

# DEEPWAVE DIGITAL

---

## Deep Learning and Signal Processing Webinar

March 25, 2020





## WHAT WE DO

Deepwave produces **Hardware & Software Products** enabling customers to develop and deploy deep learning algorithms within radio frequency (RF) and wireless technology.

- Signal ID, interference mitigation, electronic protection

Deepwave builds **Custom Neural Network** solutions for a variety of market verticals.

- Radar, comms, defense, aerospace, embedded systems

## WHO ARE OUR CUSTOMERS?

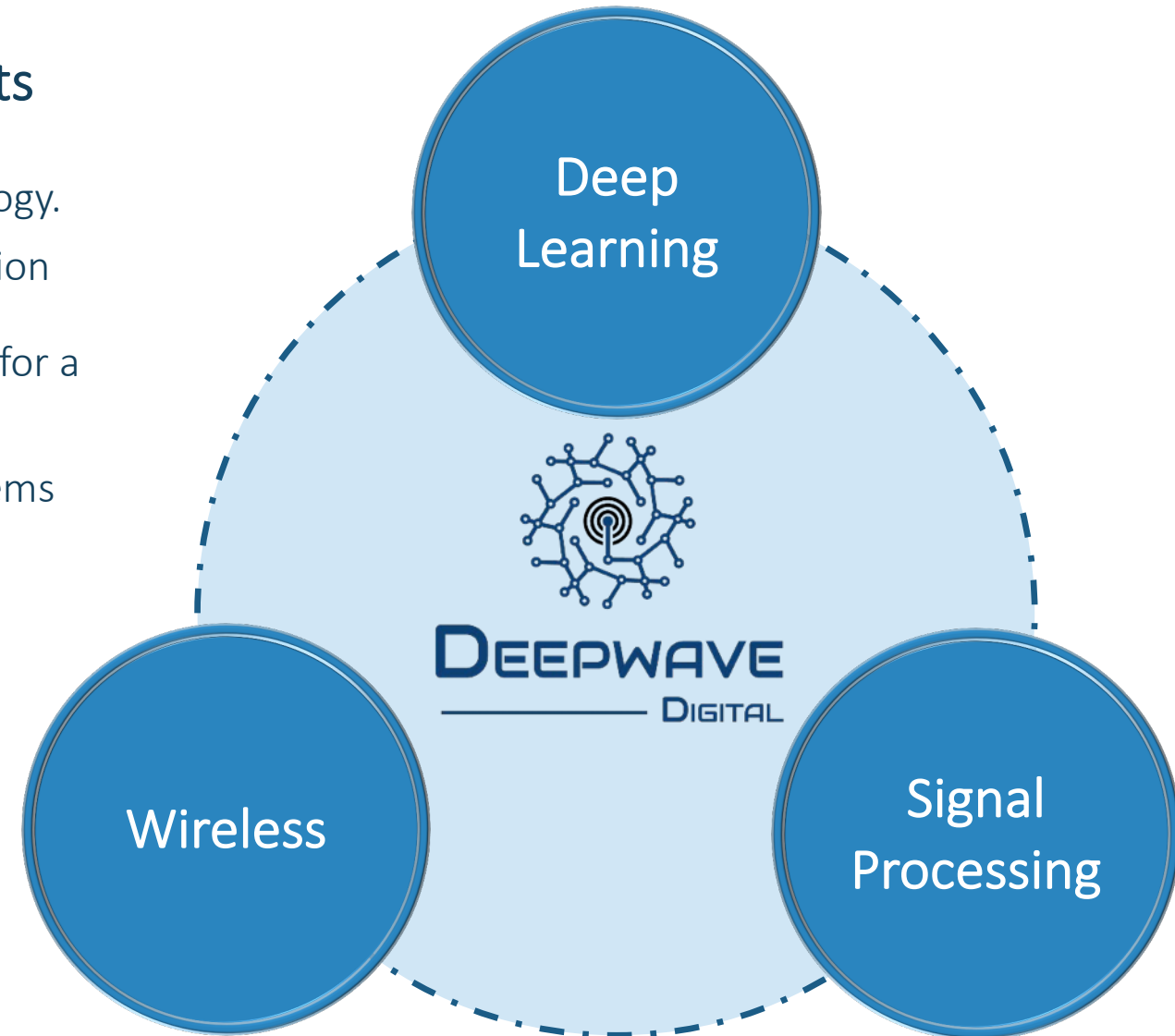


- Commercial wireless infrastructure OEMS
- Wireless Operators
- Defense Contractors
- Government DoD Agencies
- University Research Centers

## HEADQUARTERS



Founded in 2017, our headquarters is located in Philadelphia, PA



# Obstacles for Radio Frequency Systems

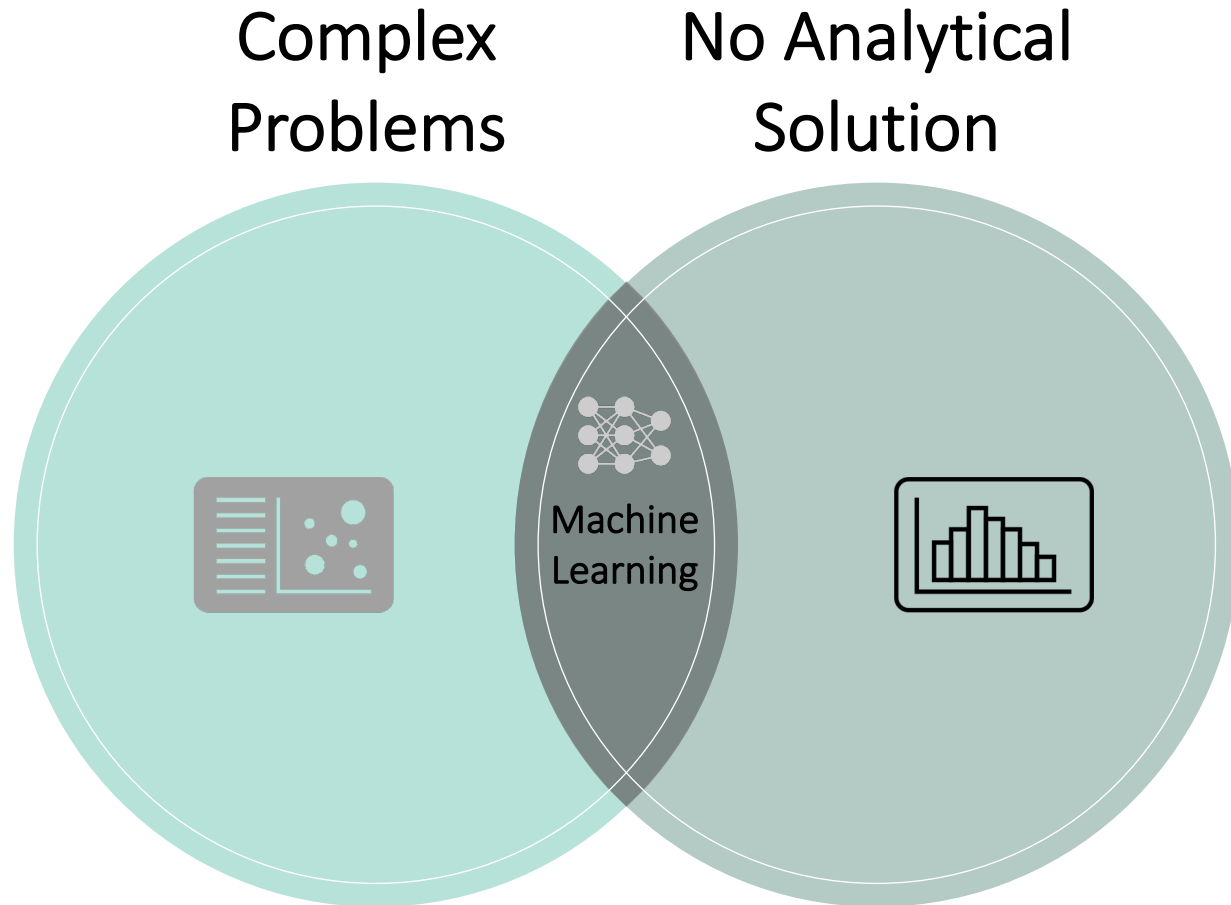
## Seemingly Insurmountable Challenges

**Congested spectrum** – Ever-growing number of devices but the amount of spectrum is fundamentally limited

**Interference** – More devices lead to more interference, limiting data rates and connectivity

**Electronic Attack (Jamming)** – Low cost processors have led to very sophisticated adversarial attacks to disable radar and communications systems

**Security** – Increased number of devices create safety concerns and more sophisticated cyber attacks are increasing in frequency

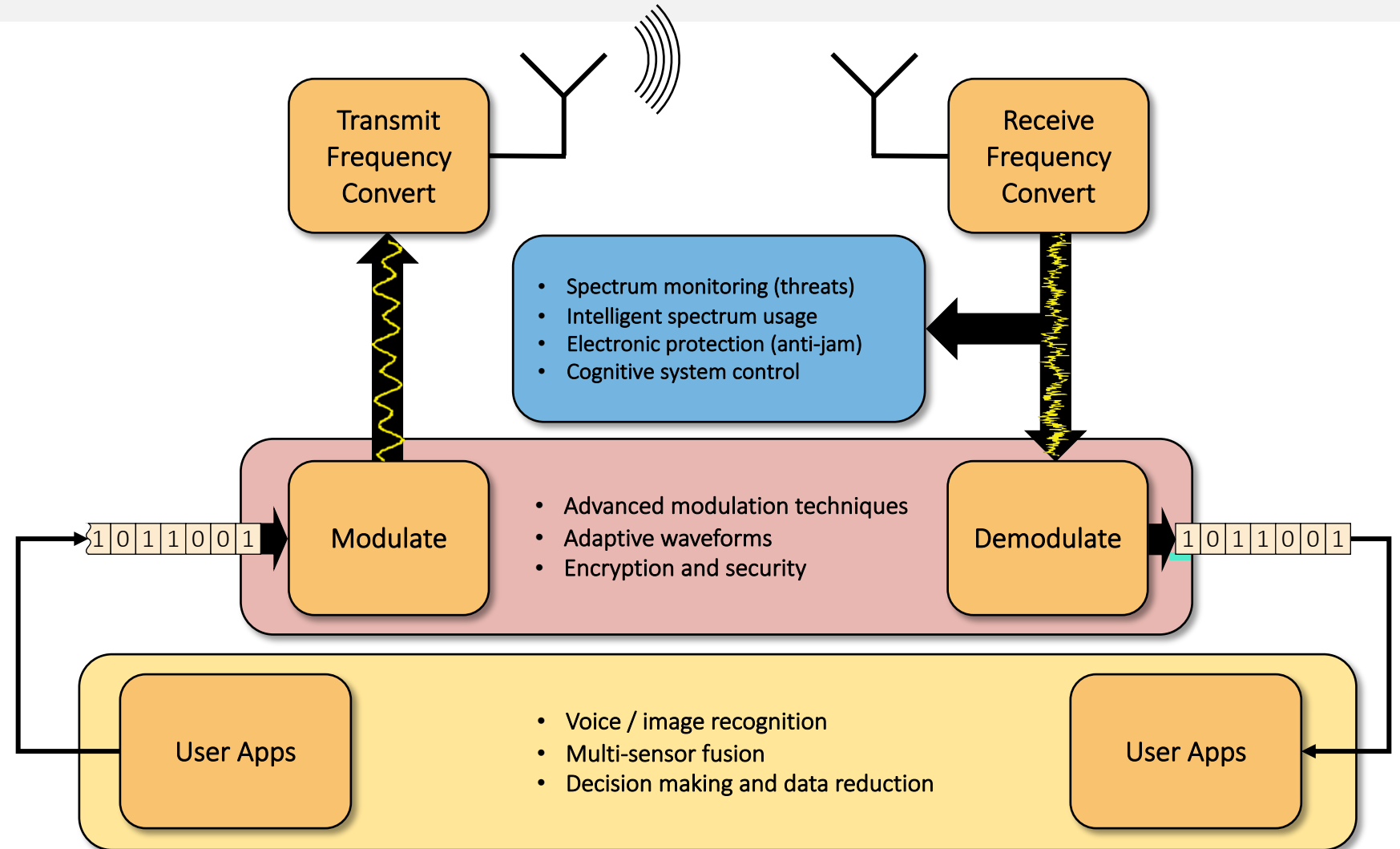


# Where to Use Deep Learning in RF Systems

Spectrum / Network  
Centric Applications

Device / Basestation  
Centric Applications

User App  
Centric Applications





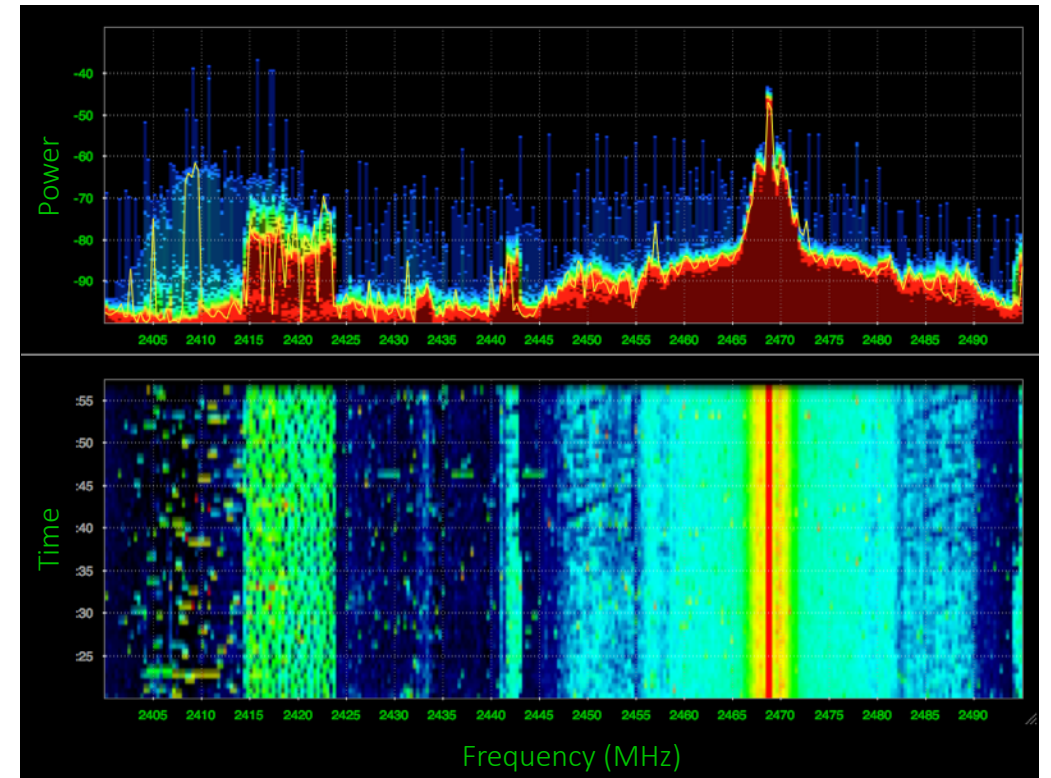
# Solve Complex Problems in Wireless Systems with AI

Example: Artificial Networks for Signal Identification in a Congested Wireless Spectrum

## Simple Example: Image Recognition



## Congested Wireless Spectrum



Deep Learning identifies intricate patterns that are too obscure and subtle to be implemented into a human-engineered algorithm

# Outline

- Introduction
- ➔ • AIR-T System
- AirStack Overview
- Signal Processing Demonstrations
- Deep Learning Workflow
- Summary

# Deepwave's Edge Compute AI/RF Solution

A Seamless Platform for a Multitude of Applications

## The Platform

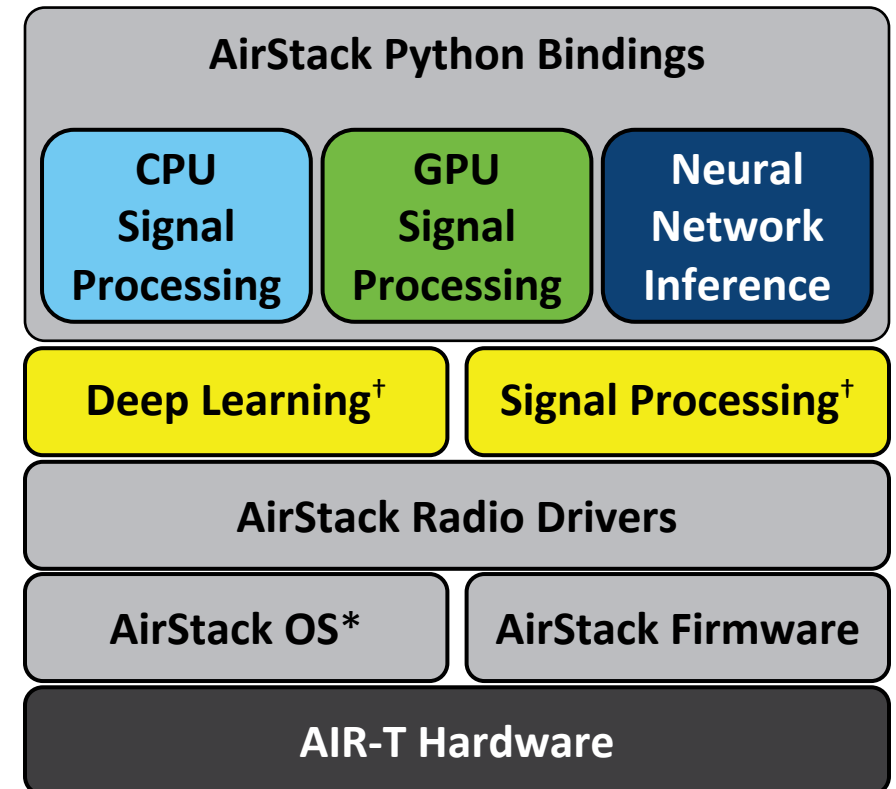
Complete Edge-compute AI Platform for RF



\*Patent Pending

## The Software

Simply build AI into wireless technology



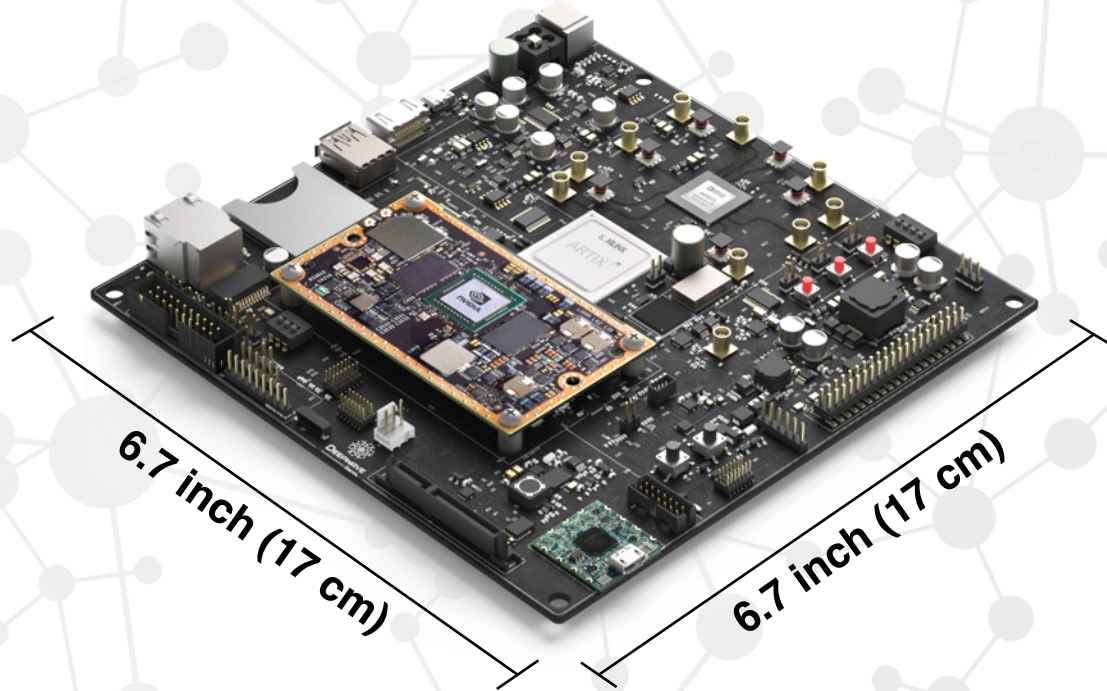
<sup>†</sup> 3<sup>rd</sup> Party APIs

\* Operating system based on NVIDIA Jetpack

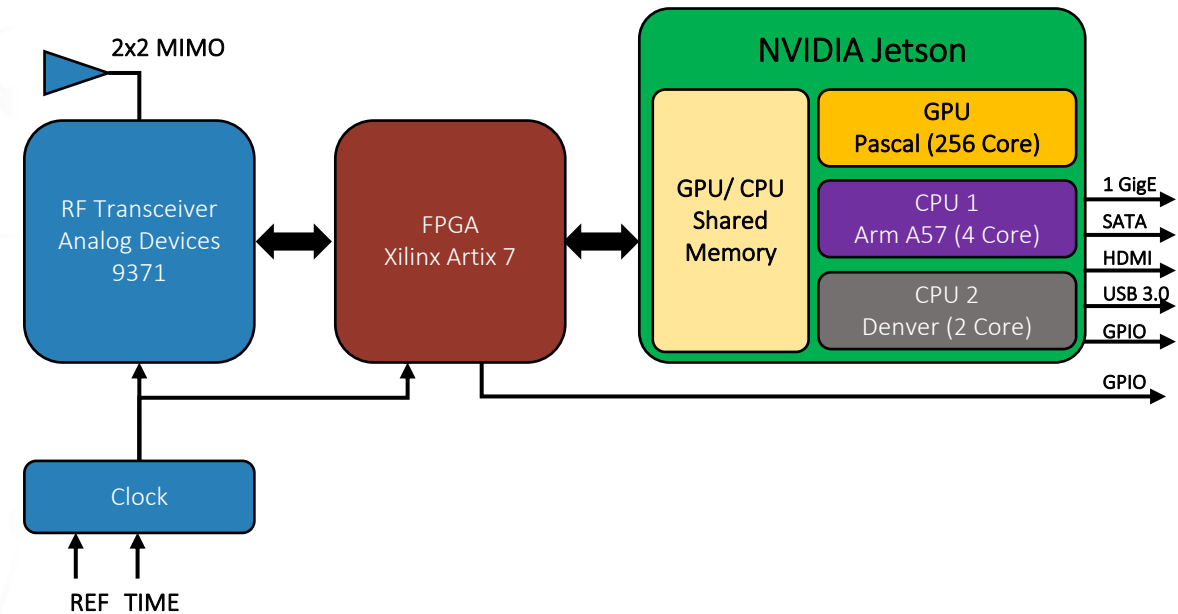
# Artificial Intelligence Radio Transceiver (AIR-T)

The only software defined radio with built-in deep learning processors

## Product



## Block Diagram



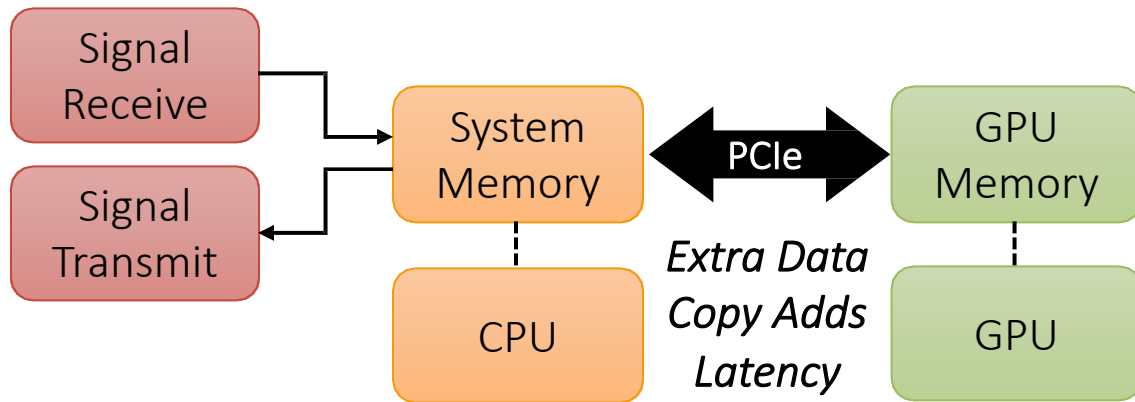
Embedded GPU allows for wideband processing with deep learning in deployed environment



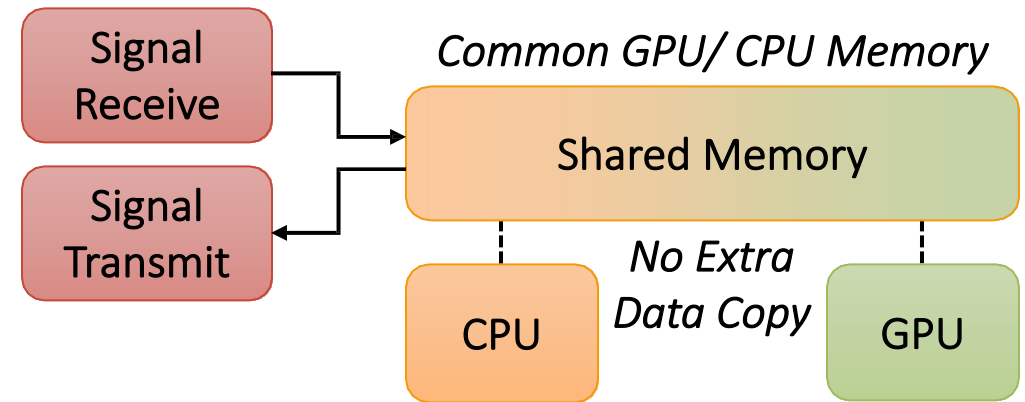
# Artificial Intelligence Radio Transceiver (AIR-T)

Shared Memory Architecture for GPU Processing

## Traditional SDR with GPU



## AIR-T



Embedded GPU eliminates extra data copy by using GPU/CPU shared memory

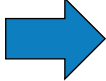
- Up to 50 millisecond latency reduction

# Artificial Intelligence Radio Transceiver (AIR-T)

## Models

	AIR7101	AIR7201
Radio		
Transmit Channels		2
Receive Channels		2
Frequency Tuning Range		300 MHz – 6 GHz
Power Control		AGC / Manual
Graphics Processing Unit (GPU)		
Cores		256
Central Processing Unit (CPU)		
Cores		6 (Dual Processors)
Memory		
GPU/CPU Shared		8 Gbytes
Field Programmable Gate Array (FPGA)		
Family		Xilinx Artix 7
Model	XC7A75T	7XC7A200T
Logic Cells	75,520	215,360
DSP Slices	180	740
Memory	3,780	12,140

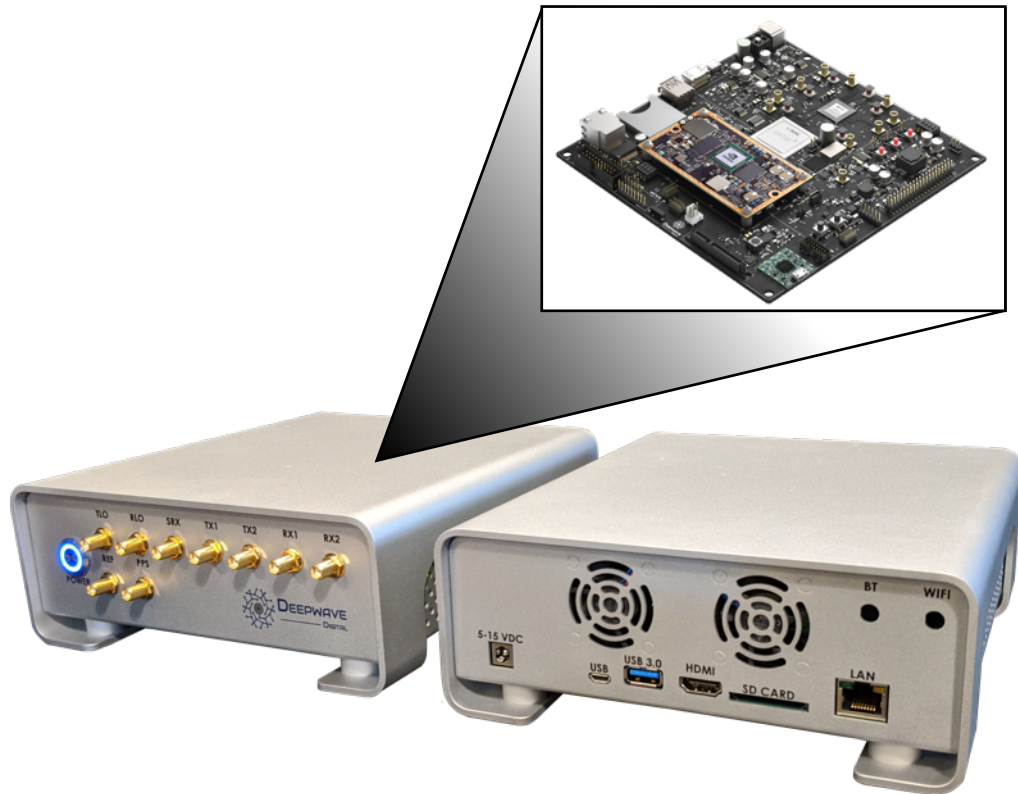
# Outline

- Introduction
- AIR-T System
-  • AirStack Overview
- Signal Processing Demonstrations
- Deep Learning Workflow
- Summary

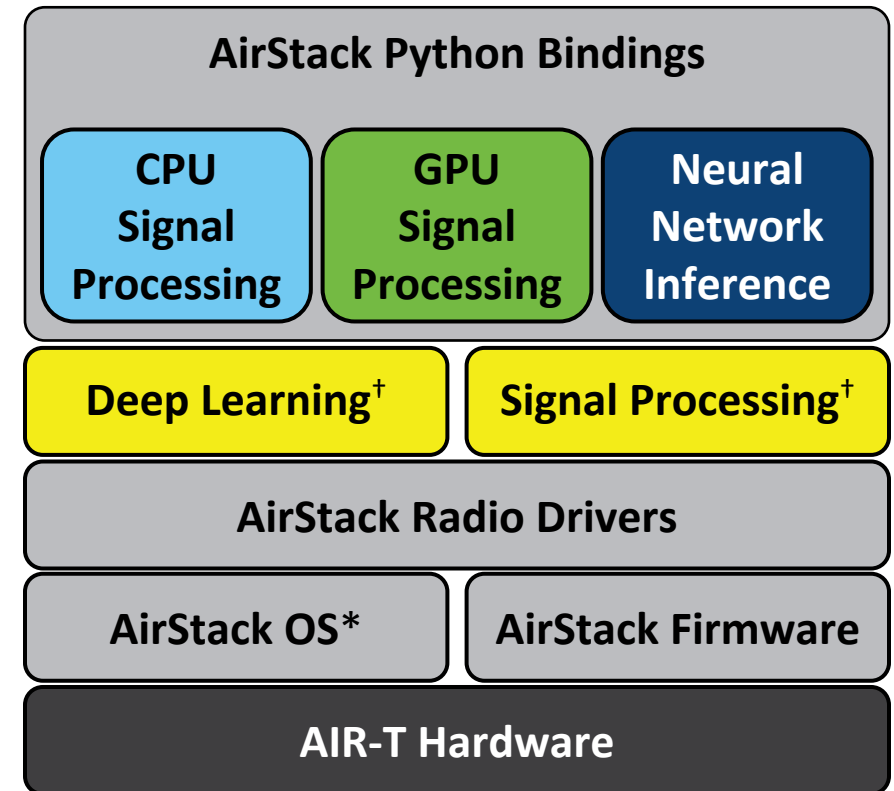
# Artificial Intelligence Radio Transceiver (AIR-T)

Deepwave's AirStack Application Programming Interface (API)

## AIR-T Hardware



## AIR-T Software: AirStack



<sup>†</sup> 3<sup>rd</sup> Party APIs

\* Operating system based on NVIDIA Jetpack



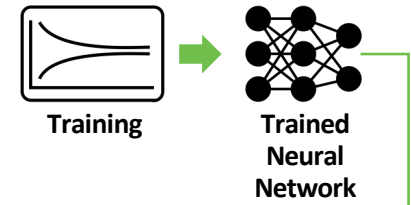
# Artificial Intelligence Radio Transceiver (AIR-T)

Training to Deployment in 3 Steps Easy Step



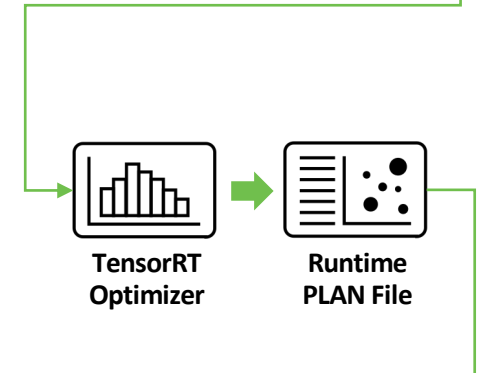
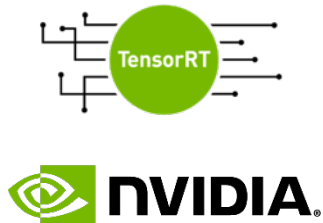
## Step 1 – Train

With any deep learning framework



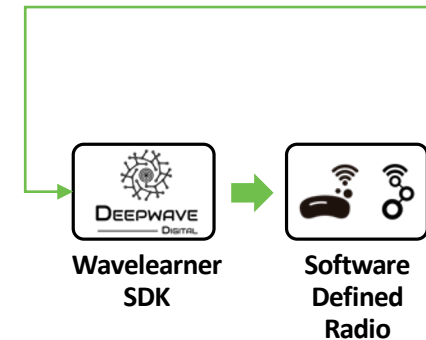
## Step 2 – Optimize

Using NVIDIA's [TensorRT](#)



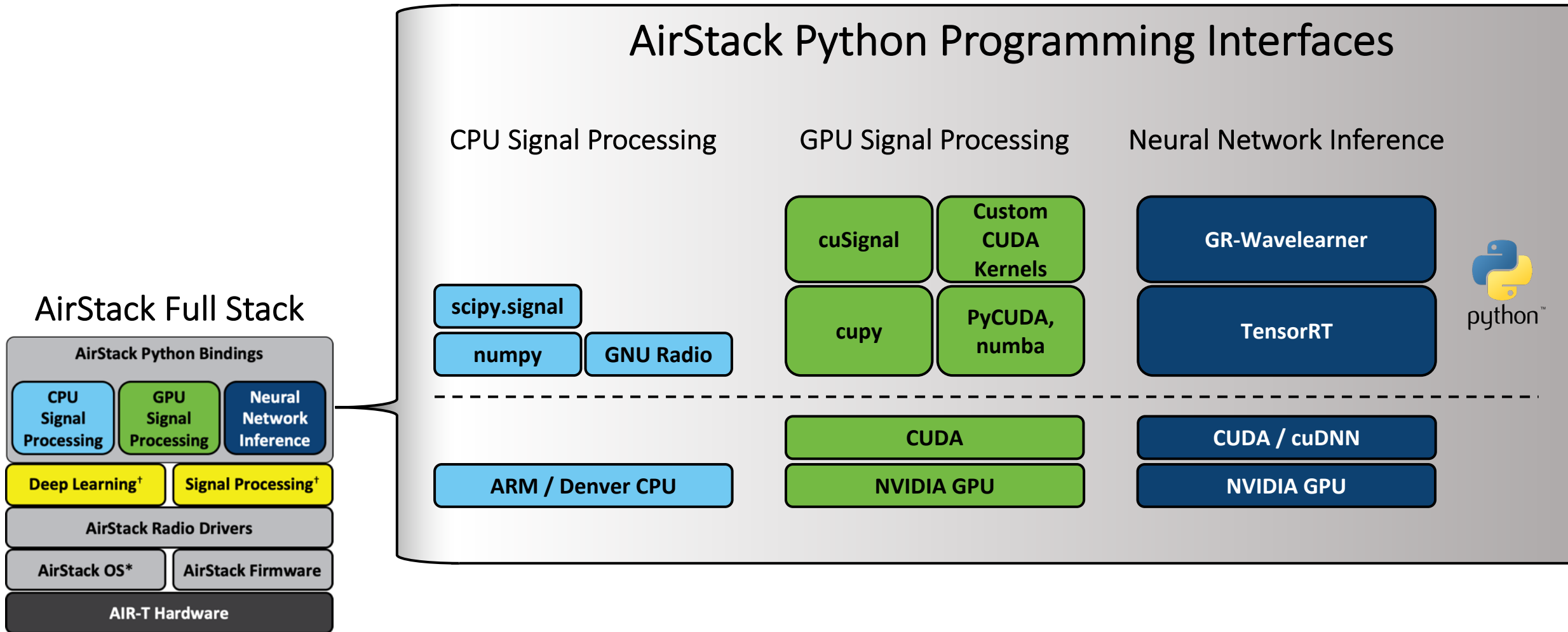
## Step 3 – Deploy on AIR-T

Using Python or Deepwave's [GR-Wavelearner](#) with [GNU Radio](#)



# Artificial Intelligence Radio Transceiver (AIR-T)

Deepwave's AirStack Application Programming Interface (API)



<sup>†</sup> 3<sup>rd</sup> Party APIs

\* Operating system based on NVIDIA Jetpack

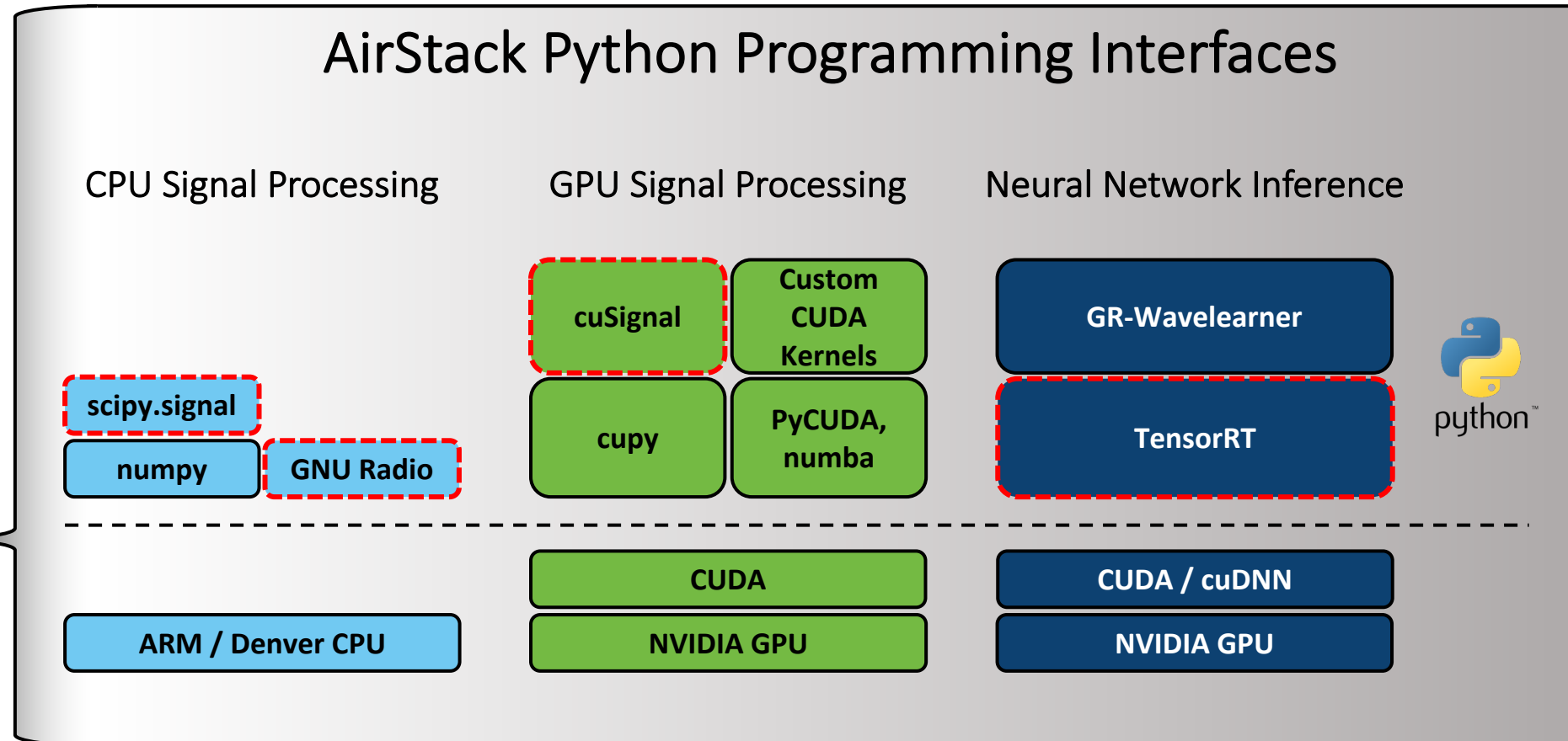
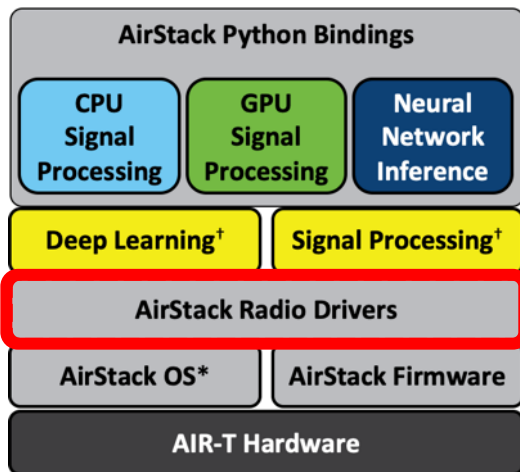
# Artificial Intelligence Radio Transceiver (AIR-T)

Deepwave's AirStack Application Programming Interface (API)

 Discussed Today

 Future Webinar

## AirStack Full Stack



# AIR-T Demonstration Setup

Live Setup

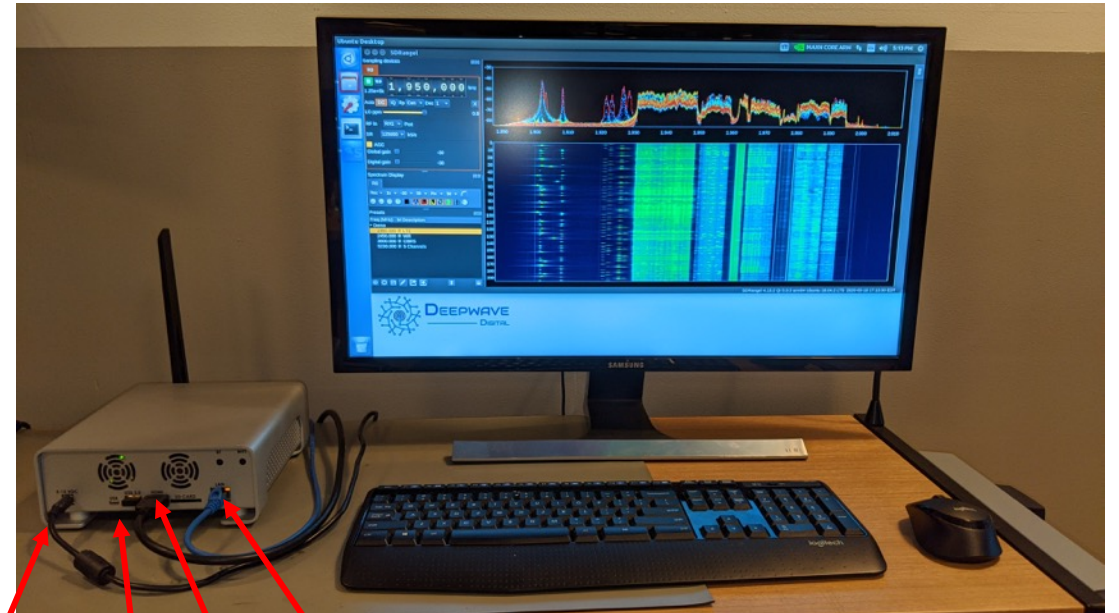
AIR-T Demo Setup Front



Power

TX, RX, Sync, Reference, LO, etc

AIR-T Demo Setup Back



Power

USB Keyboard/Mouse

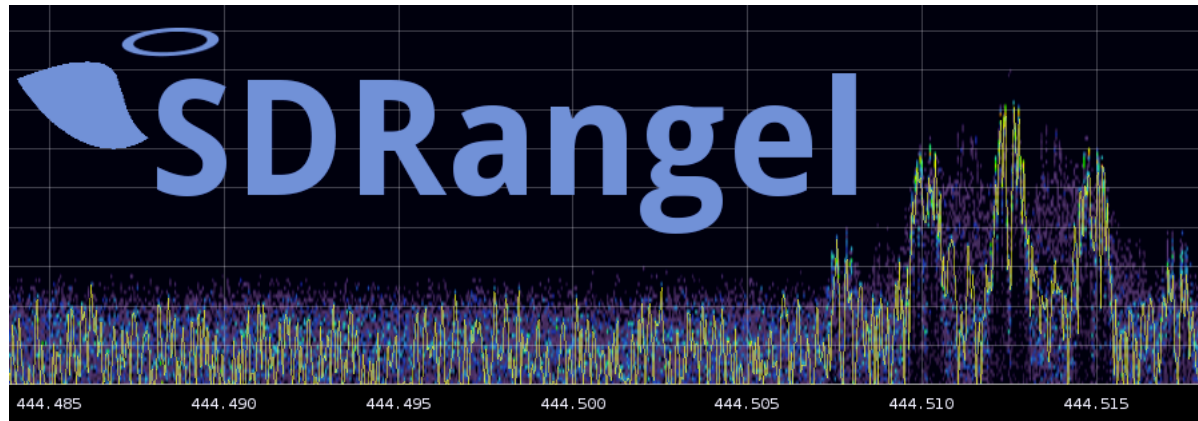
HDMI

Ethernet



# AIR-T Demonstration 1

SDRangel Open Source Software



**DEEPWAVE**  
DIGITAL

See Video for Demonstration:

<https://deepwavedigital.com/blog/deepwave-digital-webinar-march-25-2020>

# AirStack Radio Python API: SoapySDR

## Basic Python Code Example

```
1 import SoapySDR
2 from SoapySDR import SOAPY_SDR_RX, SOAPY_SDR_CS16
3 import numpy as np
4
5 rx_chan = 0          # RX1 = 0, RX2 = 1
6 N = 16384            # Number of complex samples per transfer
7 fs = 125e6           # Radio sample Rate
8 freq = 750e6         # LO tuning frequency in Hz
9 use_agc = True       # Use or don't use the AGC
10
11 # Initialize the AIR-T receiver using SoapyAIRT
12 sdr = SoapySDR.Device(dict(driver="SoapyAIRT")) # Create AIR-T instance
13 sdr.setSampleRate(SOAPY_SDR_RX, rx_chan, fs)   # Set sample rate
14 sdr.setGainMode(SOAPY_SDR_RX, rx_chan, use_agc) # Set the gain mode
15 sdr.setFrequency(SOAPY_SDR_RX, rx_chan, freq)  # Tune the frequency
16
17 # Create receiver buffer
18 buff = np.empty(N, np.complex64)
19
20 # Turn on radio
21 rx_stream = sdr.setupStream(SOAPY_SDR_RX, SOAPY_SDR_CS16, [rx_chan])
22 sdr.activateStream(rx_stream)
23 # Continuously read signal data from radio
24 while True:
25     try:
26         sr = sdr.readStream(rx_stream, [buff], N)
27         """ Insert application code here """
28     except KeyboardInterrupt:
29         """ Insert cleanup code here """
30         break
31 sdr.deactivateStream(rx_stream)
32 sdr.closeStream(rx_stream)
```

Import Python Packages

Define the parameters for the radio:

- Frequency, sample rate, gain, etc.

Initialize the radio class:

- AirStack driver is *SoapyAIRT*
- Set frequency, sample rate, gain, etc.

Create shared memory buffer for reading samples

Turn on the radio and start streaming data to buffer

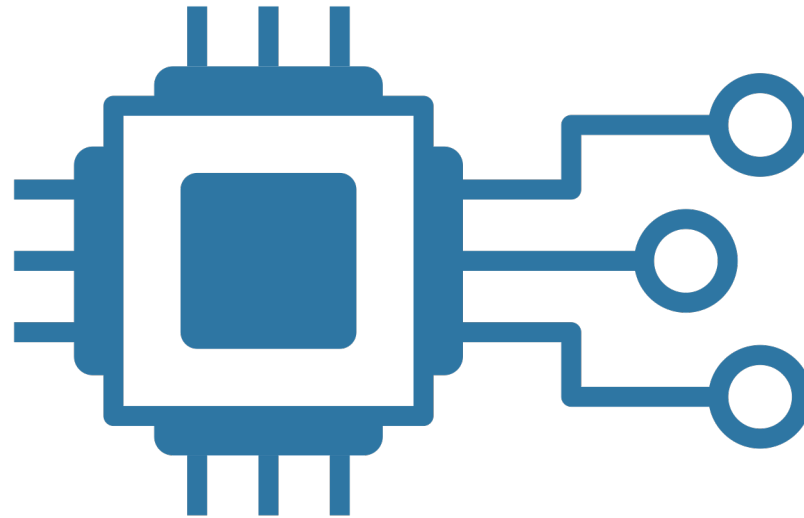
Continuously read data from radio:

- Signal processing application happens each loop
- Leverage numpy, scipy, cusignal, CUDA, etc.

Turn off the radio

# AIR-T Demonstration 2

AirStack Radio Drivers



See Video for Demonstration:

<https://deepwavedigital.com/blog/deepwave-digital-webinar-march-25-2020>



# Outline

- Introduction
- AIR-T System
- AirStack Overview
- Signal Processing Demonstrations
  - ➔ • GNU Radio
  - cuSignal
- Deep Learning Workflow
- Summary



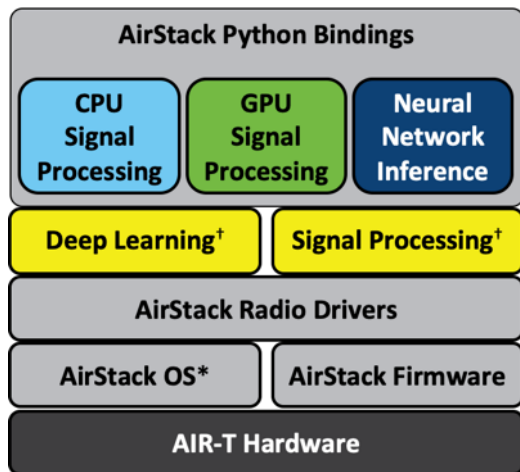
# Artificial Intelligence Radio Transceiver (AIR-T)

Deepwave's AirStack Application Programming Interface (API)

 Discussed Today

 Future Webinar

## AirStack Full Stack



## AirStack Python Programming Interfaces

CPU Signal Processing

GPU Signal Processing

Neural Network Inference

 `scipy.signal`

`numpy`

 `GNU Radio`

 `cuSignal`

Custom  
CUDA  
Kernels

`cupy`

PyCUDA,  
numba

GR-Wavelearner

 `TensorRT`

CUDA

CUDA / cuDNN

ARM / Denver CPU

NVIDIA GPU

NVIDIA GPU

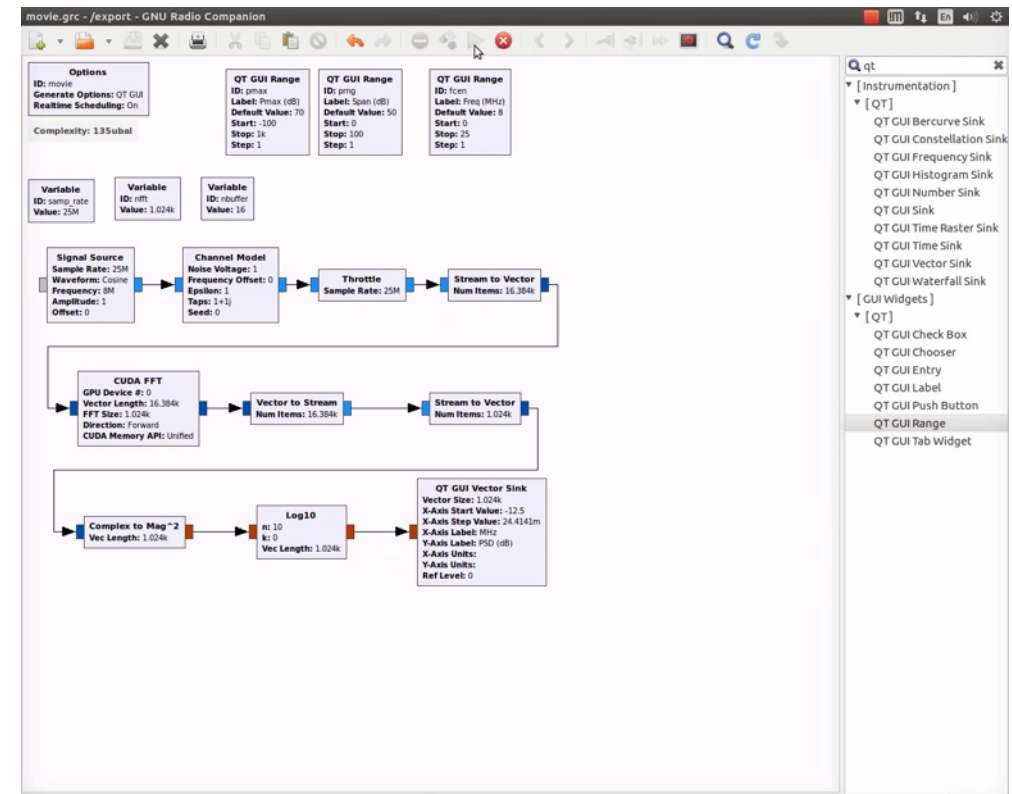


<sup>†</sup> 3<sup>rd</sup> Party APIs

\* Operating system based on NVIDIA Jetpack

# GNU Radio – Software Defined Radio (SDR) Framework

- Popular open source software defined radio (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 is integrating GPU support for both DSP and ML

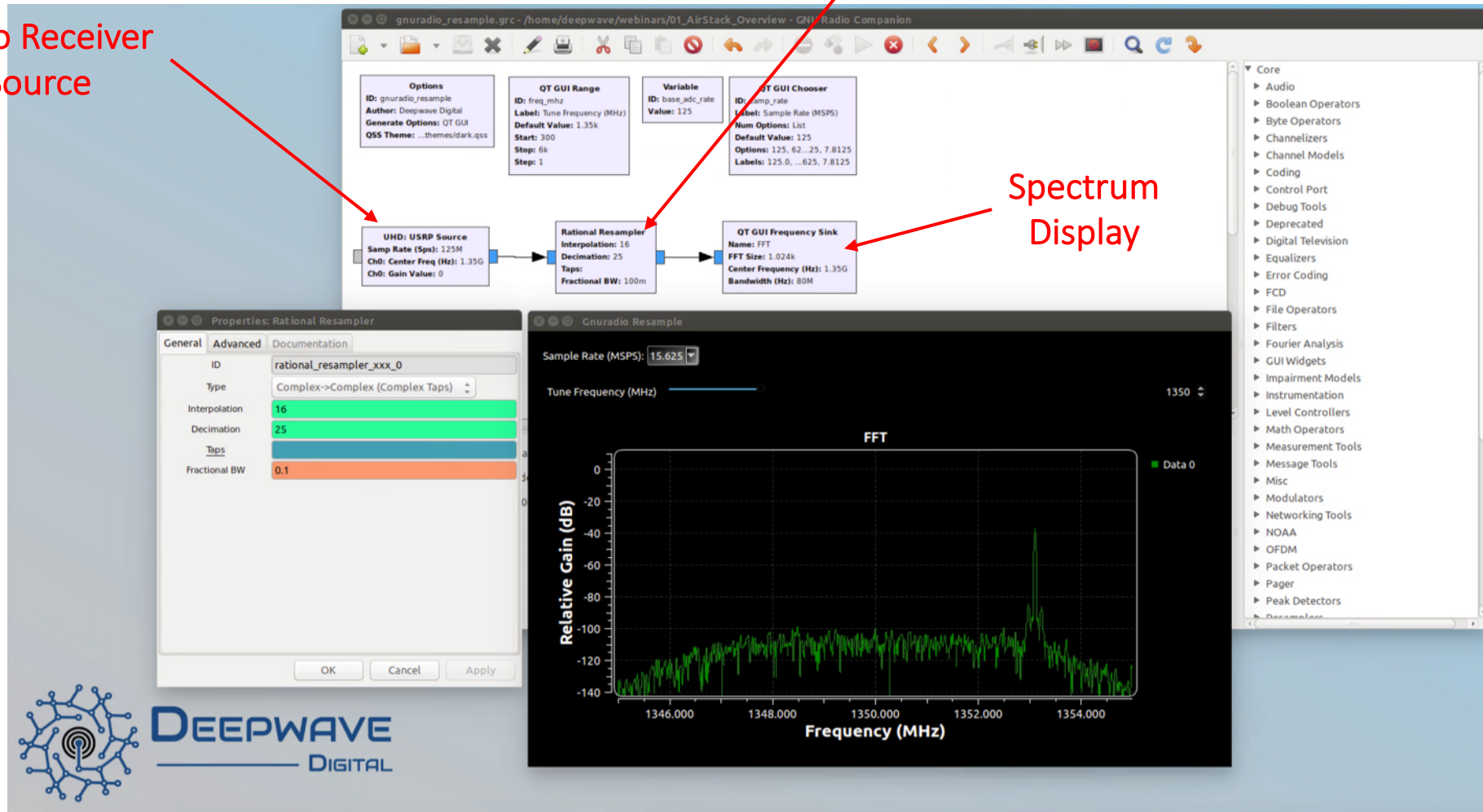


# Polyphase Resample Filter with GNU Radio

Radio Receiver  
Source

Polyphase Filter

Spectrum  
Display



# AIR-T Demonstration 3

Polyphase Filter in GNU Radio on the AIR-T

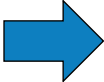


**DEEPWAVE**  
DIGITAL

See Video for Demonstration:

<https://deepwavedigital.com/blog/deepwave-digital-webinar-march-25-2020>

# Outline

- Introduction
- AIR-T System
- AirStack Overview
- Signal Processing Demonstrations
  - GNU Radio
  -  • cuSignal
- Deep Learning Workflow
- Summary

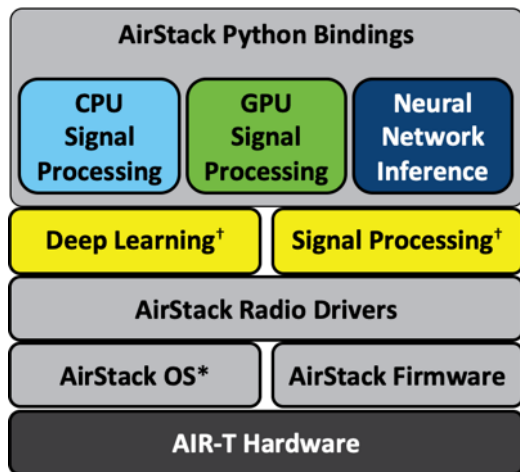
# Artificial Intelligence Radio Transceiver (AIR-T)

Deepwave's AirStack Application Programming Interface (API)

 Discussed Today

 Future Webinar

## AirStack Full Stack



## AirStack Python Programming Interfaces

CPU Signal Processing

GPU Signal Processing

Neural Network Inference

 **scipy.signal**

**numpy**

 **GNU Radio**

 **cuSignal**

**Custom  
CUDA  
Kernels**

**cupy**

**PyCUDA,  
numba**

**GR-Wavelearner**

 **TensorRT**

**CUDA**

**CUDA / cuDNN**

**ARM / Denver CPU**

**NVIDIA GPU**

**NVIDIA GPU**



Slides



cuSignal is built as a GPU accelerated version of the popular SciPy Signal library

Most of the coding has leveraged CuPy - GPU accelerated NumPy

In certain cases, we have implemented custom CUDA kernels using Numba - more on this (pros and cons!) later

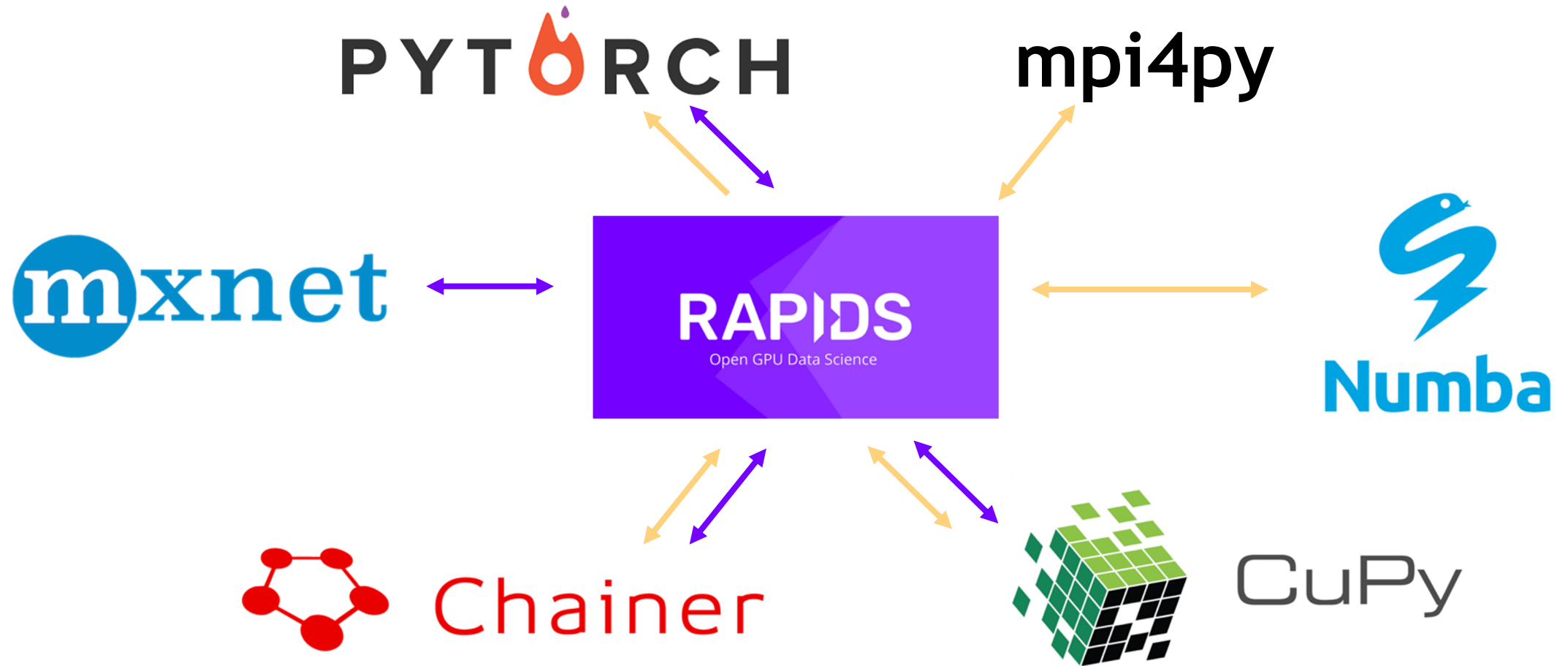
GitHub Repo:

<https://github.com/rapidsai/cusignal>



# Interoperability for the Win

DLPack and `__cuda_array_interface__`





## A NumPy-Compatible Matrix Library Accelerated by CUDA



Free and open source software developed under the Chainer project and Preferred Networks (MIT License)



Includes CUDA libraries: cuBLAS, cuDNN, cuRand, cuSolver, cuSparse, cuFFT, and NCCL



Typically a drop-in replacement for NumPy



Ability to write custom kernel for additional performance, requiring a bit of C++

# HILBERT TRANSFORM: NUMPY ↔ CUPY

```
hilbert_cpu.py x ... hilbert_gpu.py •
C: > Users > adamt > Desktop > hilbert_cpu.py > ...
1 from scipy import fft as sp_fft
2 from numpy import asarray, zeros
3
4 def hilbert(x, N=None, axis=-1):
5     x = asarray(x)
6     if iscomplexobj(x):
7         raise ValueError("x must be real.")
8     if N is None:
9         N = x.shape[axis]
10    if N <= 0:
11        raise ValueError("N must be positive.")
12
13    Xf = sp_fft.fft(x, N, axis=axis)
14    h = zeros(N)
15    if N % 2 == 0:
16        h[0] = h[N // 2] = 1
17        h[1:N // 2] = 2
18    else:
19        h[0] = 1
20        h[1:(N + 1) // 2] = 2
21
22    if x.ndim > 1:
23        ind = [newaxis] * x.ndim
24        ind[axis] = slice(None)
25        h = h[tuple(ind)]
26    x = sp_fft.ifft(Xf * h, axis=axis)
27    return x

C: > Users > adamt > Desktop > hilbert_gpu.py > ...
1 from cupy.scipy import fftpack
2 from cupy import asarray, zeros
3
4 def hilbert(x, N=None, axis=-1):
5     x = asarray(x)
6     if iscomplexobj(x):
7         raise ValueError("x must be real.")
8     if N is None:
9         N = x.shape[axis]
10    if N <= 0:
11        raise ValueError("N must be positive.")
12
13    Xf = fftpack.fft(x, N, axis=axis)
14    h = zeros(N)
15    if N % 2 == 0:
16        h[0] = h[N // 2] = 1
17        h[1:N // 2] = 2
18    else:
19        h[0] = 1
20        h[1:(N + 1) // 2] = 2
21
22    if x.ndim > 1:
23        ind = [newaxis] * x.ndim
24        ind[axis] = slice(None)
25        h = h[tuple(ind)]
26    x = fftpack.ifft(Xf * h, axis=axis)
27    return x
```

# cuSignal On The AIR-T

Leveraging cupy and cuSignal for GPU processing

Numpy/Scipy Implementation (CPU)

Cupy/cuSignal Implementation (GPU)

Import Packages

Define RF Parameters

Create Filter

Initialize Buffers

Start Radio

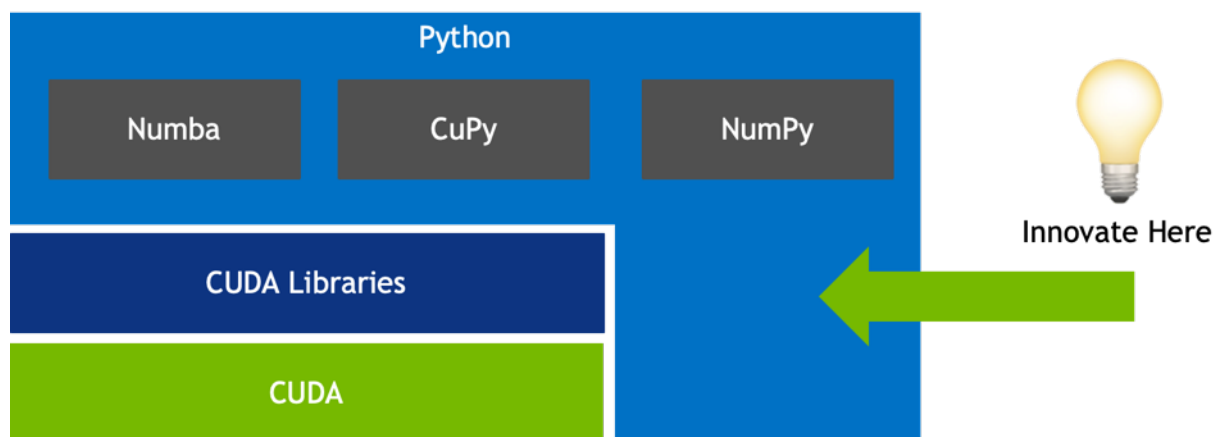
Read Data and  
Execute Application

Print and Plot Result

```
polyphase_cpu.py vs polyphase_gpu.py (/export/software/webinars/01_AirStack_Overview) 5 differences
polyphase_cpu.py (/export/software/webinars/01_AirStack_Overview)
1 import SoapySDR, numpy
2 import scipy.signal as signal
3 import polyphase_plot
4
5 buffer_size = 2**19 # Number of complex samples per transfer
6 t_test = 20 # Test time in seconds
7 freq = 1350e6 # Tune frequency in Hz
8 fs = 62.5e6 / 4 # Sample rate
9
10 # Create polyphase filter
11 fc = 1. / max(16, 25) # cutoff of FIR filter (rel. to Nyquist)
12 nc = 10 * max(16, 25) # reasonable cutoff for our sinc-like function
13 win = signal.fir_filter_design.firwin(2*nc+1, fc, window=('kaiser', 0.5))
14
15 # Init buffer and polyphase filter
16 buff = numpy.empty(buffer_size, dtype=numpy.complex64)
17 s = signal.resample_poly(buff, 16, 25, window=win)
18
19 # Initialize the AIR-T receiver using SoapyAIRT
20 sdr = SoapySDR.Device(dict(driver="SoapyAIRT")) # Create AIR-T instance
21 sdr.setSampleRate(SoapySDR.SOAPY_SDR_RX, 1, fs) # Set sample rate
22 sdr.setGainMode(SoapySDR.SOAPY_SDR_RX, 1, True) # Set the gain mode
23 sdr.setFrequency(SoapySDR.SOAPY_SDR_RX, 1, freq) # Tune the frequency
24 rx_stream = sdr.setupStream(SoapySDR.SOAPY_SDR_RX, SoapySDR.SOAPY_SDR_CF32, [1])
25 sdr.activateStream(rx_stream)
26
27 # Run test
28 n_reads = int(t_test * fs / buffer_size) + 1
29 drop_count = 0
30 for _ in range(n_reads):
31     sr = sdr.readStream(rx_stream, [buff], buffer_size)
32     if sr.ret == -4:
33         drop_count += 1
34     s = signal.resample_poly(buff, 16, 25, window=win)
35 sdr.deactivateStream(rx_stream)
36 sdr.closeStream(rx_stream)
37 gbps = n_reads * len(buff) * buff.itemsize * 8 / (2**30) / t_test
38 msg = 'Dropped Data {} Times in {:.1f} seconds at {:.1f} Gbps on GPU'
39 print(msg.format(drop_count, t_test, gbps))
40
41 polyphase_plot.psd(buff, s, fs, fs*16/25, freq, freq, title='CPU')
42
polyphase_gpu.py (/export/software/webinars/01_AirStack_Overview)
1 import SoapySDR, cupy
2 import cusignal as signal
3 import polyphase_plot
4
5 buffer_size = 2**19 # Number of complex samples per transfer
6 t_test = 20 # Test time in seconds
7 freq = 1350e6 # Tune frequency in Hz
8 fs = 62.5e6 # Sample rate
9
10 # Create polyphase filter
11 fc = 1. / max(16, 25) # cutoff of FIR filter (rel. to Nyquist)
12 nc = 10 * max(16, 25) # reasonable cutoff for our sinc-like function
13 win = signal.fir_filter_design.firwin(2*nc+1, fc, window=('kaiser', 0.5))
14 win = cupy.asarray(win, dtype=cupy.float32)
15
16 # Init buffer and polyphase filter
17 buff = signal.get_shared_mem(buffer_size, dtype=cupy.complex64)
18 s = signal.resample_poly(buff, 16, 25, window=win, use_numba=False)
19
20 # Initialize the AIR-T receiver using SoapyAIRT
21 sdr = SoapySDR.Device(dict(driver="SoapyAIRT")) # Create AIR-T instance
22 sdr.setSampleRate(SoapySDR.SOAPY_SDR_RX, 1, fs) # Set sample rate
23 sdr.setGainMode(SoapySDR.SOAPY_SDR_RX, 1, True) # Set the gain mode
24 sdr.setFrequency(SoapySDR.SOAPY_SDR_RX, 1, freq) # Tune the frequency
25 rx_stream = sdr.setupStream(SoapySDR.SOAPY_SDR_RX, SoapySDR.SOAPY_SDR_CF32, [1])
26 sdr.activateStream(rx_stream)
27
28 # Run test
29 n_reads = int(t_test * fs / buffer_size) + 1
30 drop_count = 0
31 for _ in range(n_reads):
32     sr = sdr.readStream(rx_stream, [buff], buffer_size)
33     if sr.ret == -4:
34         drop_count += 1
35     s = signal.resample_poly(buff, 16, 25, window=win)
36 sdr.deactivateStream(rx_stream)
37 sdr.closeStream(rx_stream)
38 gbps = n_reads * len(buff) * buff.itemsize * 8 / (2**30) / t_test
39 msg = 'Dropped Data {} Times in {:.1f} seconds at {:.1f} Gbps on GPU'
40 print(msg.format(drop_count, t_test, gbps))
41
42 polyphase_plot.psd(buff, s, fs, fs*16/25, freq, freq, title='GPU')
```

# AIR-T Demonstration 4

NVIDIA RAPIDS and CuSignal

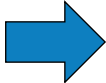


**DEEPWAVE**  
DIGITAL

See Video for Demonstration:

<https://deepwavedigital.com/blog/deepwave-digital-webinar-march-25-2020>

# Outline

- Introduction
- AIR-T System
- AirStack Overview
- Signal Processing Demonstrations
-  • Deep Learning Workflow
- Summary

# Spectrum Monitoring Using Deep Learning on the AIR-T

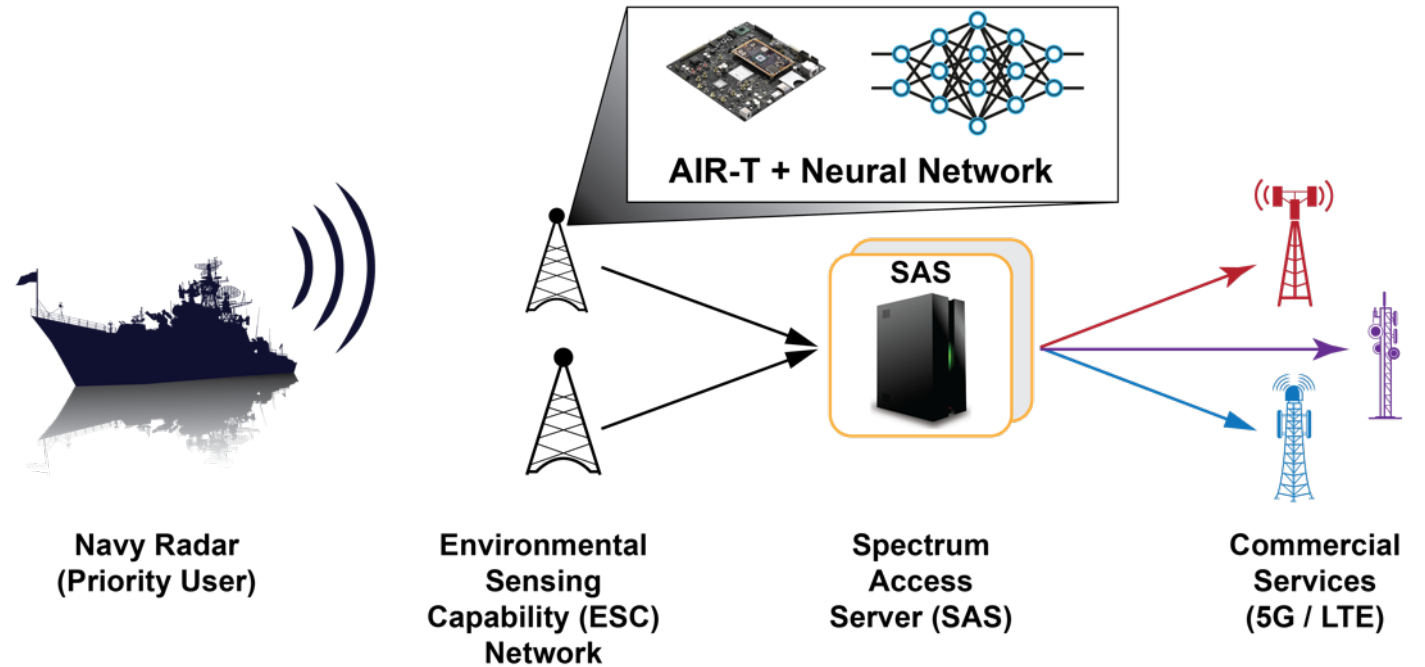
Detecting and Identifying Priority Users for 5G Citizens Broadband Radio Service (CBRS)

Step 1: Monitor spectrum for priority user, i.e., Navy radar

Step 2: Identify emissions using deep neural network

Step 3: Report the presence of a priority user to SAS

Step 4: Distribute commercial services to unused frequencies





# Create, Detect, Label, and Record Data with the AIR-T

Collect Data

Train Network

Optimize

Deploy

## Detect and Label Signals



## Signal Generator Control

- Ethernet, USB, GPIO
- AIR-T defines waveform, modulation, power, frequency, SNR, etc.

## Arbitrary Signal Generator



HDMI



## RF Signal to Record

- Known parameters because AIR-T programmed signal generator
- Recorded data is easily labeled

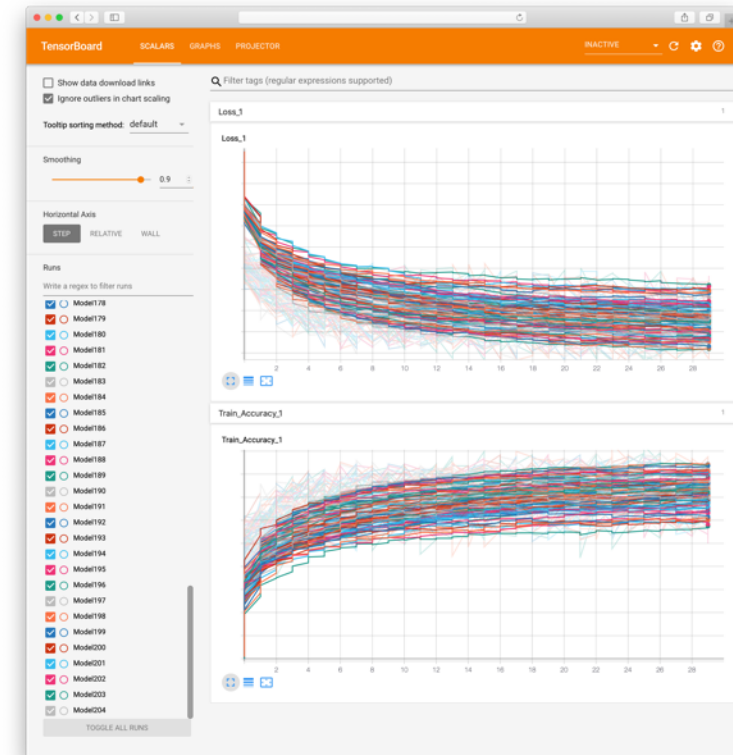
The AIR-T runs **ubuntu**

Easily command and control test equipment directly from AIR-T

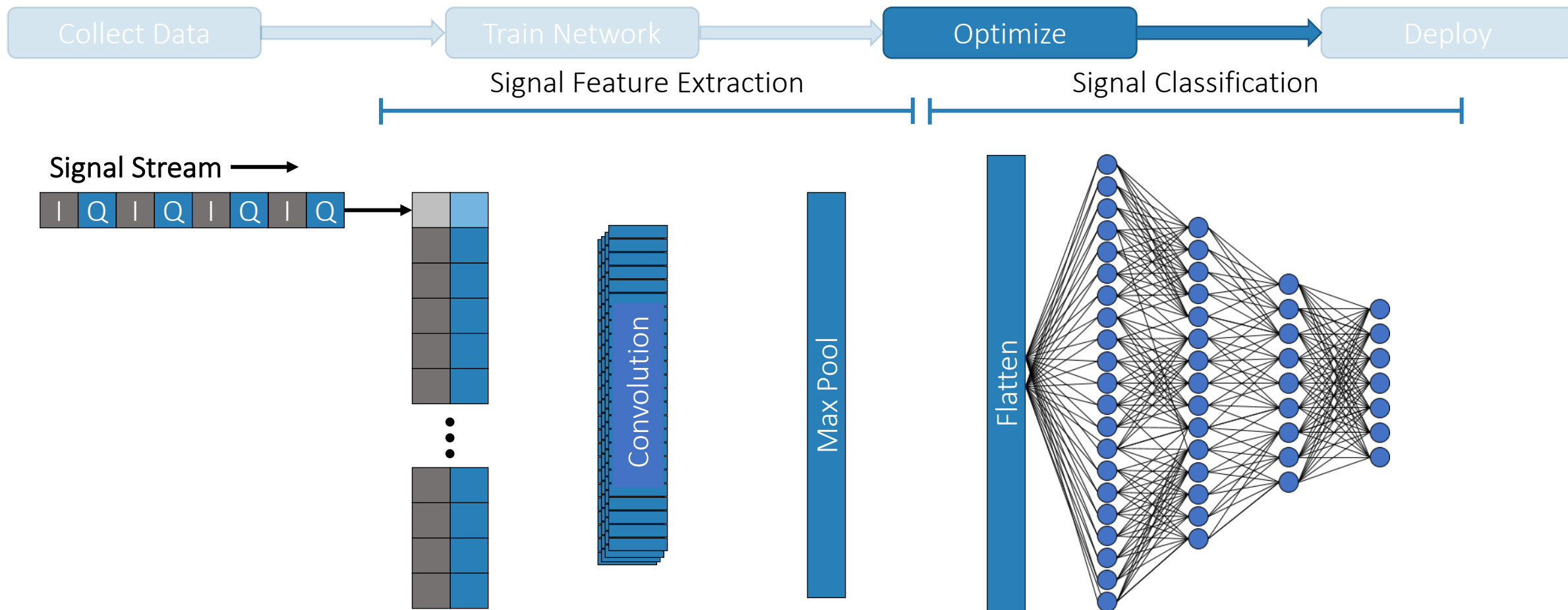
# Train the Neural Network



- Create neural networks in any deep learning framework
- Train many neural networks and freeze best model



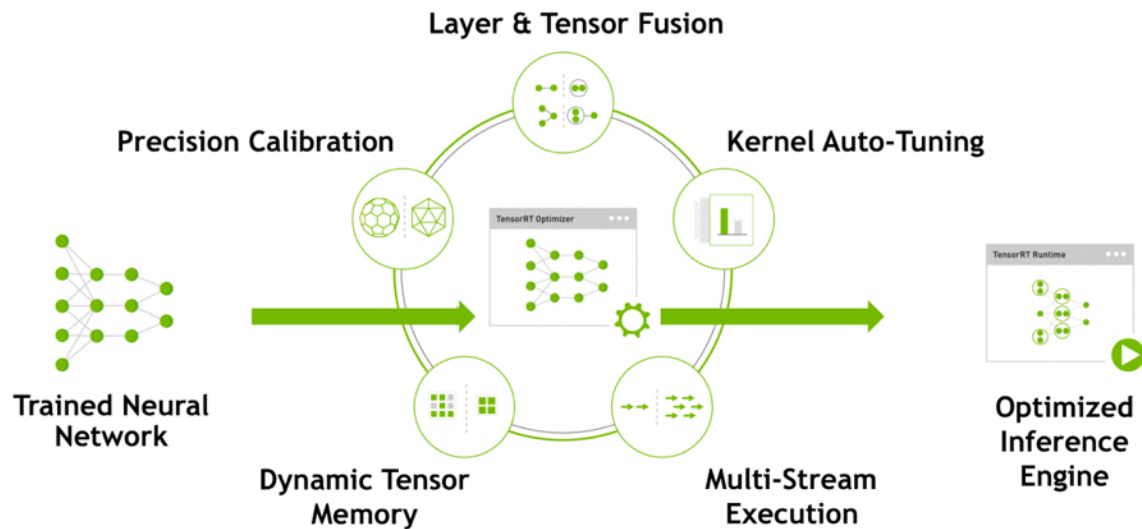
# Radar Signal Detector Model: Example Classifier



# Optimize Neural Network and Prepare for Deployment



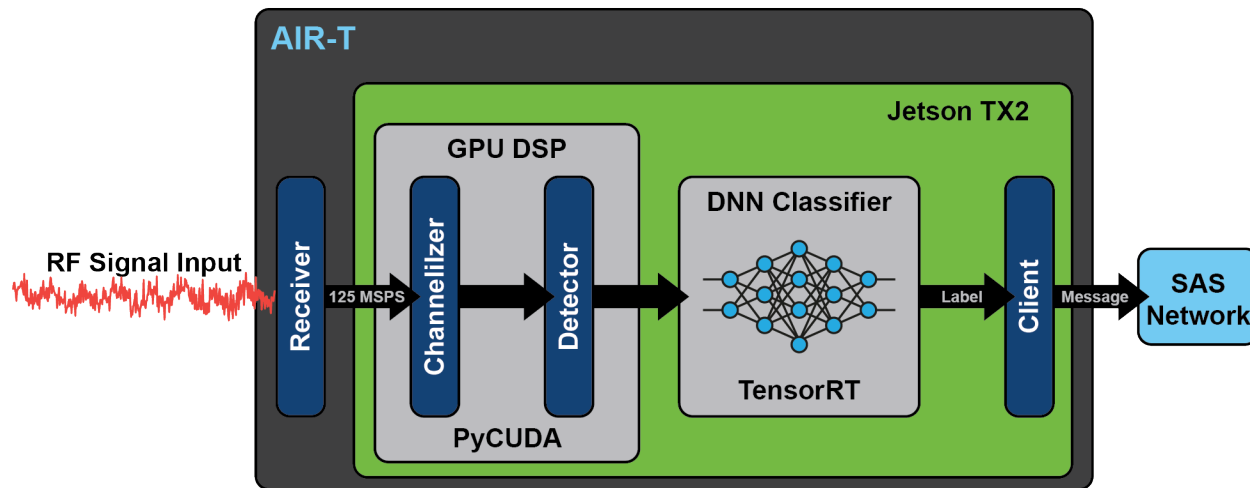
## Use NVIDIA's TensorRT for Optimization



- Unnecessary computations removed to improve efficiency
- Source code provided with many examples
  - Python interface for simplistic programming
- Resulting neural network deployable on AI-RT for real-time inference

# Spectrum Monitoring Using Deep Learning on the AIR-T

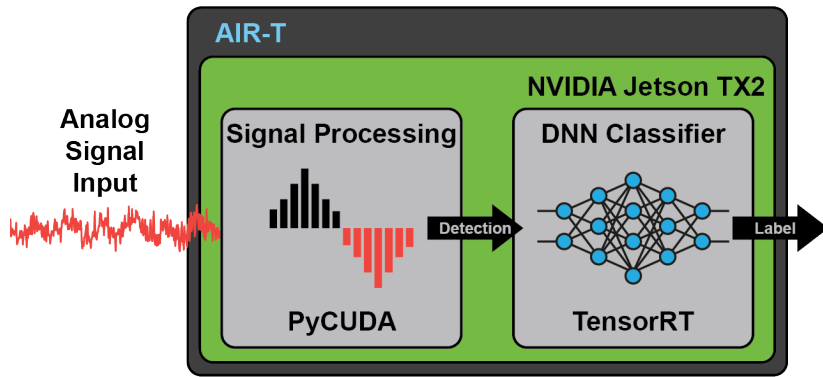
## Deploy for Inference



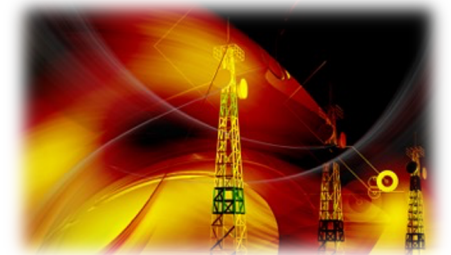
- AIR-T GPU utilization:
  - 85% includes both signal processing and inference tasks
- AIR-T CPU utilization:
  - 40% of one ARM core for processing
  - 30% of one ARM core for network I/O, data logging, and other management tasks

- Sensor has >99% detection rate with >99% classification accuracy
- First known certification of deep neural networks in the telecommunications industry for signal processing
- Deepwave's Artificial Intelligence Radio Transceiver (AIR-T) used for system development and deployment

# Commercial Signal Classifier For Defense Applications



- Deepwave's proven signal classifier sensor detects and classifies various signal types with extreme accuracy
- Embedded solution runs real-time in a low SWaP form factor
- Signal classifier may be easily updated for additional signal types and environments
- Capable of detecting and classifying signals in deployed environment at low SNR



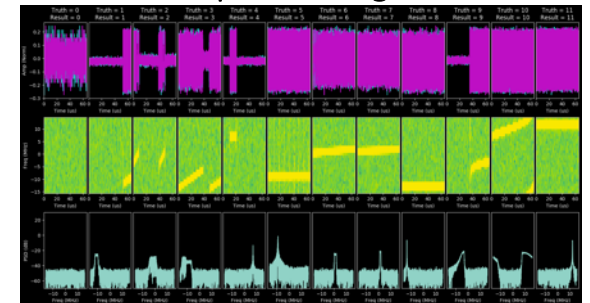


# Upcoming Webinar

Date is soon to be announced

- In this webinar we covered signal processing on the AIR-T and GPUs which is a great source for pre-processing of data for deep learning.
- Our next webinar will provide a deeper dive into deep learning, inference, and go over our AirPack product.
  - AirPack contains everything you need, including source code, to walk you through the crucial steps of training a simple convolutional neural network (CNN) to detect and classify radio frequency (RF) signals.
  - More information on AirPack at [www.deepwavedigital.com/airpack](http://www.deepwavedigital.com/airpack)
  - Date to be announced shortly

Example Training Data



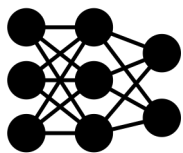
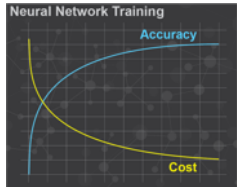


# WHAT WE DO

## Product Line

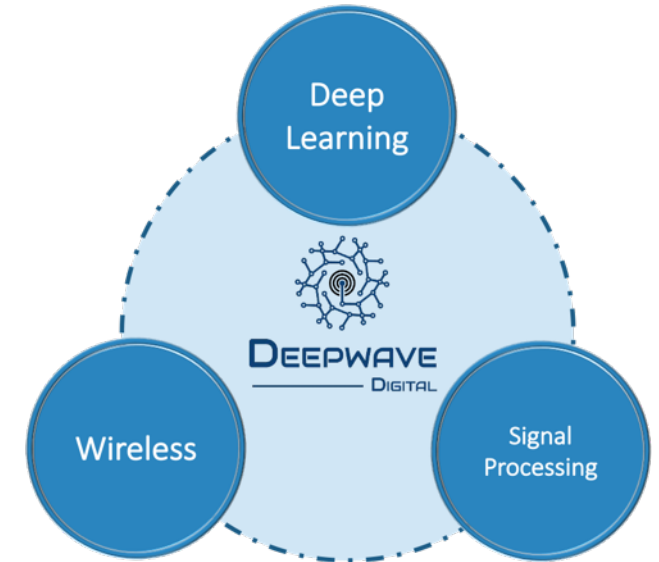


- **AIR-T** – Deep learning enabled software defined radio
  - Two models, one with upgraded FPGA
  - Rugged version coming soon
  - AirStack development environment – Python based programming for simplistic development
- **AirPack** – Comprehensive software solution for creating AI enabled signal processing applications
  - Training data, neural network, and all source code
  - Deployable for real-time inference on the AIR-T



## Neural Network Design Services

- Signal detection and identification
- Electronic protection and interference mitigation



## HEADQUARTERS

Founded in 2017 in Philadelphia, PA

## Contact Us:

[info@deepwavedigital.com](mailto:info@deepwavedigital.com)  
[www.deepwavedigital.com/inquiry](http://www.deepwavedigital.com/inquiry)

# Open Source References and Credits

Libraries directly leveraged for this presentation

Open Source Library	URL	License
Deepwave Repo	<a href="https://github.com/deepwavedigital">https://github.com/deepwavedigital</a>	Multiple
SoapySDR	<a href="https://github.com/pothosware/SoapySDR/wiki">https://github.com/pothosware/SoapySDR/wiki</a>	BSL-1.0
SDRangel	<a href="https://github.com/f4exb/sdrangel">https://github.com/f4exb/sdrangel</a>	GPL-3.0
RAPIDS	<a href="https://rapids.ai/">https://rapids.ai/</a>	Apache-2.0
cuSignal	<a href="https://github.com/rapidsai/cusignal">https://github.com/rapidsai/cusignal</a>	Apache-2.0
TensorRT	<a href="https://github.com/NVIDIA/TensorRT">https://github.com/NVIDIA/TensorRT</a>	Apache-2.0
CuPy	<a href="https://github.com/cupy/cupy">https://github.com/cupy/cupy</a>	MIT
GNU Radio	<a href="https://www.gnuradio.org/">https://www.gnuradio.org/</a>	GPL-3.0
UHD	<a href="https://github.com/EttusResearch/uhd">https://github.com/EttusResearch/uhd</a>	GPL-3.0
Jetson Stats	<a href="https://github.com/rbonghi/jetson_stats">https://github.com/rbonghi/jetson_stats</a>	AGPL-3.0



**DEEPWAVE**  
— **DIGITAL**

[www.deepwavedigital.com/inquiry](http://www.deepwavedigital.com/inquiry)