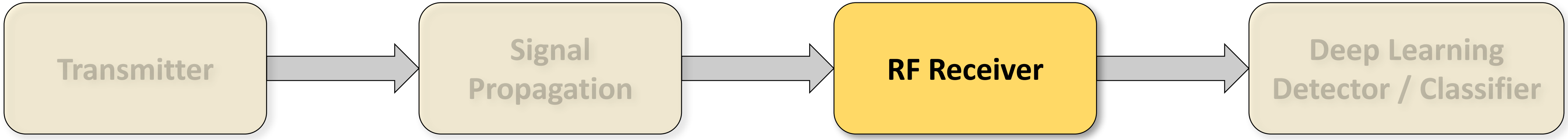
Radar Signal Detector Model: Received Signal

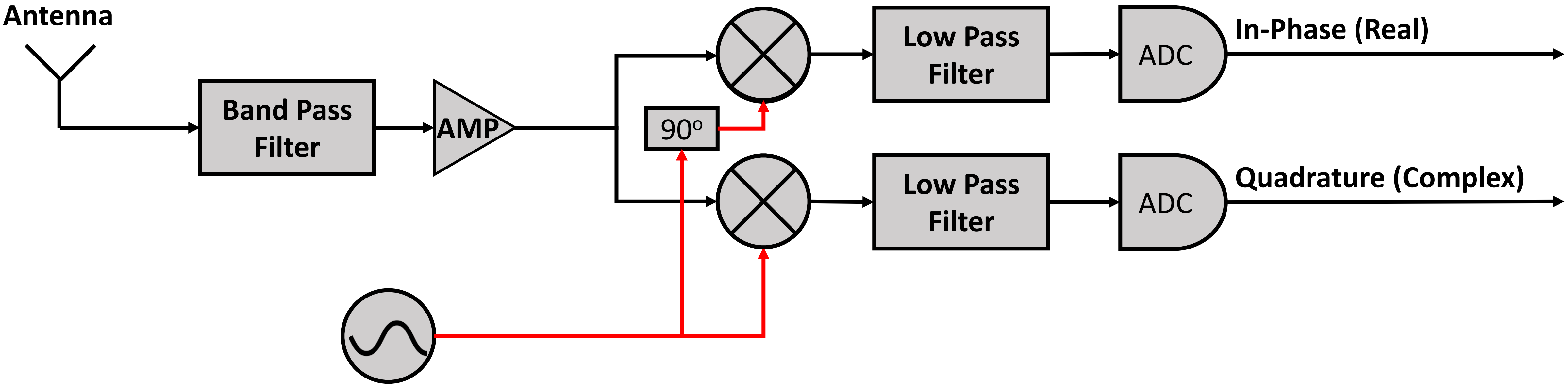


- RF Receiver system:
  - 65 dB gain
  - 5 dB noise figure
  - 14 ADC bits
  - 25 MHz bandwidth
- Hilbert Transformer receiver
  - Complex sample data



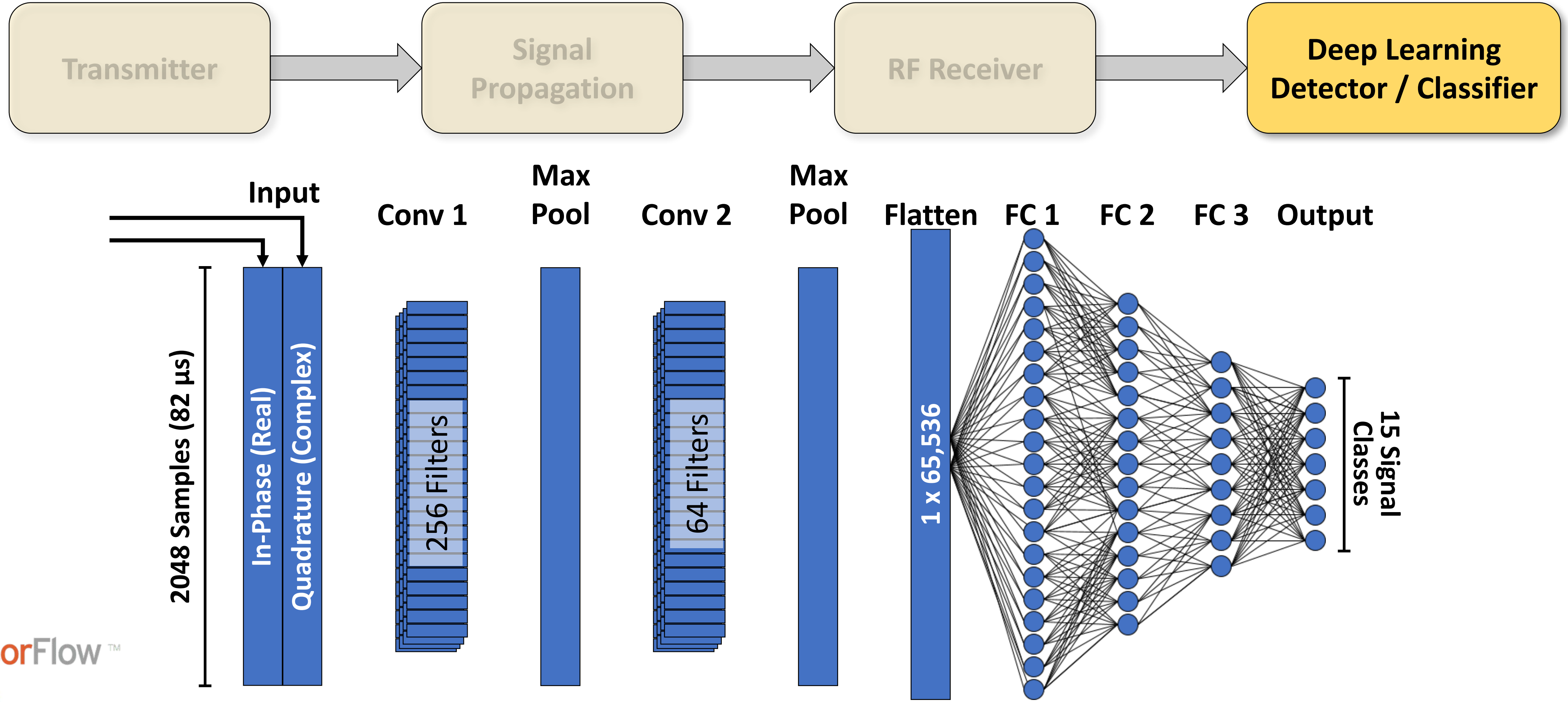
Radar Signal Detector Model: Receiver





- 5 dB Noise Figure
- 65 dB System Gain
- 14 Bits ADC

Radar Signal Detector Model: Example Classifier



2 x 2048 | 2 x 16 | 1 x 2 | 1 x 16 | 1 x 2 | 1 x 128 | 1 x 64 | 1 x 32 | 1 x 15

TensorFlow™

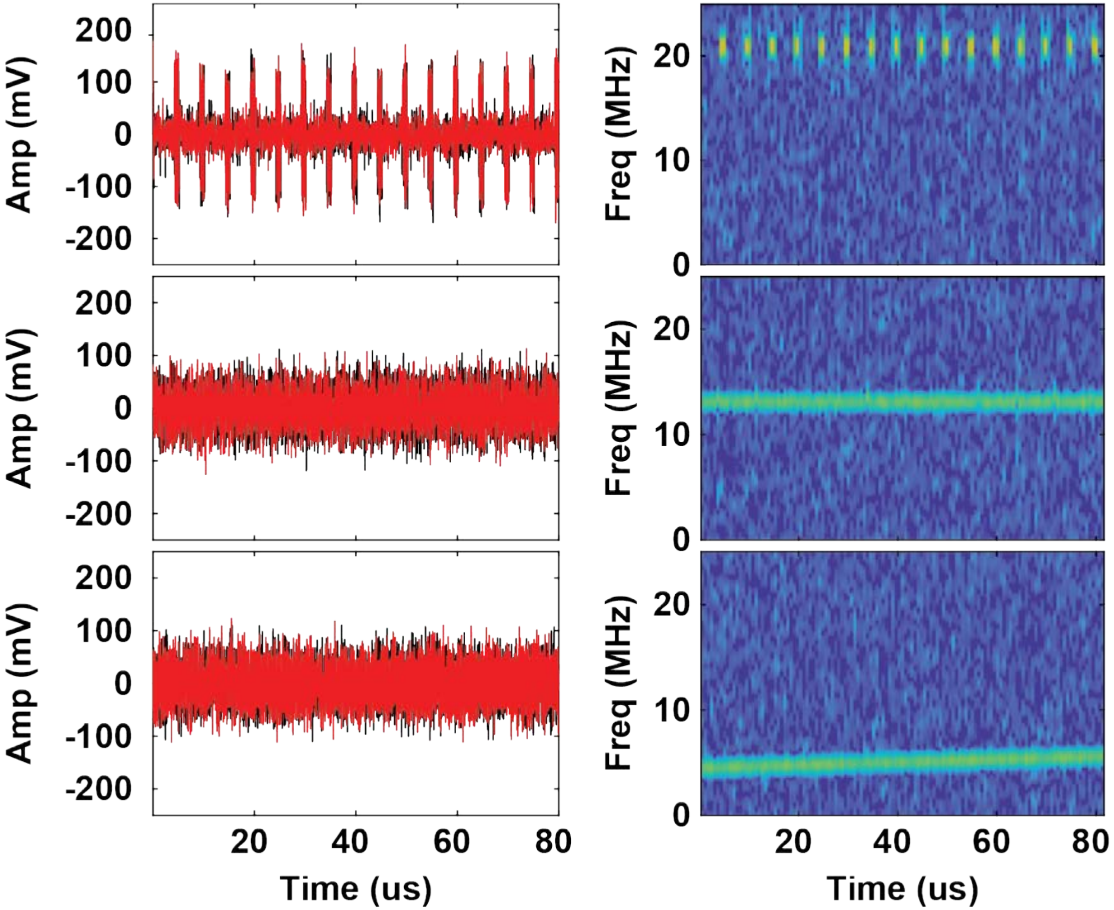
cuDNN

Outline

- Introduction
- Deep Learning in RF Systems
- Deepwave Digital Technology
- Example Signal Detection and Classification
- Training and Inference of Classifier
- Deploying a Deep Learning RF Systems

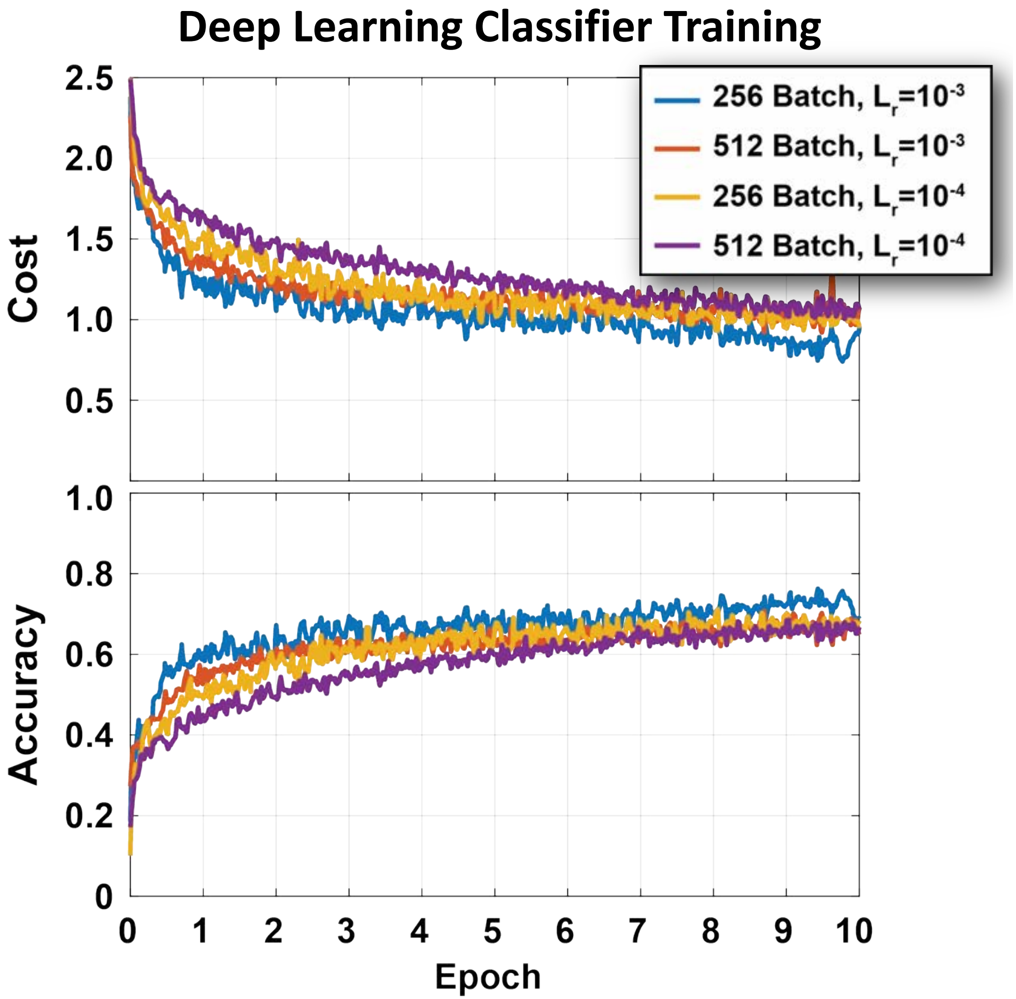
Dataset Overview

- Goal: Develop a deep learning classifier that detects signals below noise floor
  - Requires training on noisy data
- Swept SNR from -35 dB to 20 dB in 1 dB increments
  - 1000 training segments per SNR
  - 500 inference segments per SNR
- Used MATLAB to create data

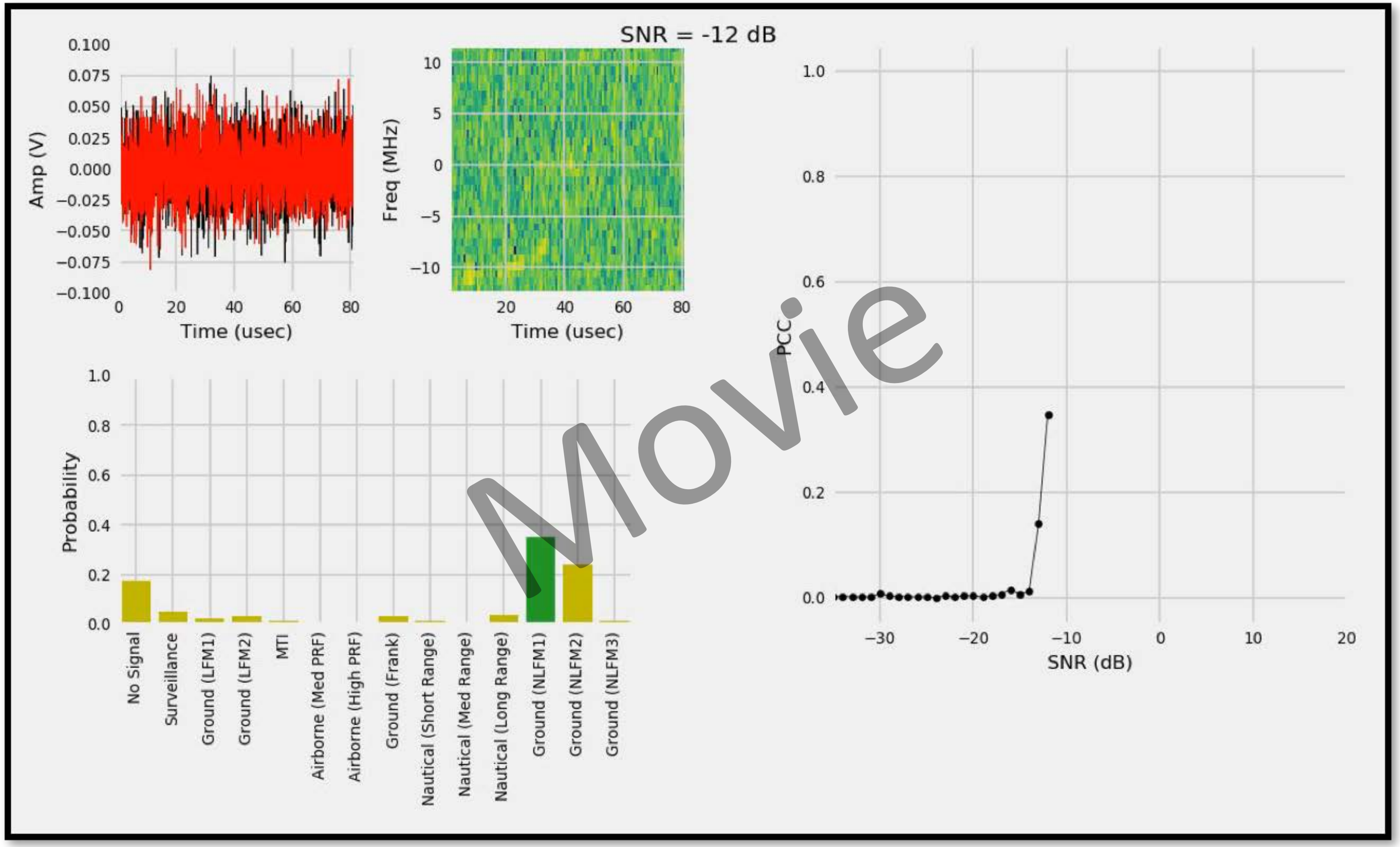


Training Process and Progress

- 1000 training segments per SNR
  - 55 different SNR values
- Training on low SNR values increase detection sensitivity
- 100% accuracy not expected due to training at extremely low SNR values
- Softmax cross entropy
- Adam Optimizer

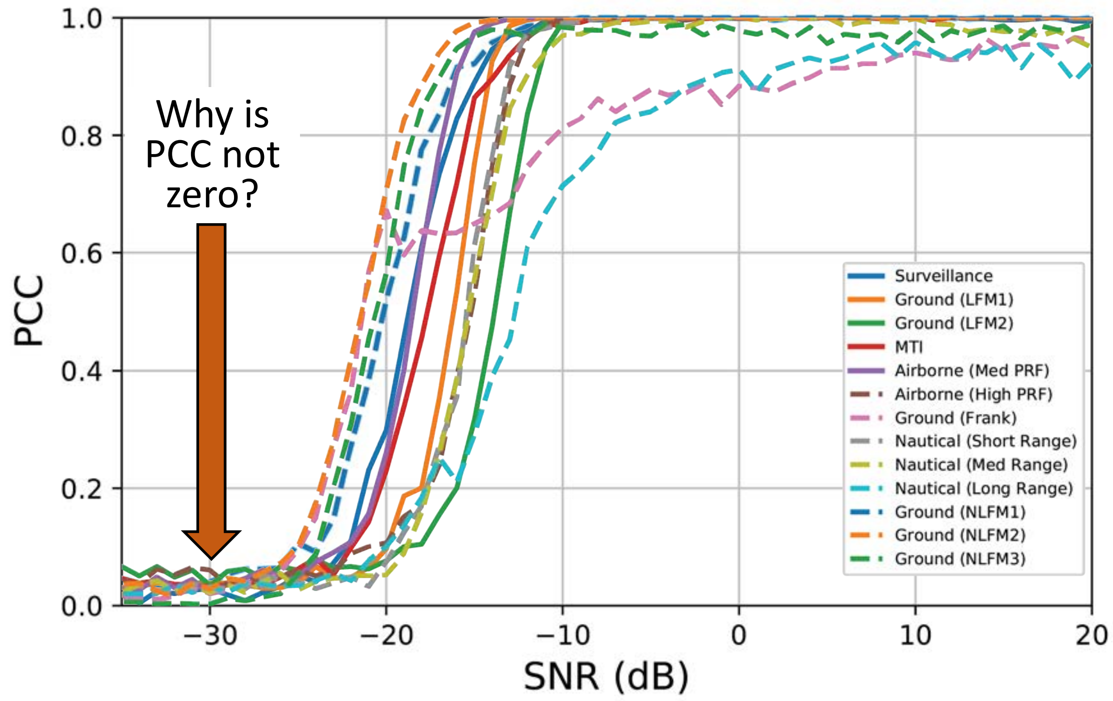

Deep Learning Classifier Training

Detecting and Classifying Low Power Signals



Receiver Operating Characteristic (ROC) Curve

Probability of Correct Classification for Various Radars

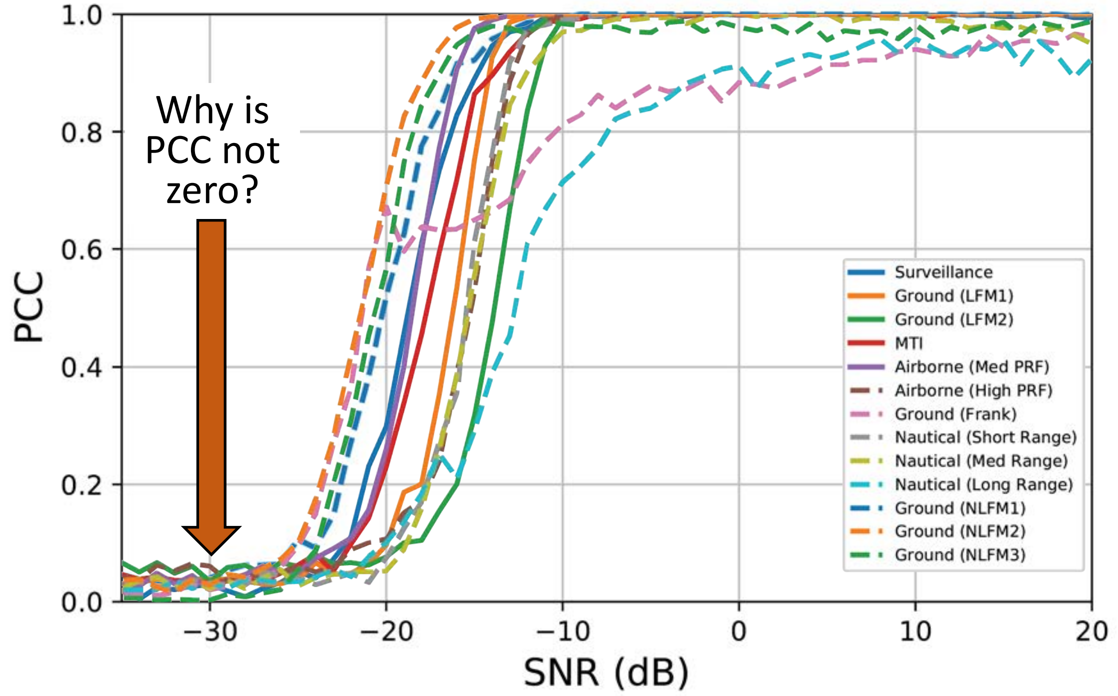


Decibel (dB) Refresher

Signal-to-Noise Ratio (dB)	Receiver Noise Power (milliwatts)	Received Signal Power (milliwatts)
20	1	100
10	1	10
0	1	1
-10	1	0.1
-20	1	0.01
-30	1	0.001

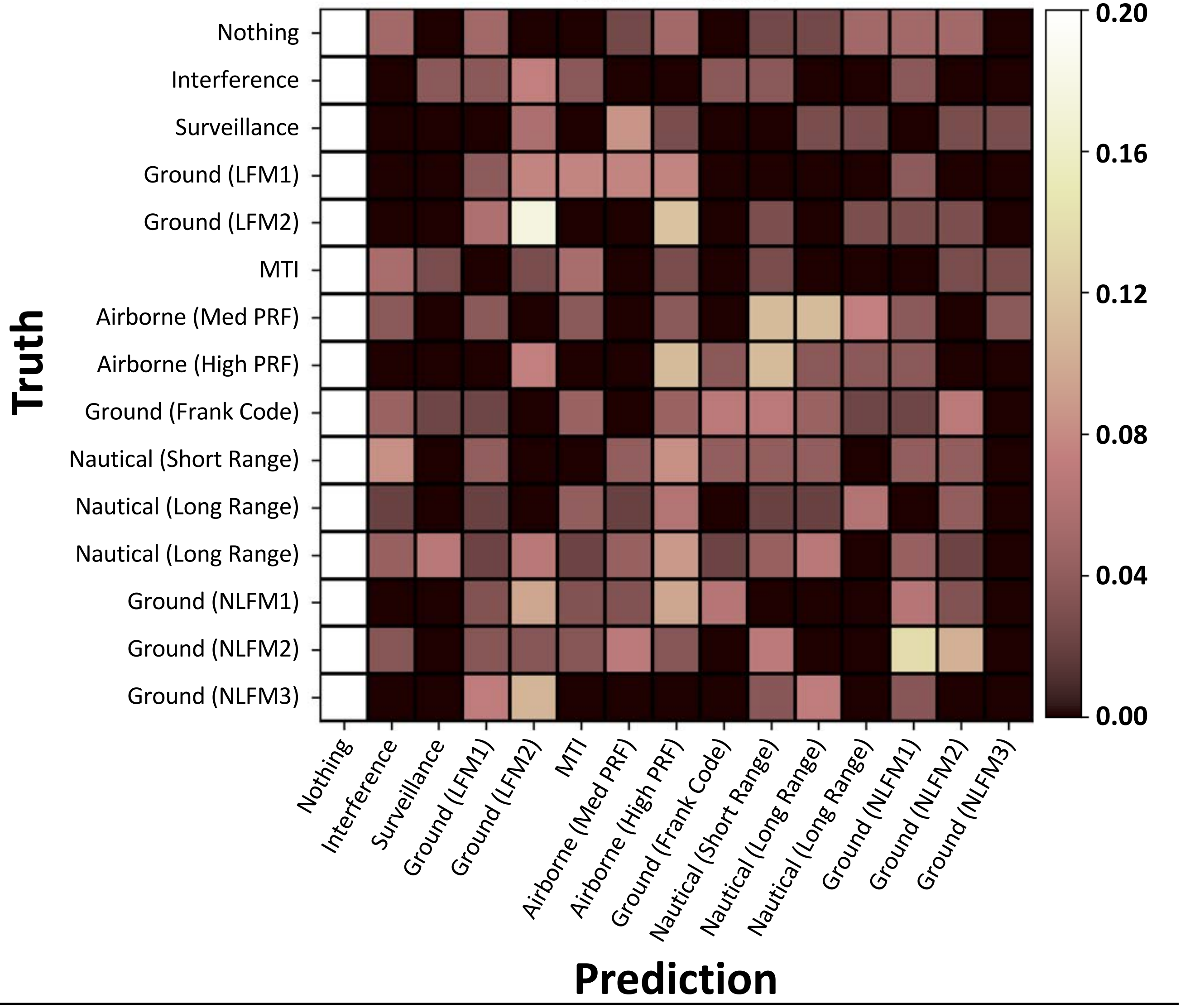
Receiver Operating Characteristic (ROC) Curve

Probability of Correct Classification for Various Radars

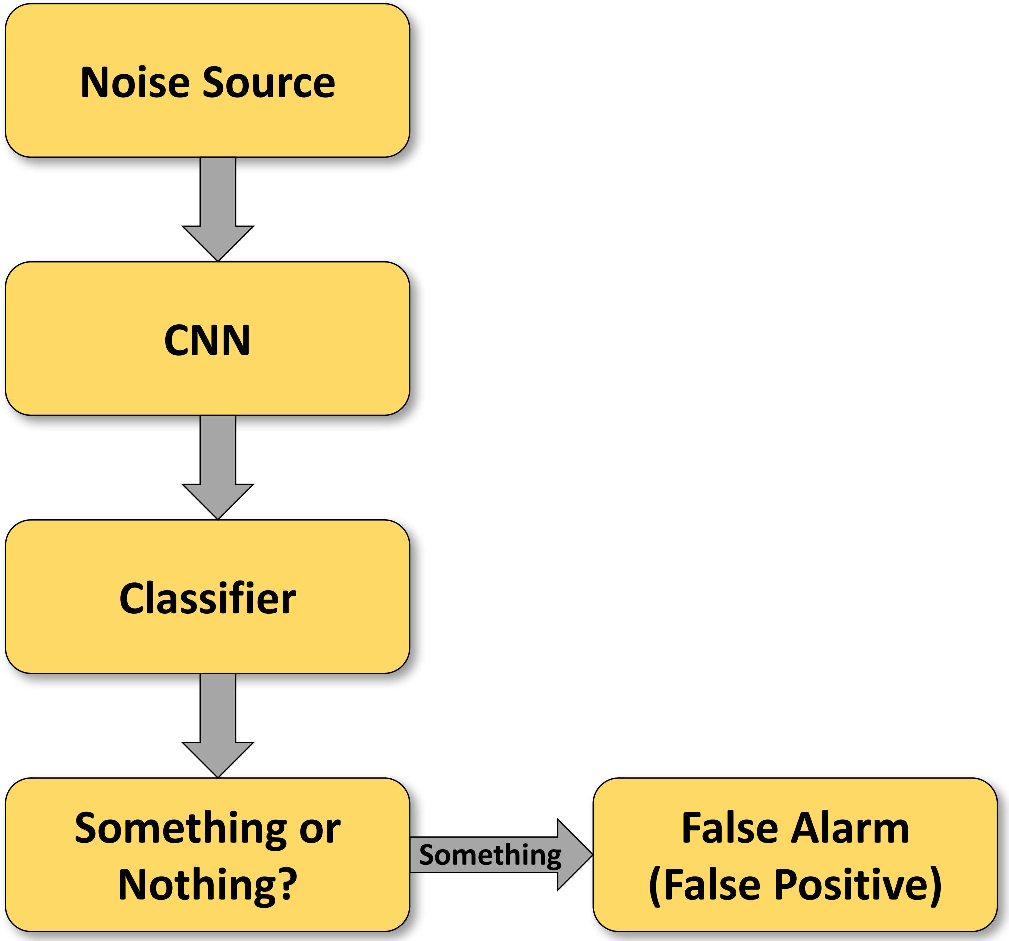


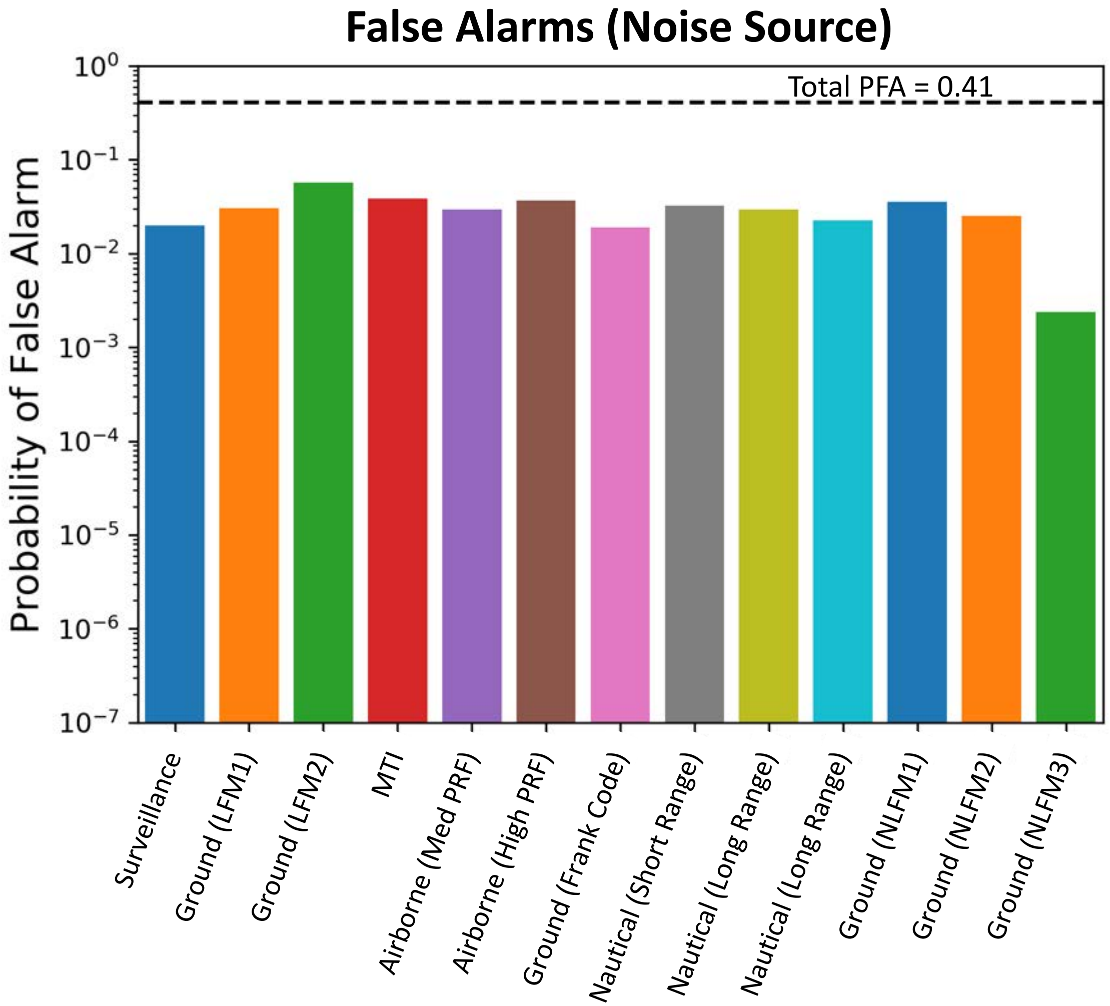
Why is PCC not zero?

Confusion Matrix (-35 dB SNR)

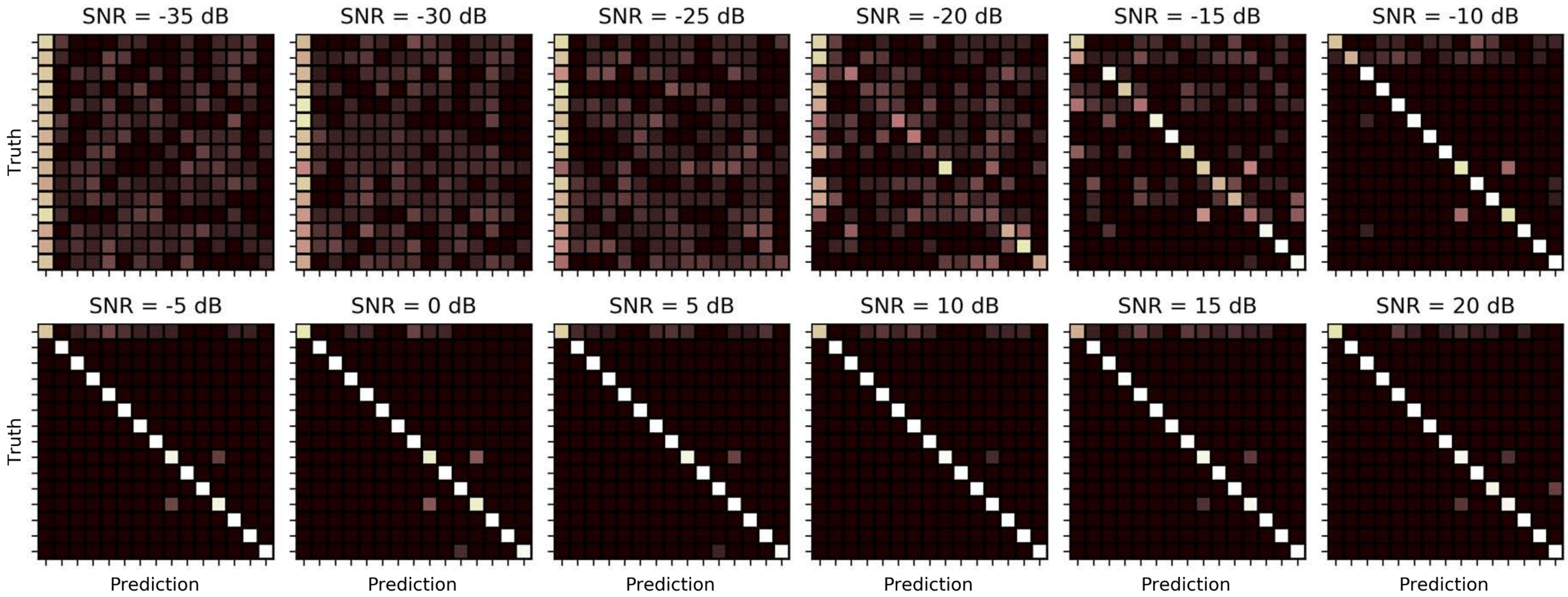


Measuring Probability of False Alarm





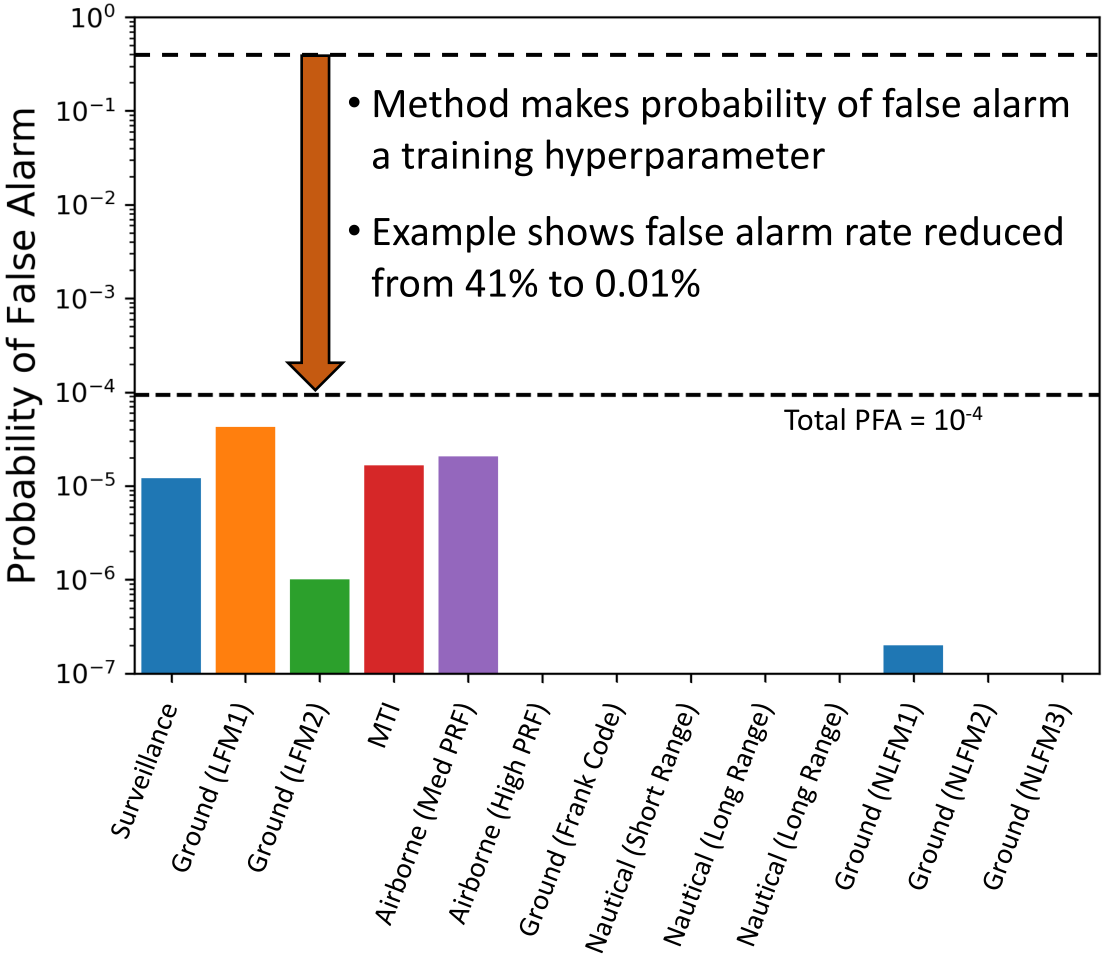
Confusion Matrix and Signal to Noise Ratio



Significant false alarm rate limits algorithm’s applicability and creates non-zero probability of correct classification (PCC) at low SNR values

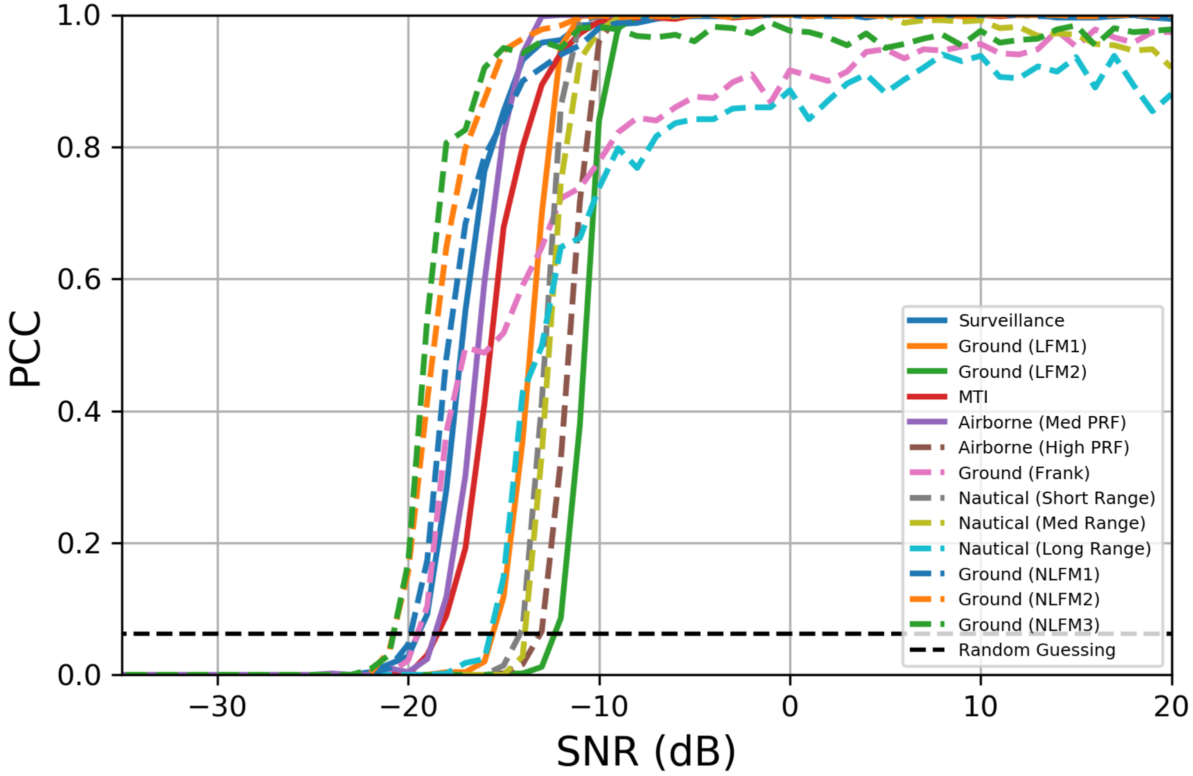
Deepwave Training Method to Reduce False Alarms

False Alarms (Noise Only)

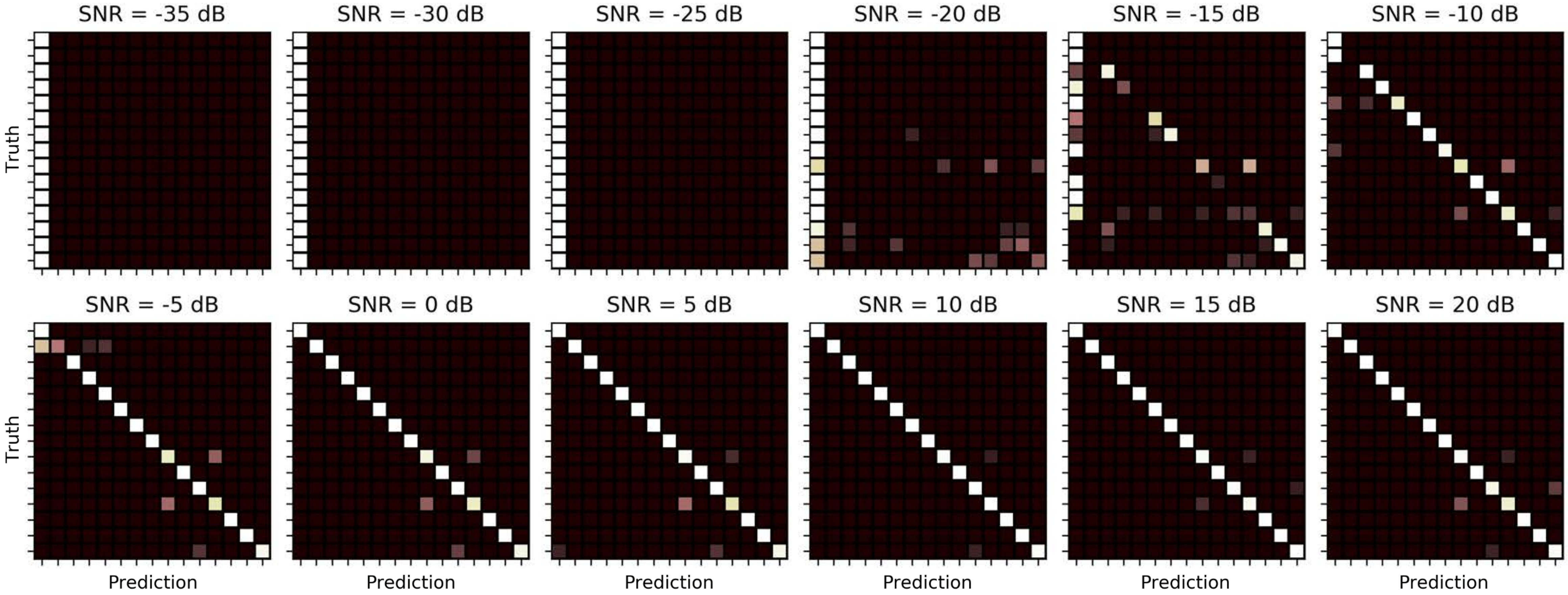


- Method makes probability of false alarm a training hyperparameter
- Example shows false alarm rate reduced from 41% to 0.01%

Probability of Correct Classification for Various Radars



New Inference Confusion Matrix



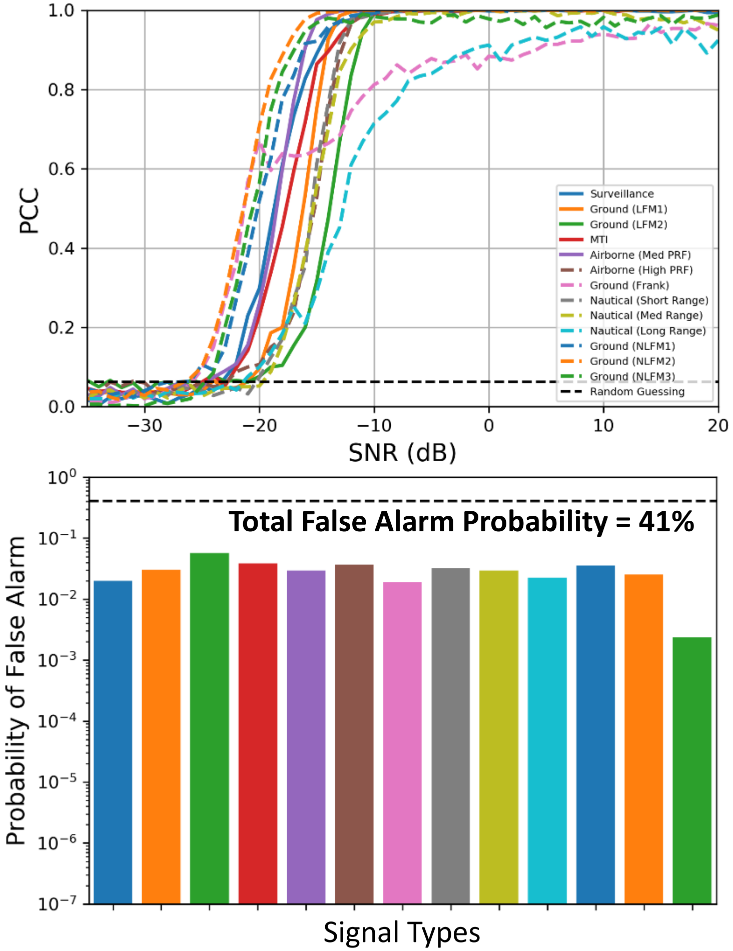
Deepwave training algorithms allows for tunability of false alarm rate

Deepwave Method to Reduce False Alarms

Probability of Detecting and Classifying Signal

Probability of False Alarm (False Positive)

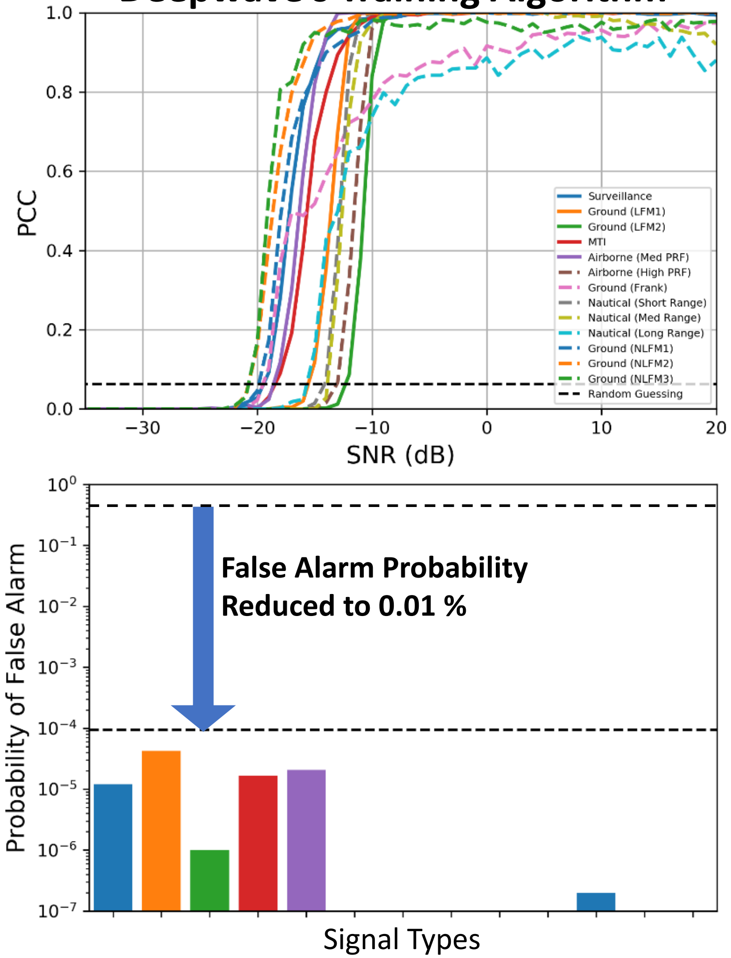
Trained Neural Network



Total False Alarm Probability = 41%

Apply Deepwave's False Alarm Learning Algorithm

Deepwave's Training Algorithm



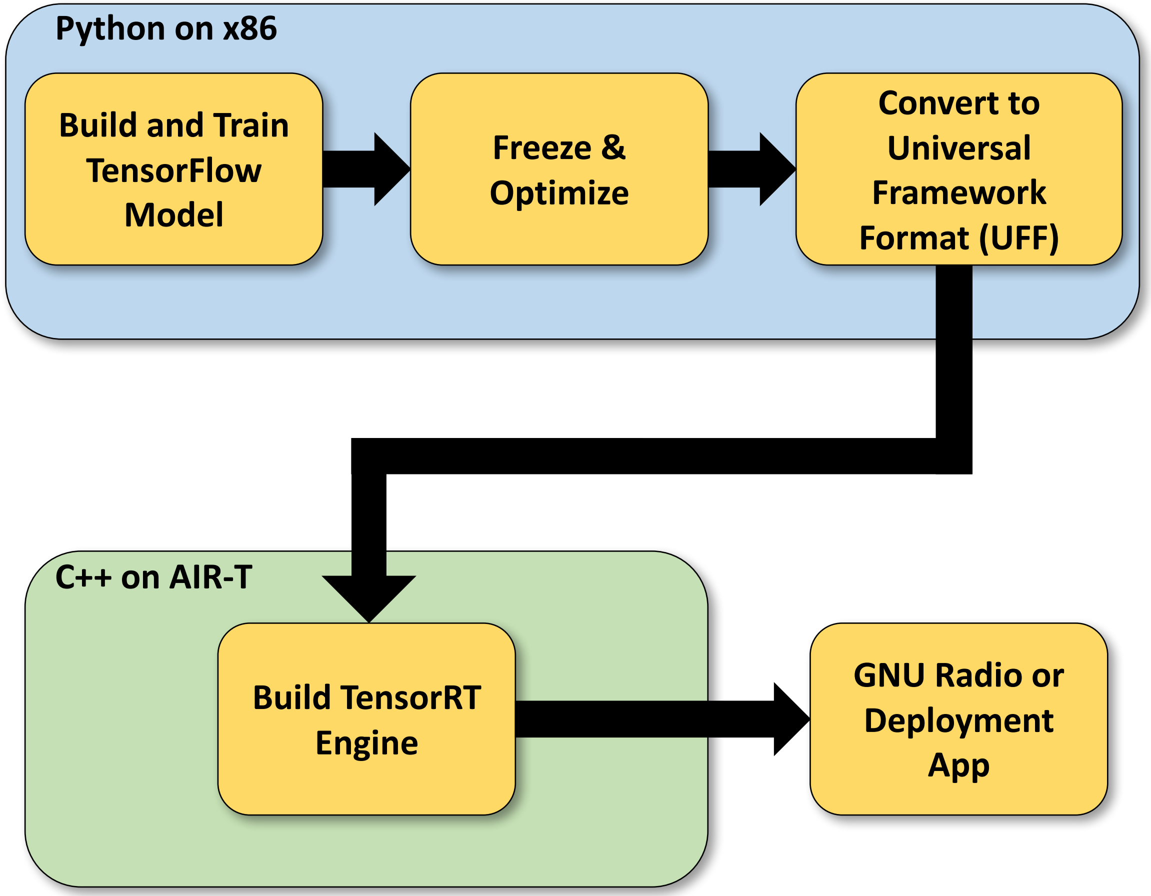
False Alarm Probability Reduced to 0.01 %

False alarm conscious training method has < 5dB impact on detector sensitivity

Outline

- Introduction
- Deep Learning in RF Systems
- Deepwave Digital Technology
- Example Signal Detection and Classification
- Training and Inference of Classifier
- ➡ Deploying a Deep Learning RF Systems

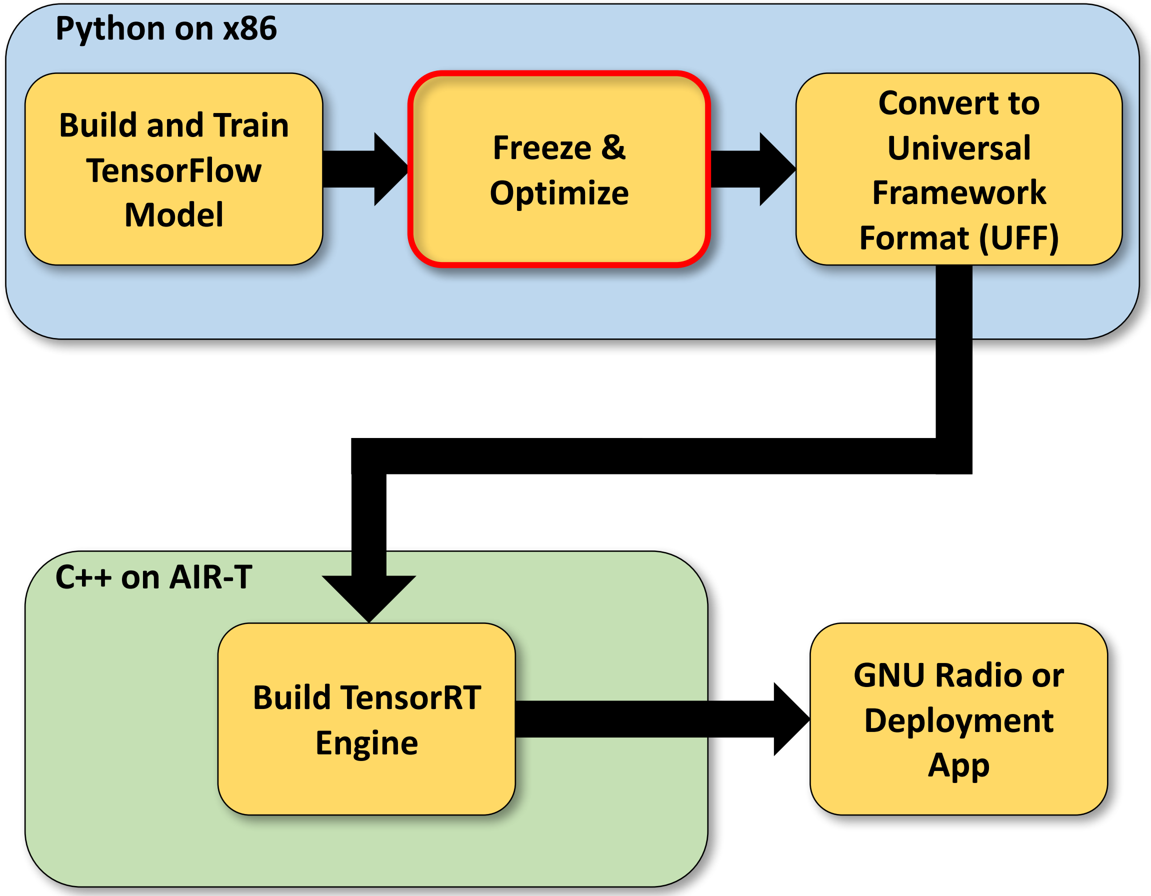
Creating Inference Engine for GNU Radio


Python on x86
Build and Train TensorFlow Model
Freeze & Optimize
Convert to Universal Framework Format (UFF)
C++ on AIR-T
Build TensorRT Engine
GNU Radio or Deployment App

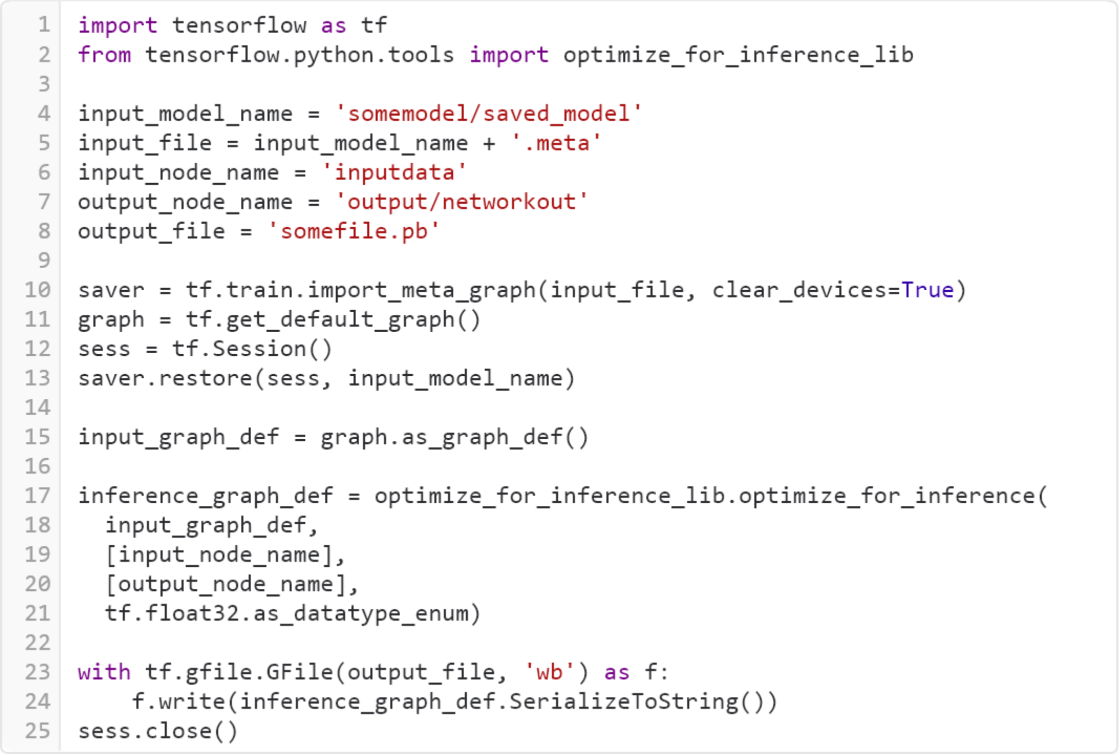
Implementation Caveats

- Yesterday’s announcement of TensorRT support in native TensorFlow is not included
- TensorRT Python API not supported on ARM
  - Some C++ required to build and run inference
  - Several steps can still be performed in Python, but on an x86 machine
- Inference performance tied to TensorRT kernel selection and optimization
  - RF case is somewhat unique (i.e., we’re not processing images)
    - Unique network shape
    - Possible that optimizations have yet to be applied
- GNU Radio does not support float16

Creating Inference Engine for GNU Radio

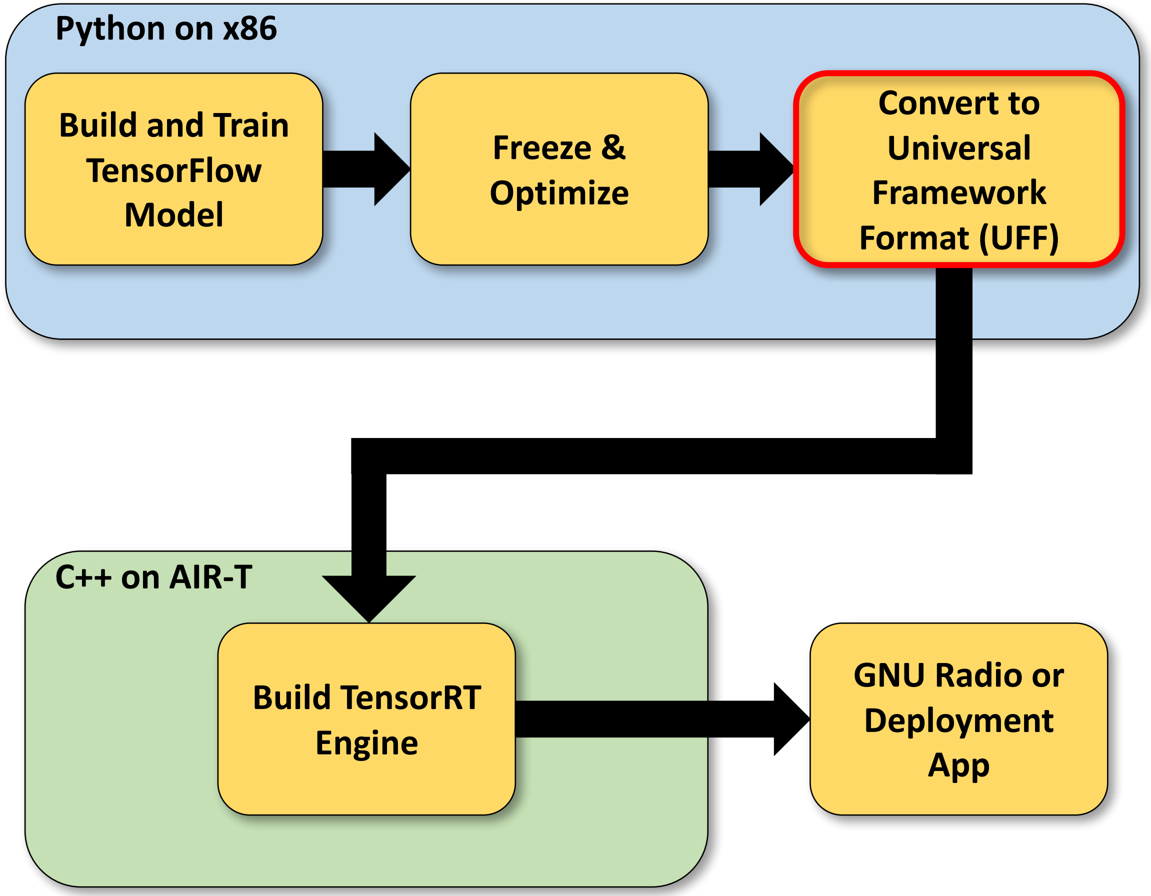

Python on x86
Build and Train TensorFlow Model
Freeze & Optimize
Convert to Universal Framework Format (UFF)
C++ on AIR-T
Build TensorRT Engine
GNU Radio or Deployment App

Freeze and Optimize Trained Model

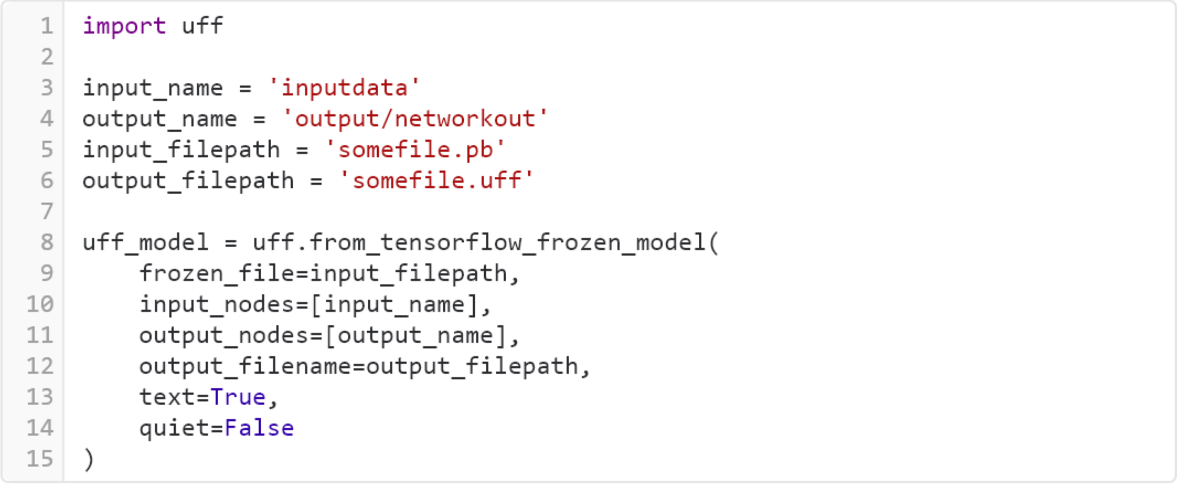


```
import tensorflow as tf
from tensorflow.python.tools import optimize_for_inference_lib

input_model_name = 'somemodel/saved_model'
input_file = input_model_name + '.meta'
input_node_name = 'inputdata'
output_node_name = 'output/networkout'
output_file = 'somefile.pb'

saver = tf.train.import_meta_graph(input_file, clear_devices=True)
graph = tf.get_default_graph()
sess = tf.Session()
saver.restore(sess, input_model_name)

input_graph_def = graph.as_graph_def()

inference_graph_def = optimize_for_inference_lib.optimize_for_inference(
  input_graph_def,
  [input_node_name],
  [output_node_name],
  tf.float32.as_datatype_enum)

with tf.gfile.GFile(output_file, 'wb') as f:
    f.write(inference_graph_def.SerializeToString())
sess.close()
```

Creating Inference Engine for GNU Radio

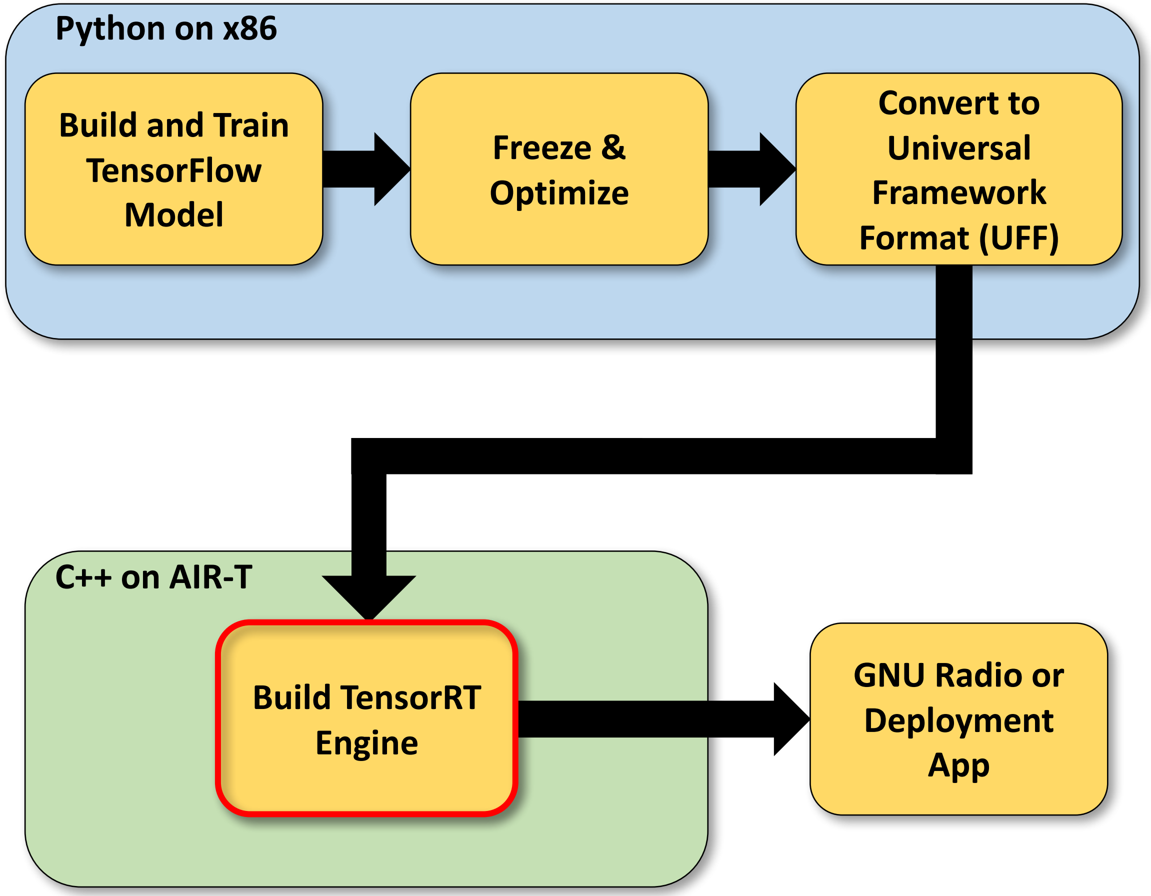

Python on x86
Build and Train TensorFlow Model
Freeze & Optimize
Convert to Universal Framework Format (UFF)
C++ on AIR-T
Build TensorRT Engine
GNU Radio or Deployment App

Convert to UFF

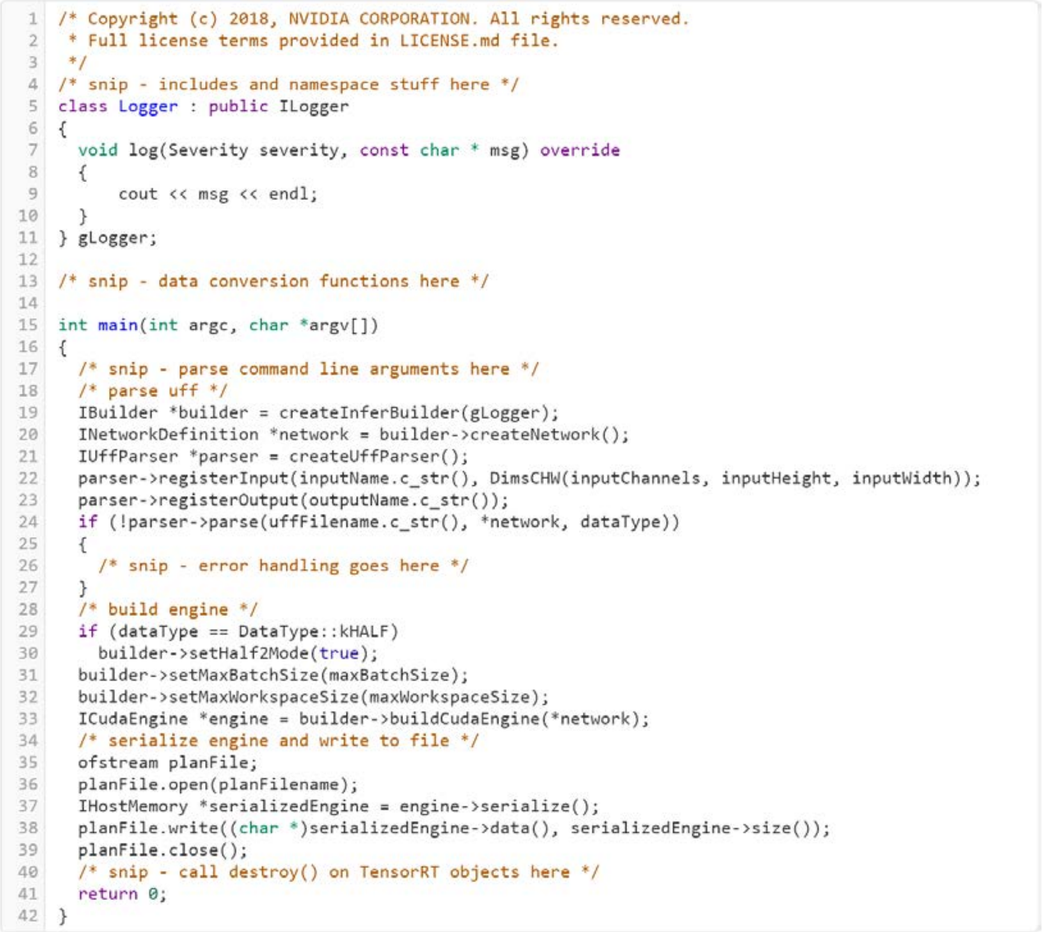


```python
import uff

input_name = 'inputdata'
output_name = 'output/networkout'
input_filepath = 'somefile.pb'
output_filepath = 'somefile.uff'

uff_model = uff.from_tensorflow_frozen_model(
    frozen_file=input_filepath,
    input_nodes=[input_name],
    output_nodes=[output_name],
    output_filename=output_filepath,
    text=True,
    quiet=False
)
```

Creating Inference Engine for GNU Radio

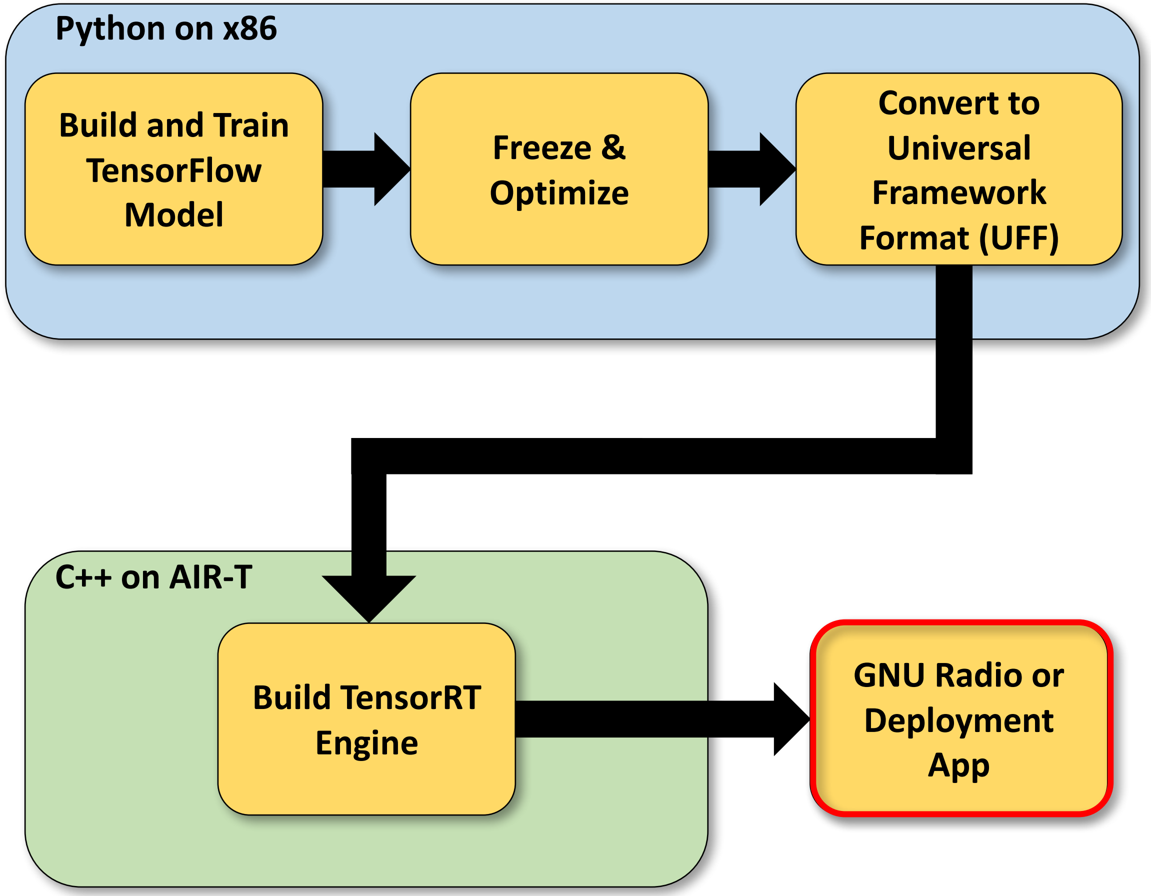

Python on x86
Build and Train TensorFlow Model
Freeze & Optimize
Convert to Universal Framework Format (UFF)
C++ on AIR-T
Build TensorRT Engine
GNU Radio or Deployment App

Build TensorRT Engine

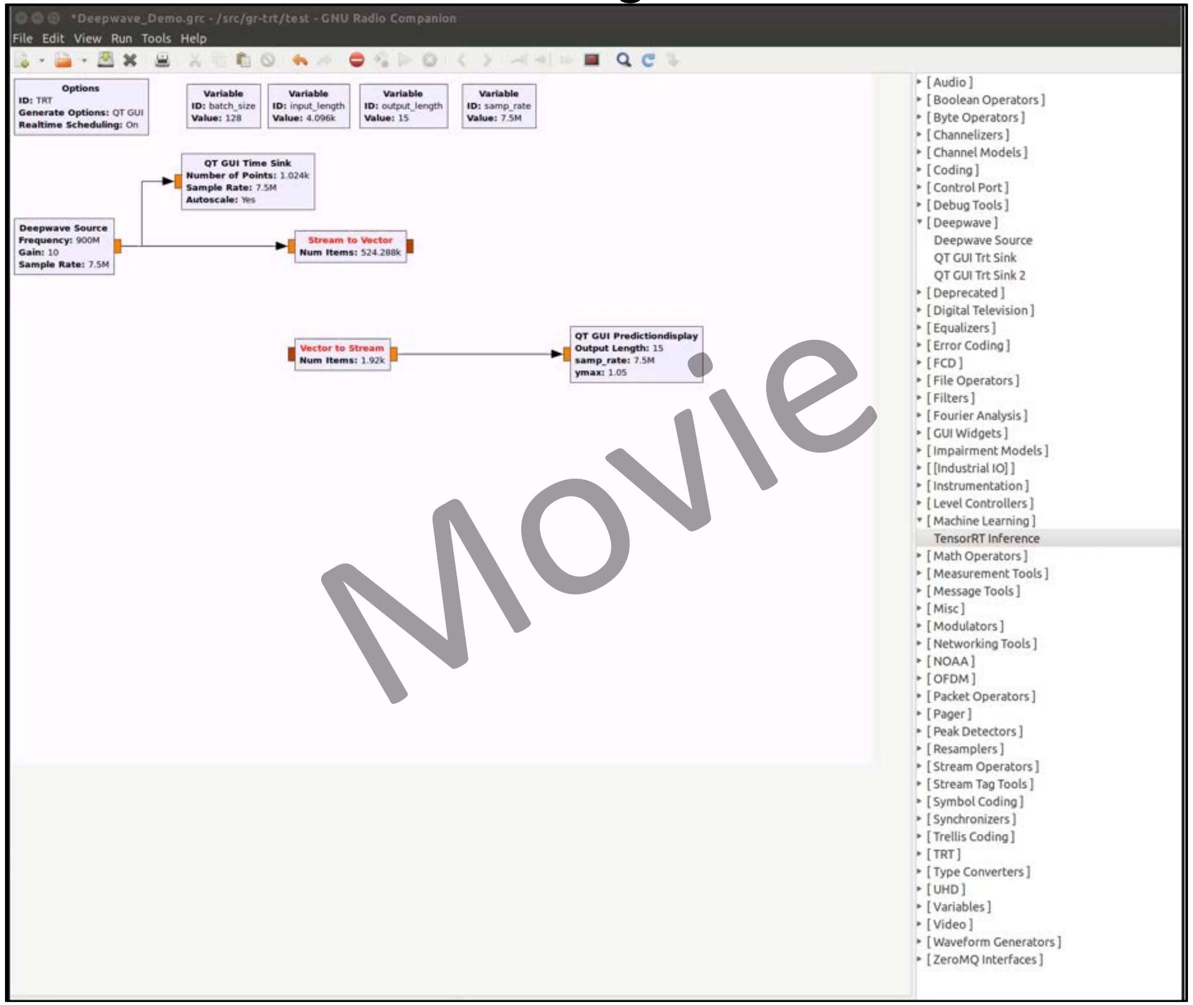


```
/* Copyright (c) 2018, NVIDIA CORPORATION. All rights reserved.
 * Full license terms provided in LICENSE.md file.
 */
/* snip - includes and namespace stuff here */
class Logger : public ILogger
{
  void log(Severity severity, const char * msg) override
  {
      cout << msg << endl;
  }
} gLogger;

/* snip - data conversion functions here */

int main(int argc, char *argv[])
{
  /* snip - parse command line arguments here */
  /* parse uff */
  IBuilder *builder = createInferBuilder(gLogger);
  INetworkDefinition *network = builder->createNetwork();
  IUffParser *parser = createUffParser();
  parser->registerInput(inputName.c_str(), DimsCHW(inputChannels, inputHeight, inputWidth));
  parser->registerOutput(outputName.c_str());
  if (!parser->parse(uffFilename.c_str(), *network, dataType))
  {
    /* snip - error handling goes here */
  }
  /* build engine */
  if (dataType == DataType::kHALF)
    builder->setHalf2Mode(true);
  builder->setMaxBatchSize(maxBatchSize);
  builder->setMaxWorkspaceSize(maxWorkspaceSize);
  ICudaEngine *engine = builder->buildCudaEngine(*network);
  /* serialize engine and write to file */
  ofstream planFile;
  planFile.open(planFilename);
  IHostMemory *serializedEngine = engine->serialize();
  planFile.write((char *)serializedEngine->data(), serializedEngine->size());
  planFile.close();
  /* snip - call destroy() on TensorRT objects here */
  return 0;
}
```

Creating Inference Engine for GNU Radio


Python on x86
Build and Train TensorFlow Model
Freeze & Optimize
Convert to Universal Framework Format (UFF)
C++ on AIR-T
Build TensorRT Engine
GNU Radio or Deployment App

Execute Inference Engine with GNU Radio



Options
ID: TRT
Generate Options: QT GUI
Realtime Scheduling: On

Variable
ID: batch_size
Value: 128

Variable
ID: input_length
Value: 4.096k

Variable
ID: output_length
Value: 15

Variable
ID: samp_rate
Value: 7.5M

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 7.5M
Autoscale: Yes

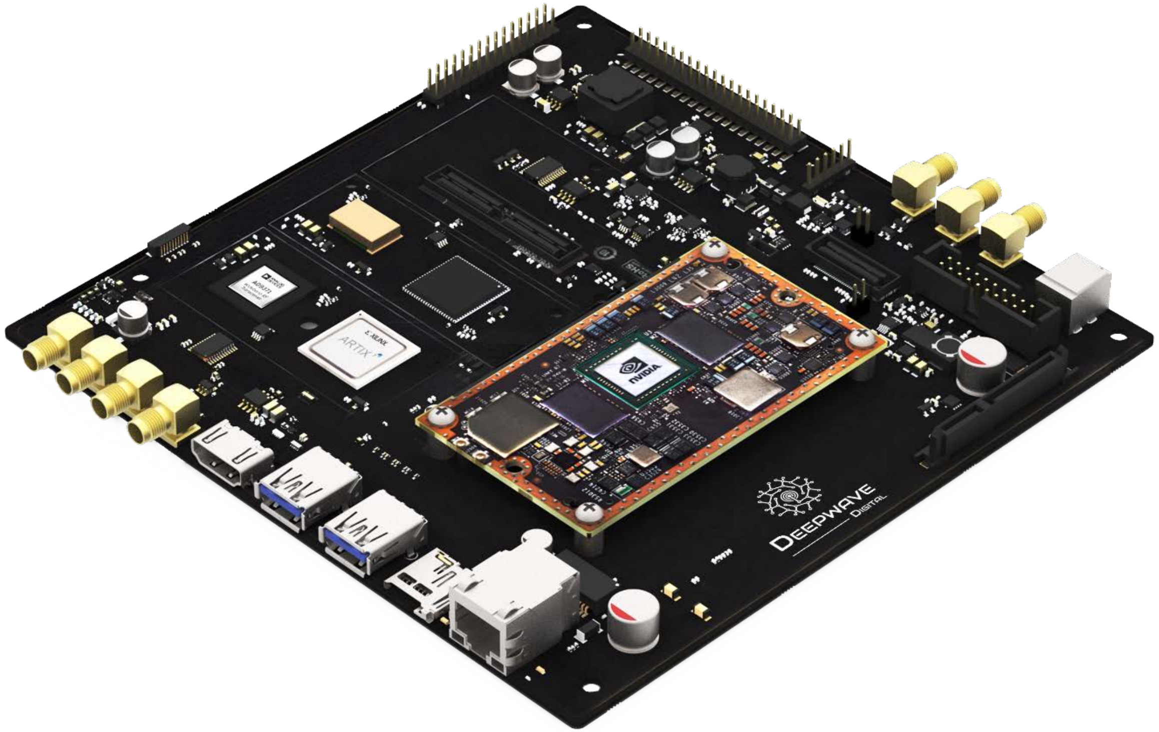
Deepwave Source
Frequency: 900M
Gain: 10
Sample Rate: 7.5M

Stream to Vector
Num Items: 524.288k

Vector to Stream
Num Items: 1.92k

QT GUI Predictiondisplay
Output Length: 15
samp_rate: 7.5M
ymax: 1.05

AIR-T Availability and Pricing



- Presales: Beginning April, 2018
  - 10% discount on all pre-orders
- Anticipated Ship Date: September 2018
- Early product testing available for select institutions:
  - Government and FFRDC labs
  - Currently looking for telecommunications partners

Contact us at sales@deepwavedigital.com

Exact specifications may slightly differ from drawing

DEEPWAVE

DIGITAL