DEEPWAVE DIGITAL

End-to-End Signal Processing and Deep Learning (using Embedded GPUs)

Daniel B. Bryant

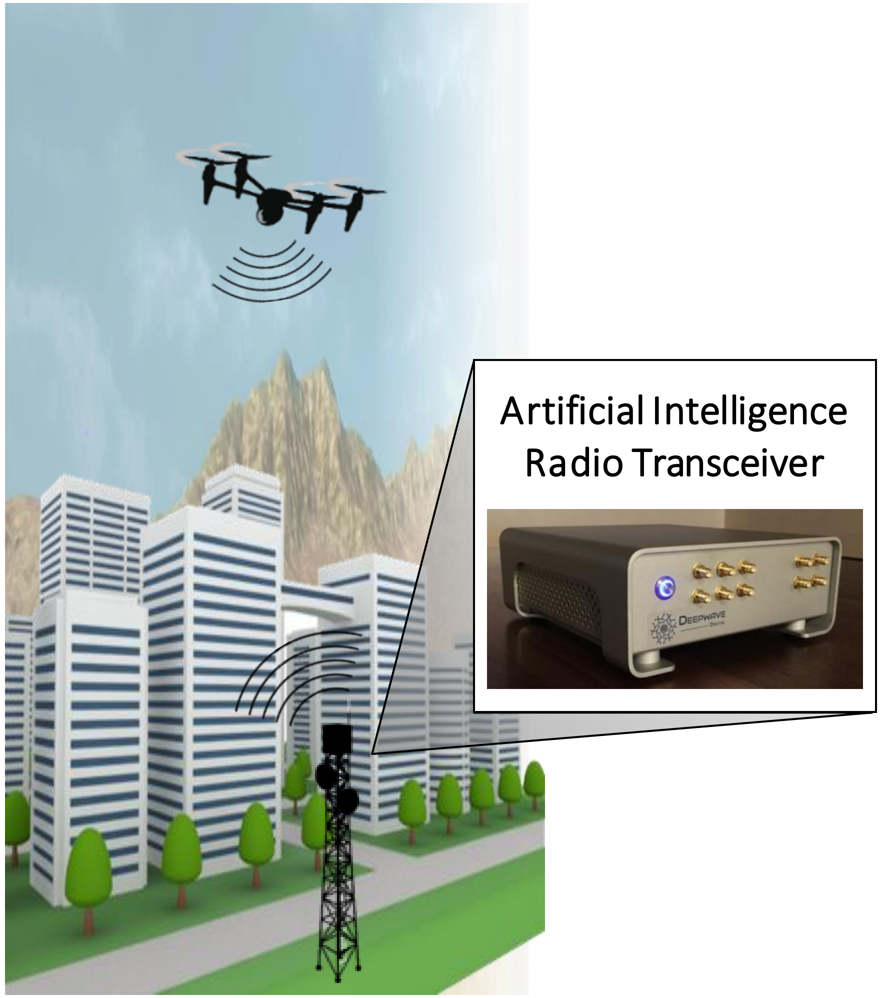
dan@deepwavedigital.com

Deepwave Digital

Enabling the Incorporation of Deep Learning and Radio Frequency (RF) Systems

- **Full stack solutions** for deep learning and GPU enabled signal processing systems
  - Edge compute hardware
  - Custom Applications
  - Tight coupling of hardware and software for performance
    - Radio embedded with FPGA, CPU, GPU
    - GPU-based signal processing algorithms
    - Pruned neural networks for inference on edge RF systems

- **Testing and deployment platform** for customer developed applications
  - AIR-T open platform for custom applications
  - Streamlines development, testing, and deployment
  - Many open source software tools



AI to Solve Complex Problems

Artificial Networks Using Deep Learning

Simple Example: Image Recognition

AI to Solve Complex Problems

Artificial Networks Using Deep Learning

Simple Example: Image Recognition

AI to Solve Complex Problems

Artificial Networks Using Deep Learning

Simple Example: Image Recognition

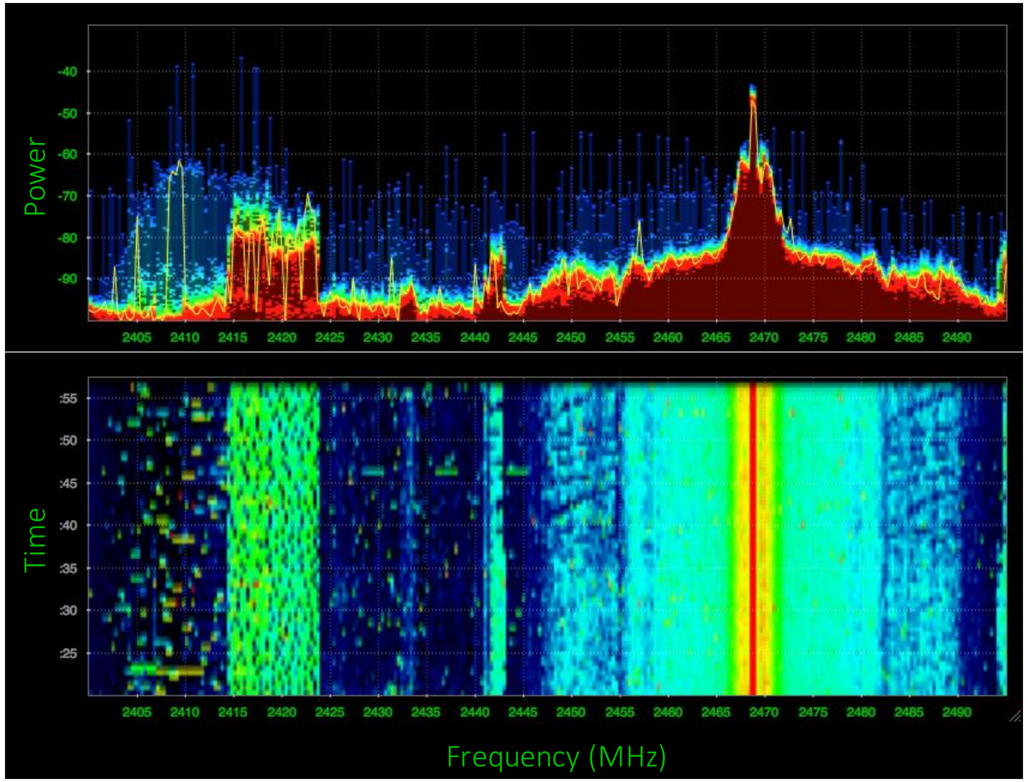
Deep Learning identifies intricate patterns that are too obscure and subtle to be implemented into a human-engineered algorithm

AI to Solve Complex Problems

Artificial Networks Using Deep Learning

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

Deep Learning and Radio Frequency (RF) Systems

Deep Learning is Emerging

Cyber

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

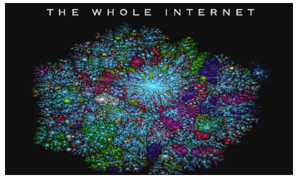
Medicine

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

Autonomy

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

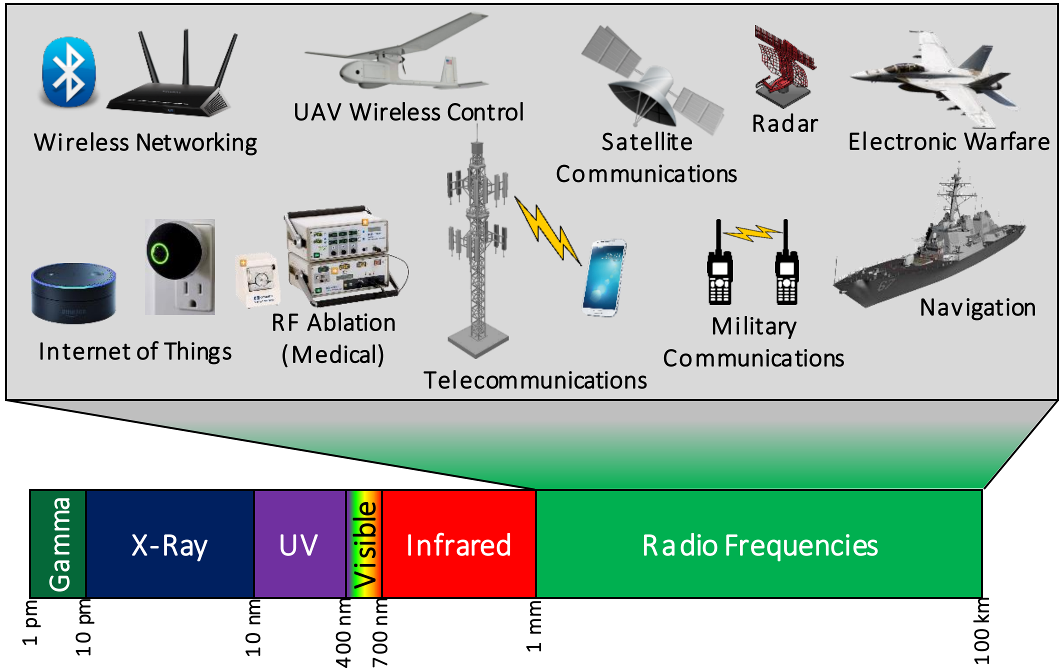
Internet



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

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

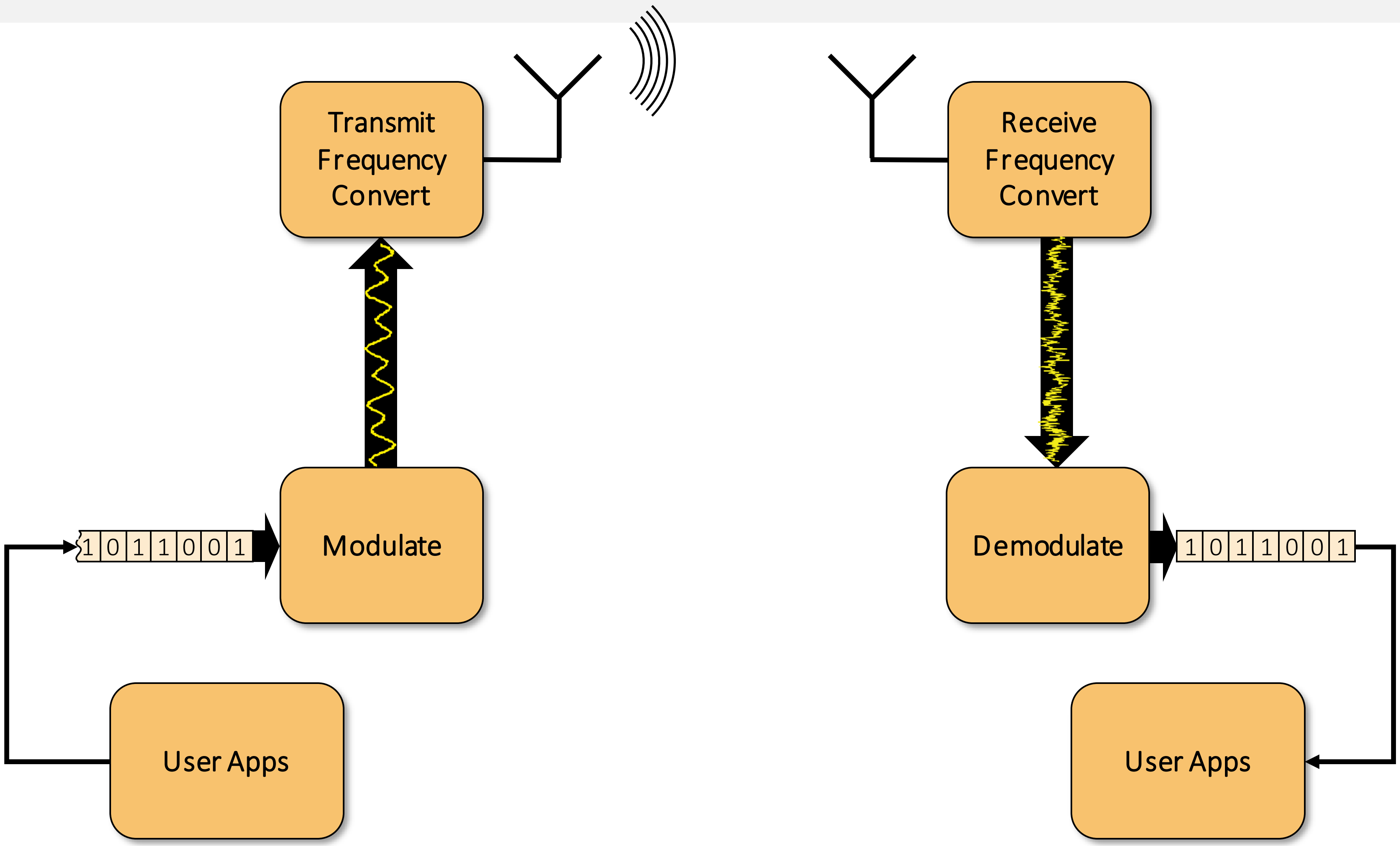
Radio Frequency Technology is Pervasive



Deep learning technology enabled and accelerated by GPU processors

- Only beginning to impact design and applications in wireless and radio frequency systems

Where to Use Deep Learning in RF Systems

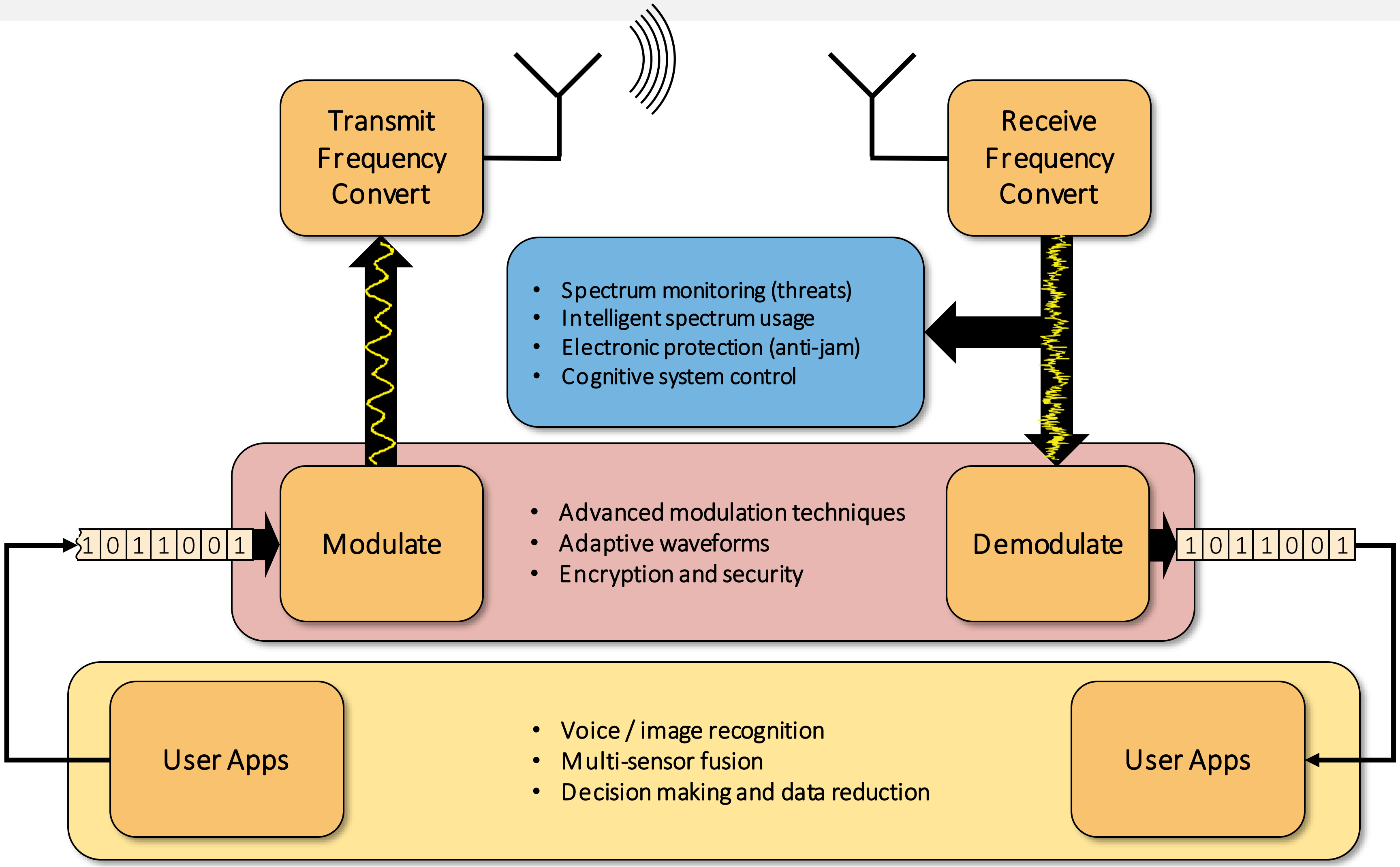


Where to Use Deep Learning in RF Systems

Spectrum / Network
Centric Applications

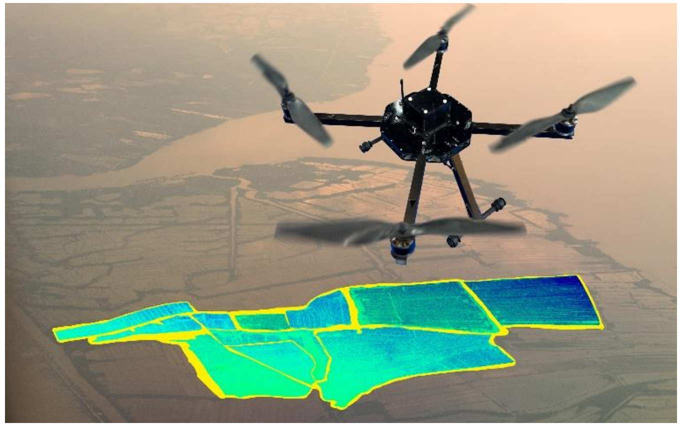
Device / Basestation
Centric Applications

User App
Centric Applications



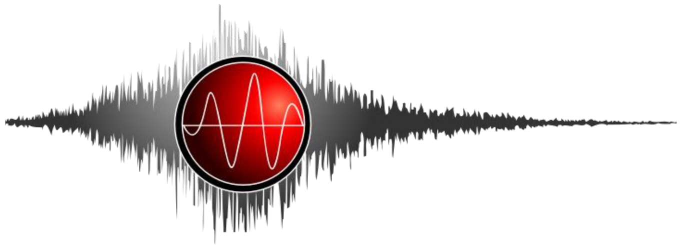
Deep Learning Comparison

Image and Video



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

Audio and Language



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

RF Systems and Signals

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

Existing deep learning potentially adaptable to systems and signals

- Must contend with wideband signals and complex data types

Why Has Deep Learning in RF Not Been Addressed

Backhaul Bandwidth

- Insufficient bandwidth to upload to data center for processing
- Applications are latency sensitive

Edge Compute Resources

- Insufficient resources for AI/RF applications
- No RF agile AI / RF radio systems

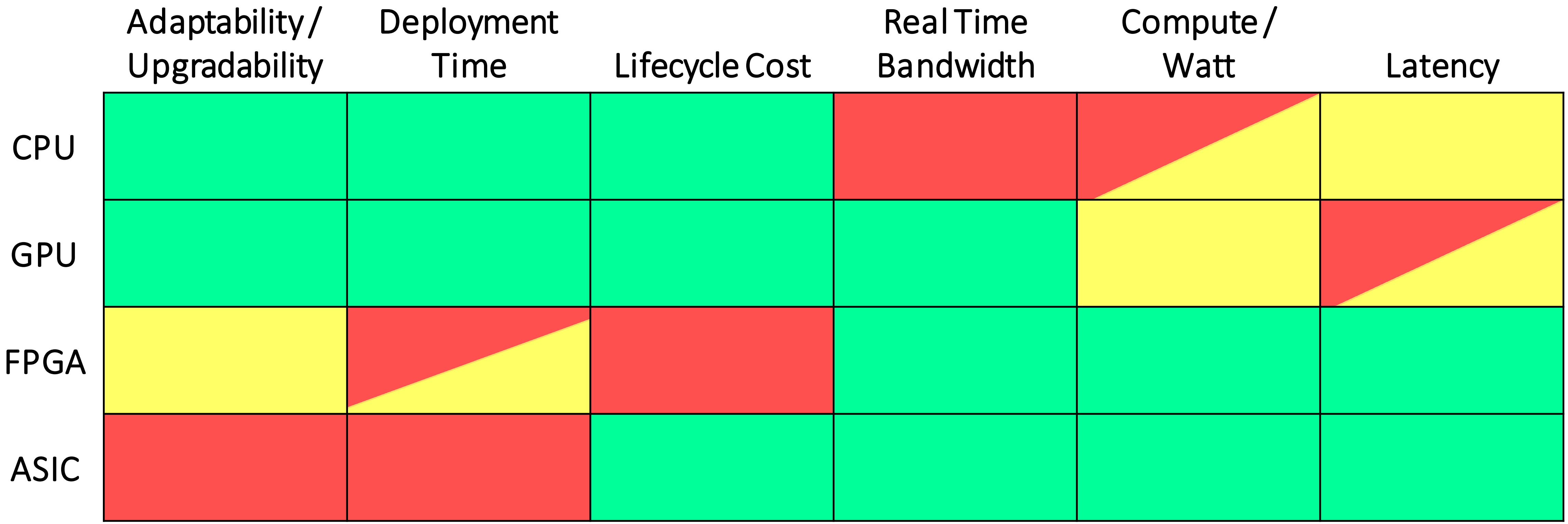
Disjointed software

- Complications of AI / signal processing software merger
- No existing unifying framework

Hardware for Deep Learning in RF Systems

	Training		Inference	
	Pros	**Cons**	**Pros**	**Cons**
CPU	• Supported by ML Frameworks • Lower power consumption	• Slower than GPU • Fewer software architectures	• Adaptable architecture • Software programmable • Medium latency	• Low parallelism • Limited real-time bandwidth • Medium power requirements
GPU	• Supported by ML Frameworks • Widely utilized • Highly parallel / adaptable • Good throughput vs power	• Overall power consumption • Requires highly parallel algorithms	• Adaptable architecture • High real-time bandwidth • Software programmable	• Medium power requirements • Not well integrated into RF • Higher latency
FPGA	Not widely utilized, not well suited (yet)		• High power efficiency • High real-time bandwidth • Low latency	• Long development / upgrades • Limited reprogrammability • Requires special expertise
ASIC	Not widely utilized, not well suited		• Extremely power efficient • High real-time bandwidth • Highly reliable • Low latency	• Extremely expensive • Long development time • No reprogrammability • Requires special expertise

Critical Performance Parameters for Deep Learning in RF Systems

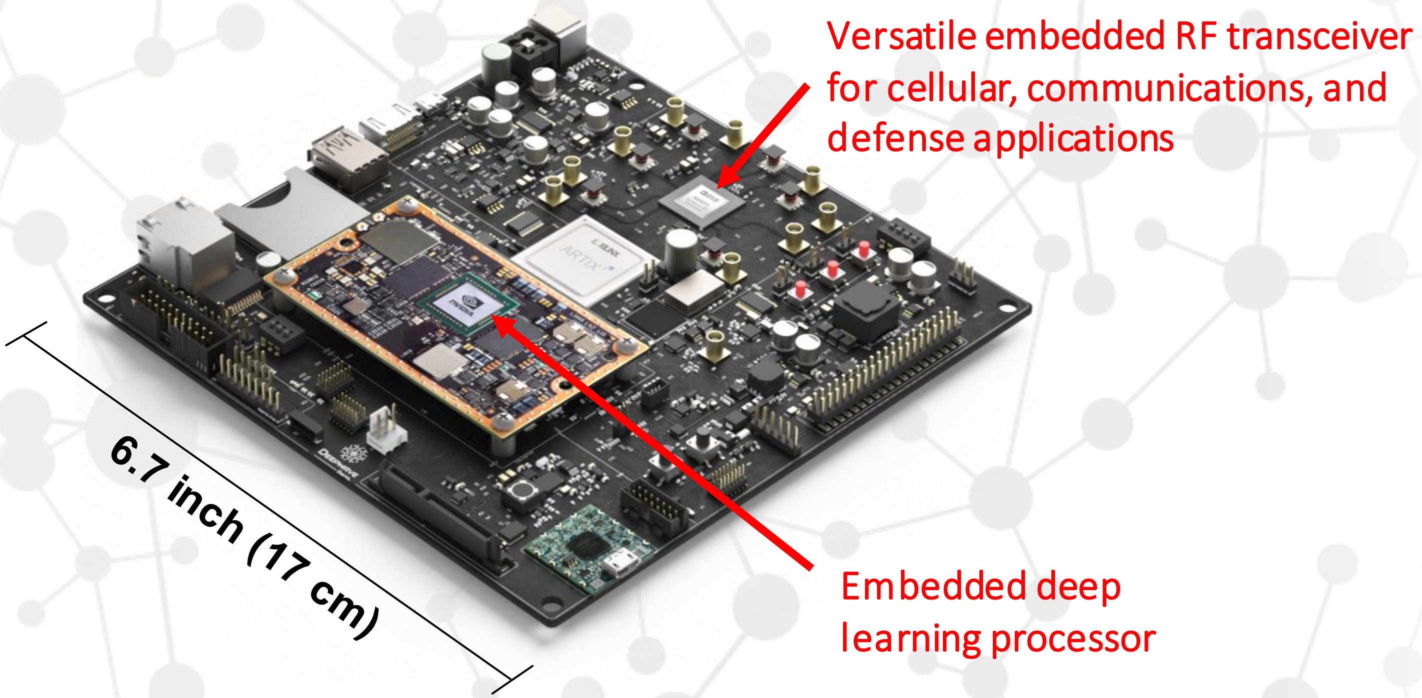


GPU signal processing can provide wideband capability and software upgradability at lower cost and development time

- Must contend with increased latency (~2 microsecond)

Artificial Intelligence Radio Transceiver (AIR-T)

AIR-T Platform



Mini ITX Form Factor

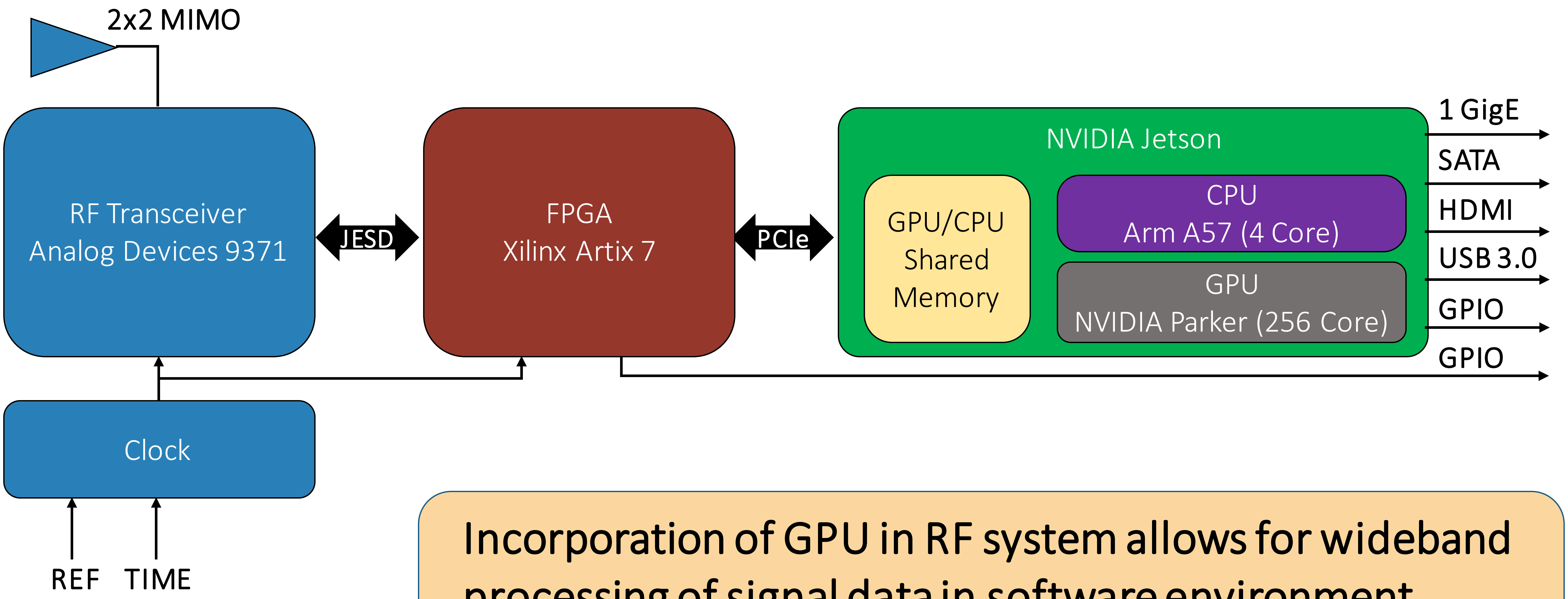
System Specifications

- 2x2 MIMO Transceiver
  - Tunable from 300 MHz to 6 GHz
  - 125 MSPS (100 MHz bandwidth per channel)
- Digital Signal / Deep Learning Processors
  - Xilinx FPGA
  - NVIDIA Jetson TX2
    - 6 CPU cores
    - 256 Core GPU
    - Shared GPU/CPU memory (zero-copy)
- AirStack Software Suite
  - Ubuntu Linux w/ Deepwave hardware drivers
  - All common AI software frameworks supported
  - Python or C++

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

Artificial Intelligence Radio Transceiver (AIR-T)

Block Diagram



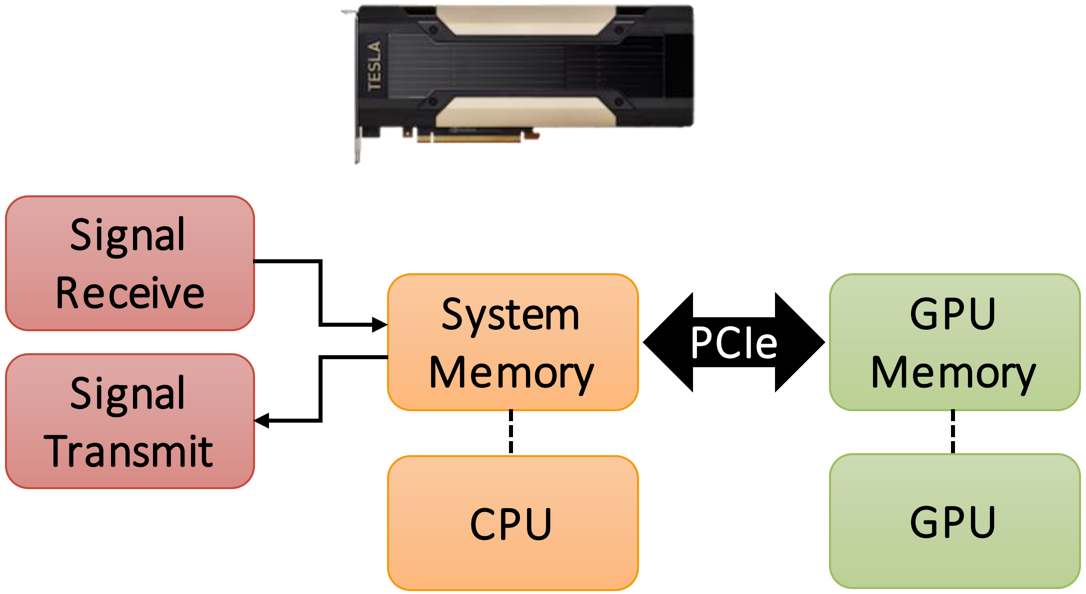
Incorporation of GPU in RF system allows for wideband processing of signal data in software environment

- Reduces development time and cost

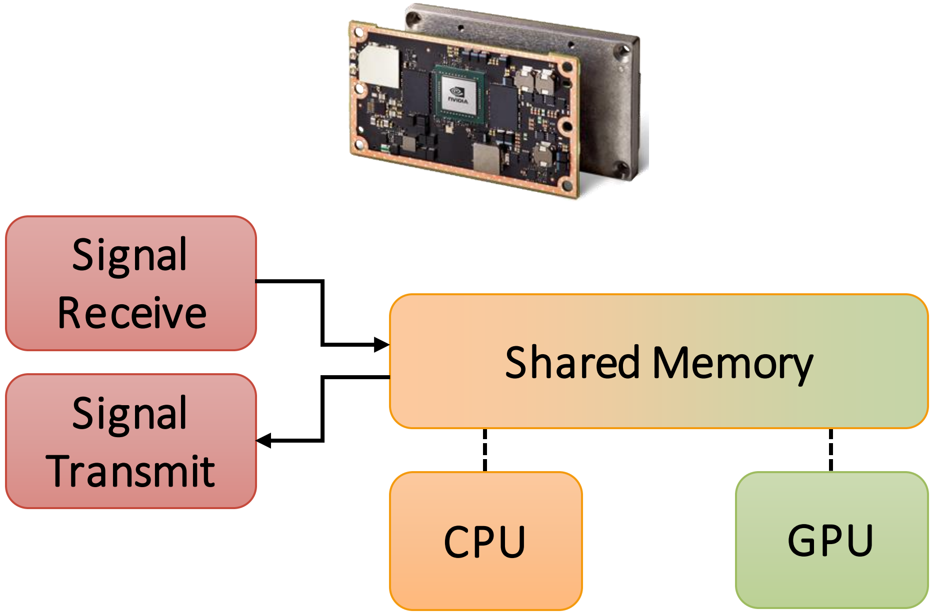
Shared Memory Architecture for Embedded GPUs

Reducing data copies and latency

Traditional GPU: PCIe



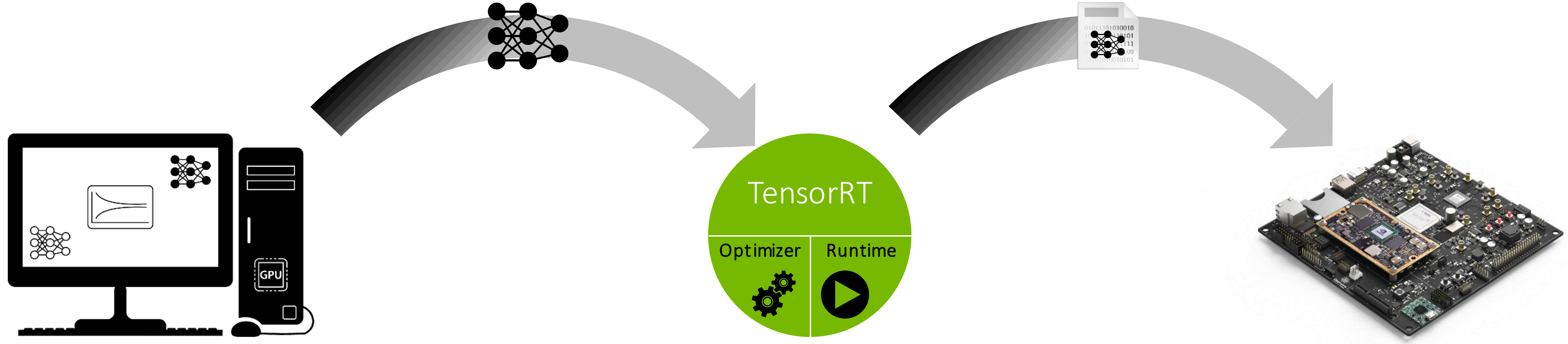
Jetson GPU: Embedded



Jetson Embedded GPUs eliminate extra data copy with GPU/CPU shared memory

- Enables signal processing stream applications with latency driven requirements

Inference at the Edge with AirStack Software



Train Neural Network

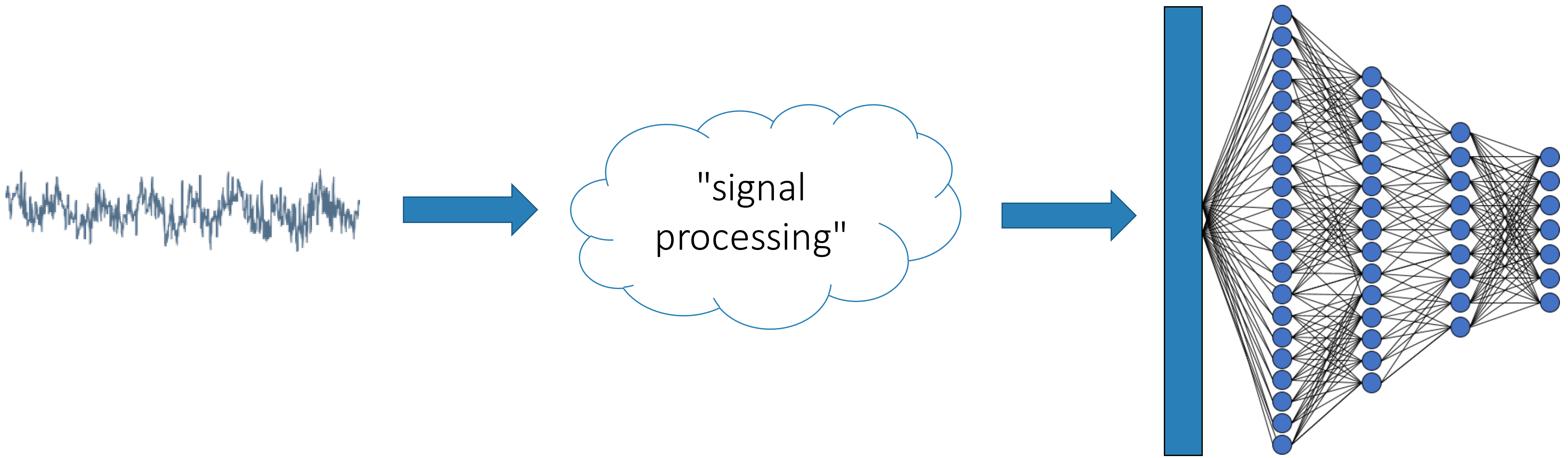
Optimize Neural Network

Deploy Application

Streamlined workflow for deploying deep learning in software defined radio

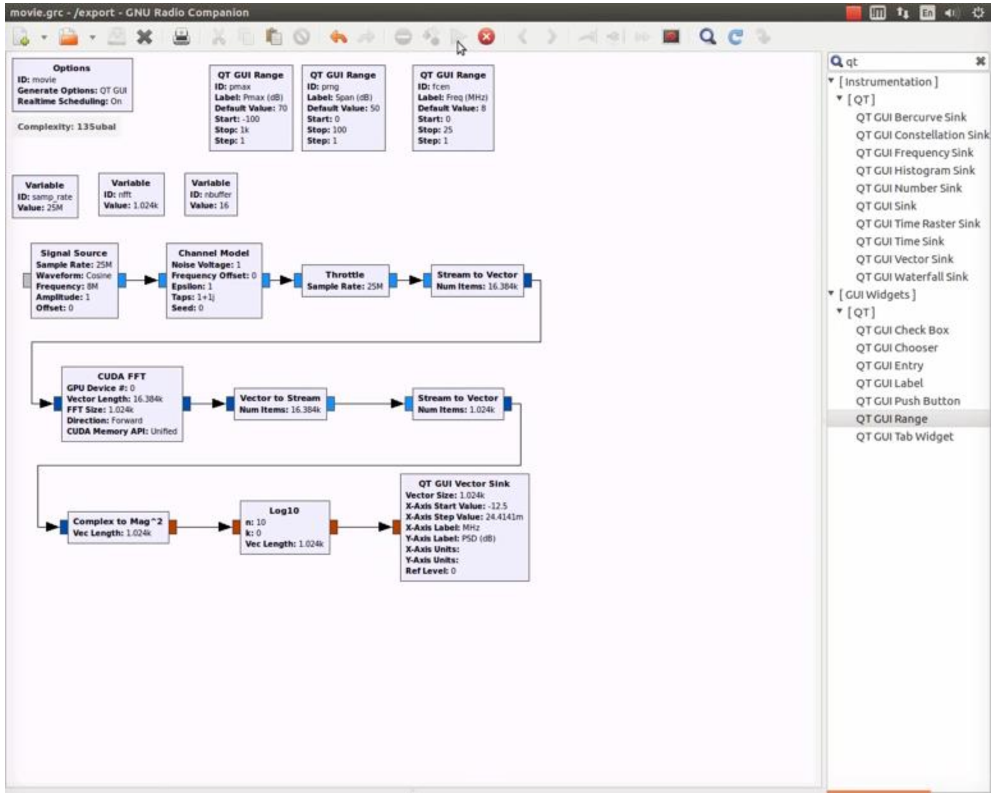
Inference Pipeline for Signals

Signal Input | Magic | Inference Engine

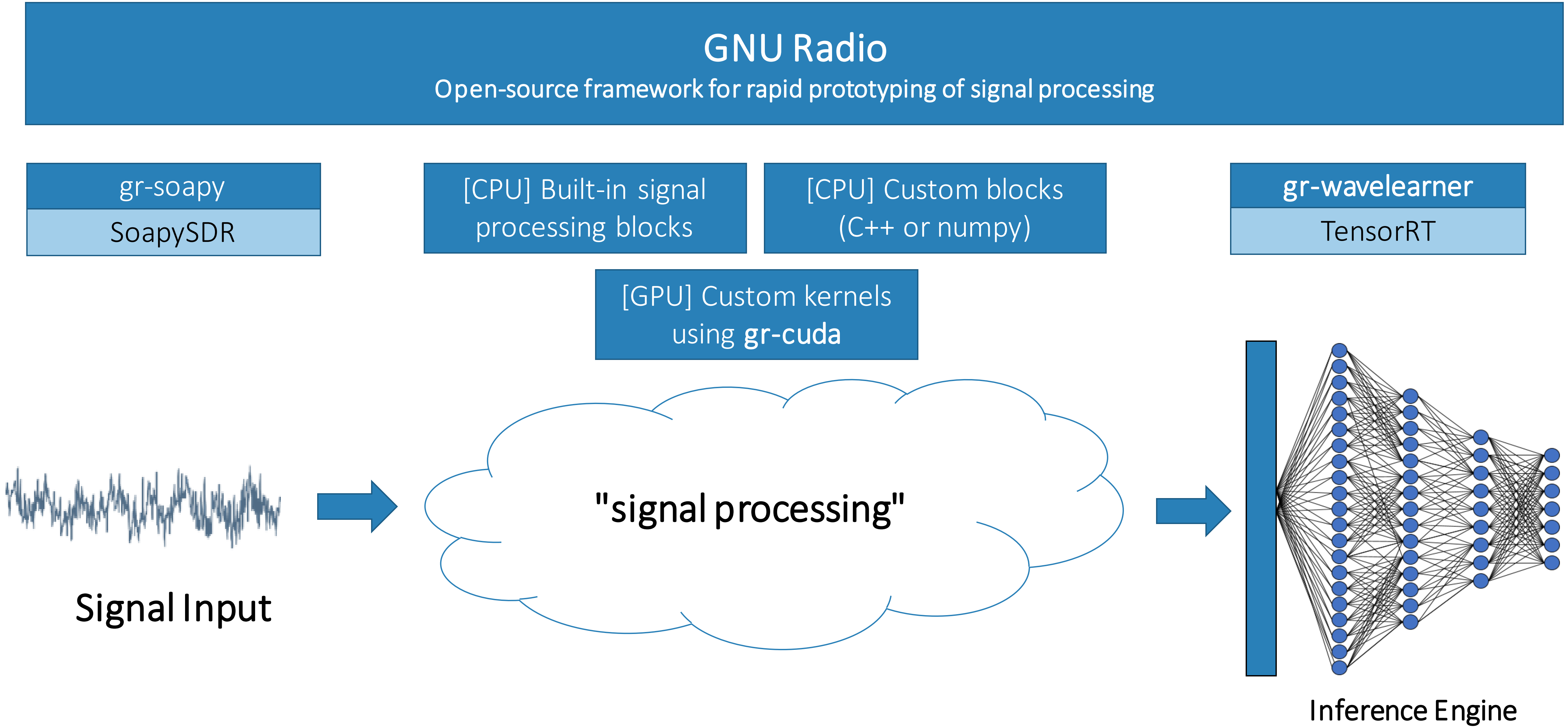


GNU Radio – Software Defined Radio (SDR) Framework

- 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
- Deepwave is integrating GPU support for both DSP and ML



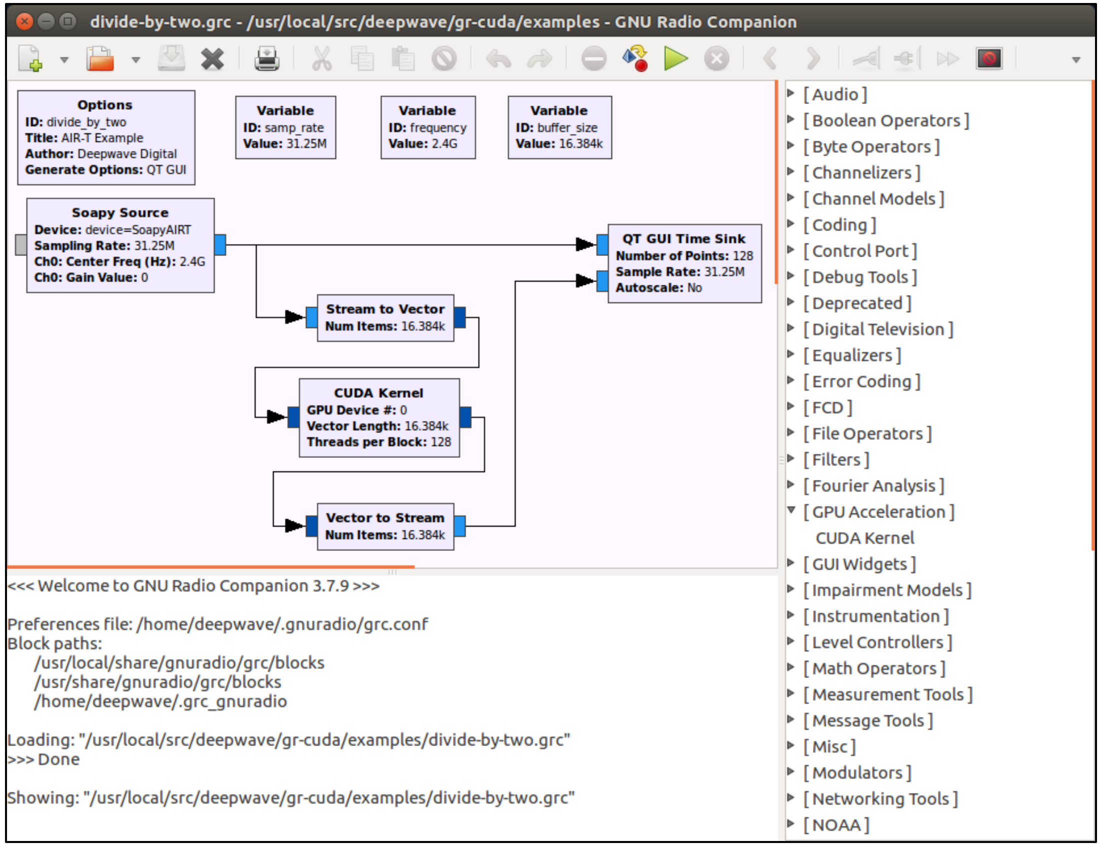
Tying it Together in GNU Radio



GPU Custom Signal Processing: GR-CUDA

Custom GPU Signal Processing with GNU Radio on the AIR-T

- Deepwave provides a simple example for wrapping a custom CUDA kernel with a GNU Radio block
  - Uses pyCUDA under the hood
  - Can place a series of operations into one block with a simple interface
  - Output can be routed to gr-wavelearner for inference
- Source code available on GitHub
- Full tutorial on Deepwave website



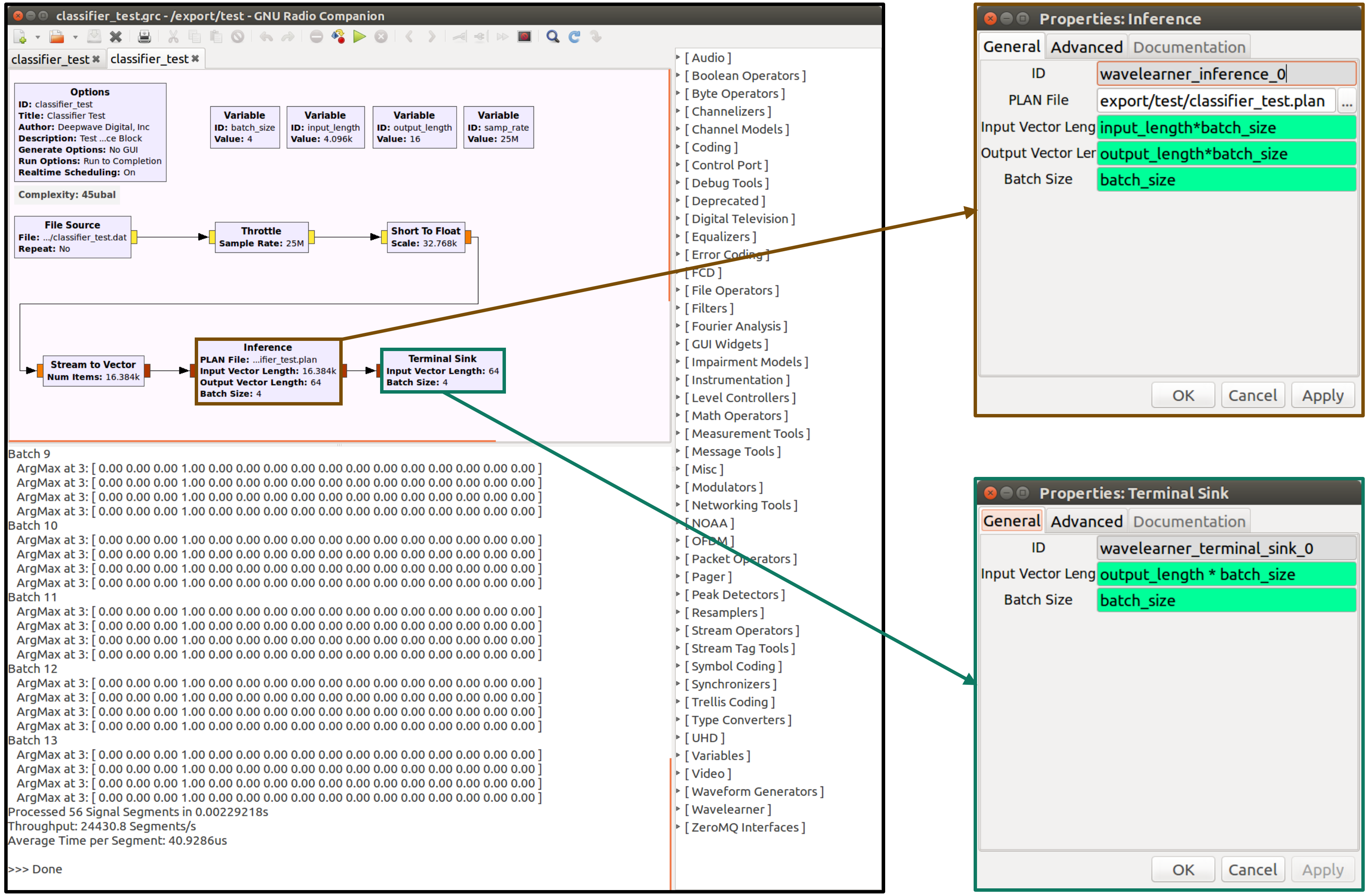
<u>GR-CUDA GitHub</u>: https://github.com/deepwavedigital/gr-cuda

<u>Deepwave Tutorial</u>: https://deepwavedigital.com/tutorials/custom-gpu-signal-processing-with-gnu-radio-on-the-air-t

GR-Wavelearner

Easily Incorporate Inference into Signal Processing

- Three blocks currently:
  - Inference – wraps a serialized TensorRT neural network
  - Terminal Sink – Python module for displaying classifier output
  - FFT – cuFFT wrapper
- Open source module for GNU Radio
- C++ and Python API
- GPLv3 license
- README with instructions to get started quickly



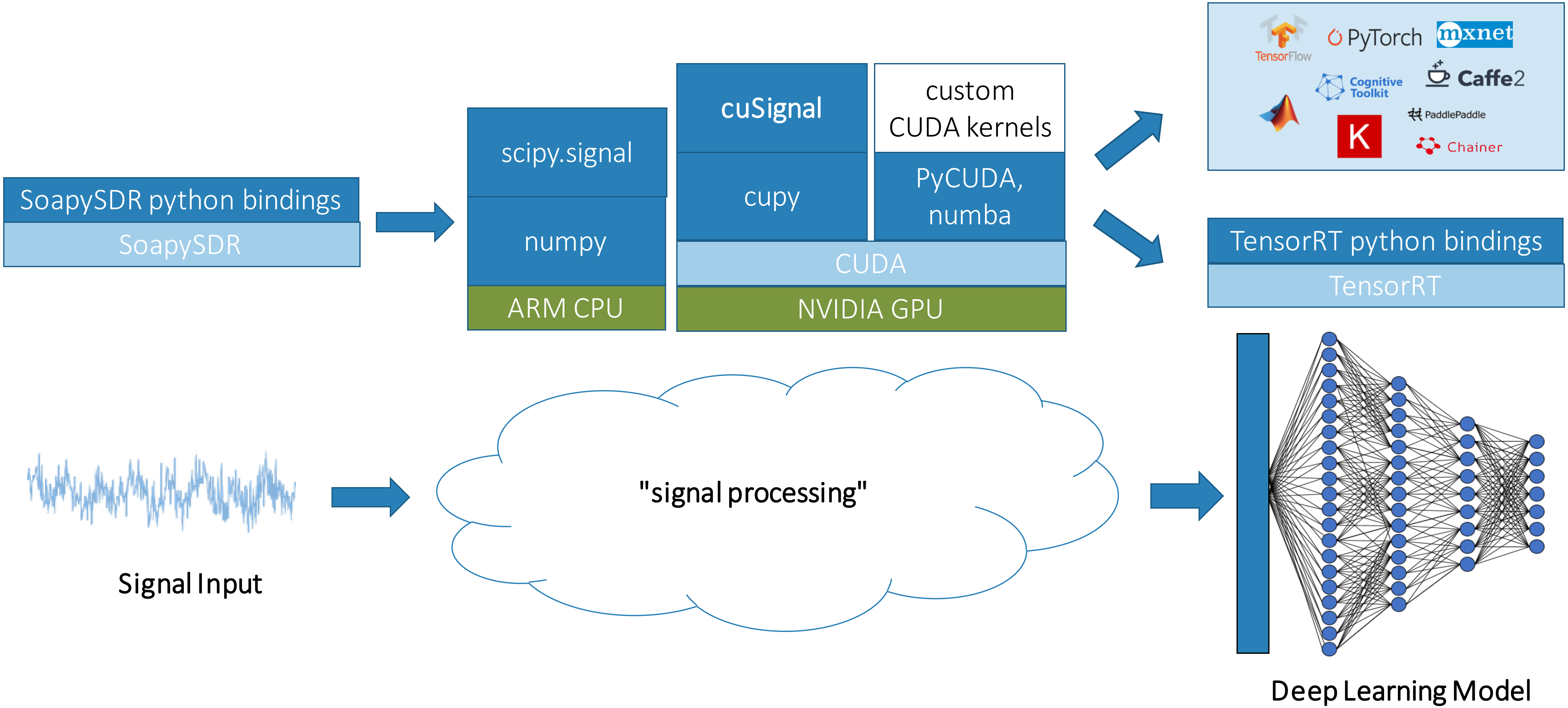
What About Training?

GNU Radio
THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

(some assembly required)

Where do I put my training wheels?

Python Stack for Training and Inference



Example: A Simple Radio Interface

```python
import SoapySDR
from SoapySDR import SOAPY_SDR_RX, SOAPY_SDR_CF32

# Initialize the AIR-T radio using SoapySDR
sdr = SoapySDR.Device(dict(driver="SoapyAIRT"))  # Create AIR-T instance
sdr.setSampleRate(SOAPY_SDR_RX, 0, 125e6)        # Set sample rate
sdr.setGainMode(SOAPY_SDR_RX, 0, automatic=True) # Automatic gain control
sdr.setFrequency(SOAPY_SDR_RX, 0, 2.4e9)         # Tune the radio
# Setup to receive data on channel 0 and turn the radio on
stream = sdr.setupStream(SOAPY_SDR_RX, SOAPY_SDR_CF32, [0])
sdr.activateStream(stream)
```

Example: A Simple Radio Interface

```
import numpy
import numba.cuda

num_samples = 2048
buffer = numba.cuda.mapped_array(num_samples, dtype=numpy.complex64)

while True:
    result = sdr.readStream(stream, [buffer], num_samples, timeoutUs=int(1e6))
    assert result.ret == num_samples

    # ... do stuff with samples in buffer
```

Data is transferred directly from the radio hardware into memory accessible by the GPU

Example: Power Estimation / Energy Detection

```
import numpy

# buffer (length num_samples) contains data from the radio
input_array = numpy.asarray(buffer)
power_sum = numpy.sum(input_array.real ** 2 + input_array.imag ** 2)
average_power = power_sum / num_samples
```

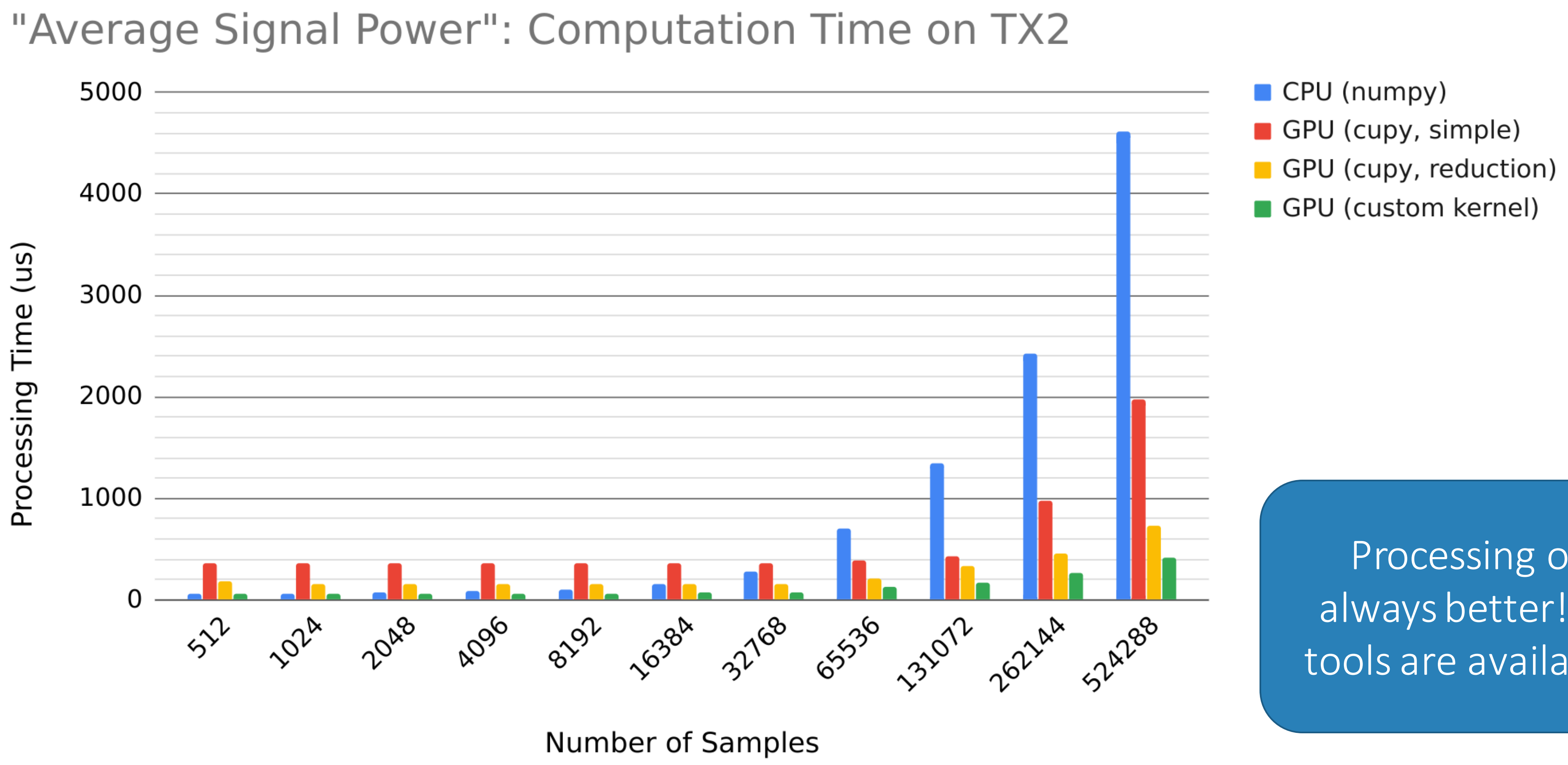
```
import cupy

# buffer (length num_samples) contains data from the radio
input_array = cupy.asarray(buffer)
power_sum = cupy.sum(input_array.real ** 2 + input_array.imag ** 2)
average_power = float(power_sum) / num_samples
```

Processing data on the ARM CPU

Processing on the GPU
and reading back a single float

AIR-T: numpy vs. cupy vs. a custom kernel



Processing on the GPU is not always better! **Benchmark,** good tools are available and easy to use

Deep Dive: Why did we see this performance?

- Annotate each signal processing function to capture a profile using *nvprof*
- cupy makes this simple, and you can analyze mixed CPU/GPU code just as easily
- Use pytest-benchmark to run each function many times...

```python
import cupy.prof

def test_cupy(num_samples, benchmark):
    input = numba.cuda.mapped_array(num_samples, dtype=numpy.complex64)
    input[:] = numpy.complex64(1.0 + 0.0j)
    input_array = cupy.asarray(input)

    @cupy.prof.TimeRangeDecorator()
    def inner_loop_cupy():
        power_sum = cupy.sum(input_array.real ** 2 + input_array.imag ** 2)
        average_power = float(power_sum) / num_samples
    benchmark(inner_loop_numpy)
```

Deep Dive: Profiling Results

Profile of 16384 samples case, GPU (cupy) and GPU (custom kernel)

```
==31551==       Range "inner_loop_cupy"
            Type  Time(%)      Time   Calls       Avg       Min       Max  Name
          Range:  100.00%  595.78ms    1002  594.59us  453.15us  10.181ms  inner_loop_cupy
 GPU activities:   38.94%  43.836ms    1002  43.748us  23.682us  96.553us  cupy_multiply
                   38.10%  42.883ms    1002  42.797us  27.747us  85.928us  cupy_conj
                   16.39%  18.446ms    1002  18.409us  9.2170us  58.886us  cupy_sum
                    6.58%  7.4023ms    1002  7.3870us  1.1200us  17.282us  cupy_true_divide
      API calls:  100.00%  170.54ms    4008  42.549us  31.648us  433.38us  cuLaunchKernel

==31551==       Range "inner_loop_custom_kernel"
            Type  Time(%)      Time   Calls       Avg       Min       Max  Name
          Range:  100.00%  109.58ms    1002  109.36us  91.296us  1.5531ms
 GPU activities:  100.00%  23.676ms    1002  23.628us  18.721us  27.683us  average_power
      API calls:  100.00%  32.936ms    1002  32.869us  30.208us  104.70us  cuLaunchKernel
```

Profiler immediately shows each cupy kernel and the overhead of kernel launch.
Note the lack of any memory copying calls in either case!

Example: Performing Inference

```
import tensorrt

# Input and output buffers created using numba.cuda.mapped_array().
# Input is samples from radio, output is inference results from the network.

network_file = "demo_network.plan"
batch_size = 1
runtime = tensorrt.Runtime()
stream = cupy.cuda.Stream()
with open(network_file, 'rb') as file:
    engine = runtime.deserialize_cuda_engine(file.read())
    context = engine.create_execution_context()
    context.execute_async(batch_size, stream_handle=stream.ptr,
        bindings=[ int(input_buffer.__cuda_array_interface__['data'][0]),
                   int(output_buffer.__cuda_array_interface__['data'][0]) ])
    stream.synchronize()
```

Signal Processing Recap

Python signal processing code can be shared between training and inference pipelines

Rich open-source libraries exist today to make this easy on the GPU

Profiling support is mature and will help you optimize

Algorithms can be easily wrapped in GNU Radio blocks or python to integrate deep learning with a larger signal processing system

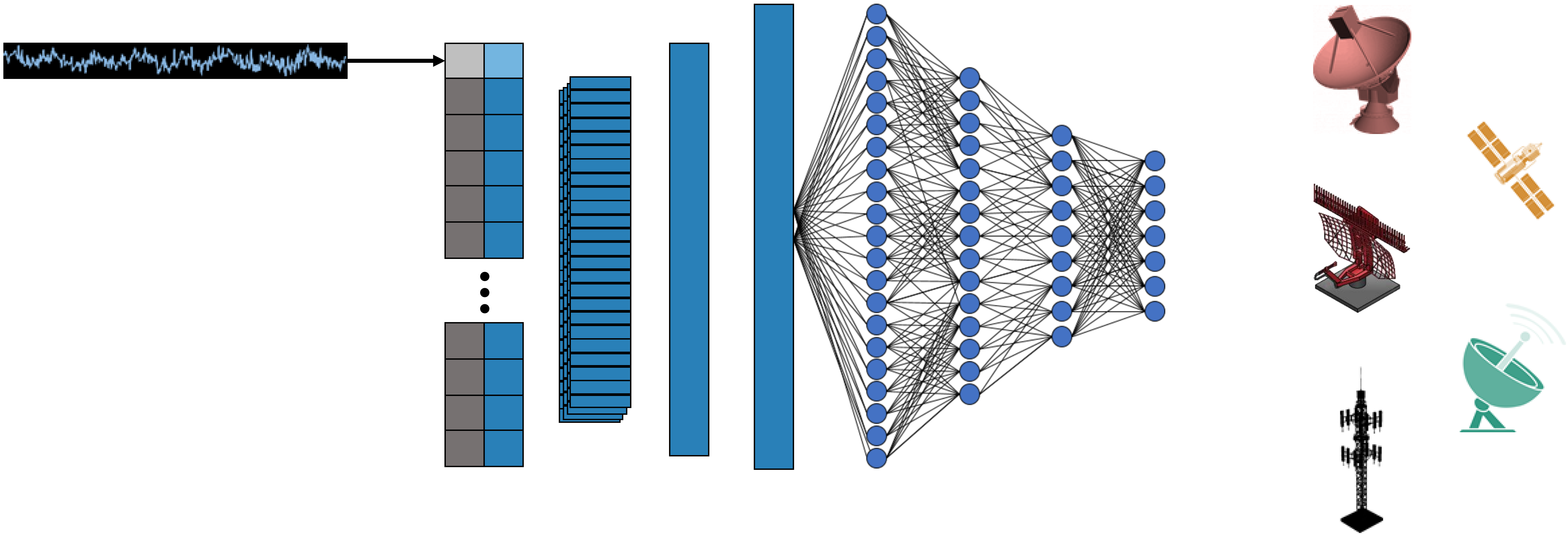
Deep Learning Wireless Deployment Scenario

Goal: Detect and classify signals in congested environment using AIR-T

Radar Signal Detector Model

Example Classifier

Signal stream in → Process with deep learning → Signal identification out



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							

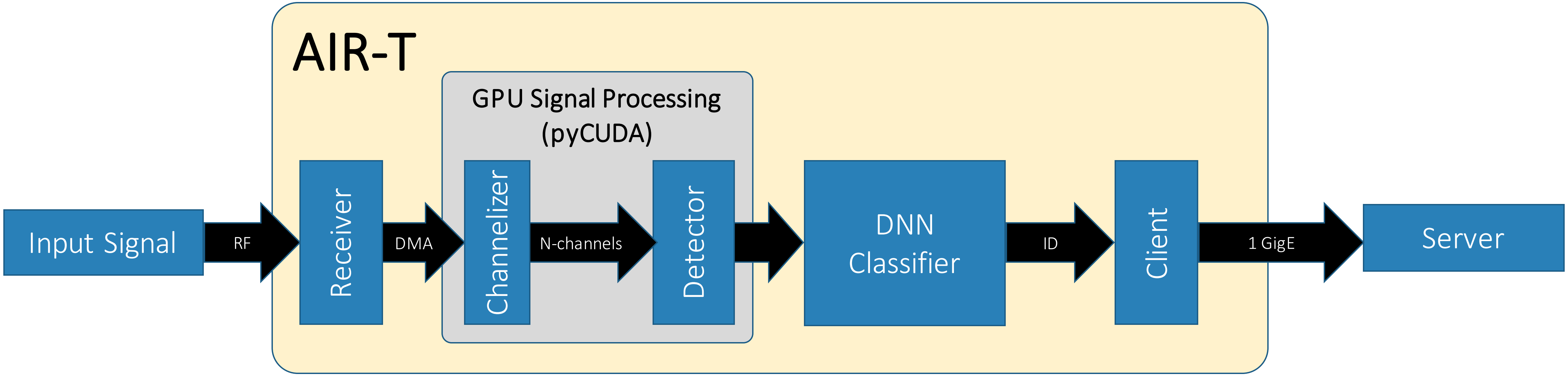
Technique demonstration shown with nominal radar signals

- Method applicable to communications, cellular, and other RF protocols

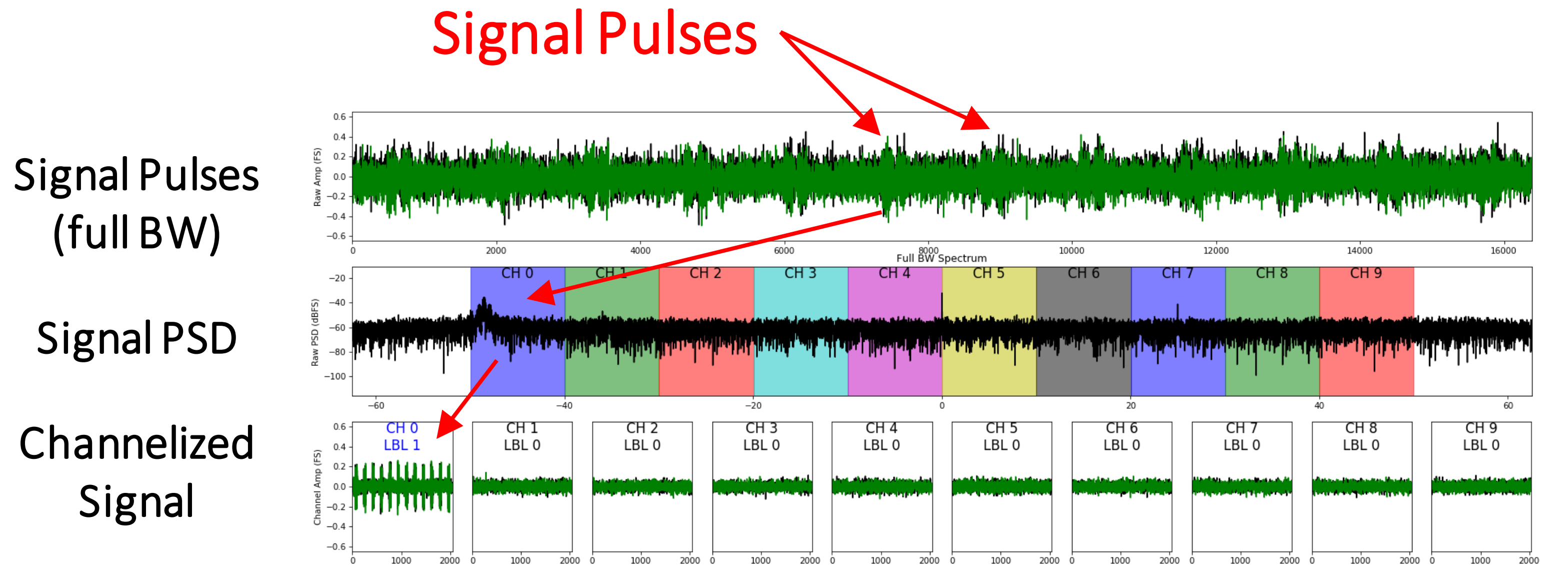
Real Time Deep Neural Network (DNN) Classifier

Monitors 125 MHz of instantaneous bandwidth

- Trained on 10,000+ signal segments
- Hardened through channel models and analog distortions
- 125 MSPS GPU channelizer
- Modular design



Pre-processing Overview and Methodology



- Significantly improves signal classification performance
  - Increases SNR and PCC
  - Signal isolation
- Provide separate path for negative SNR (signal less likely to be present) cases

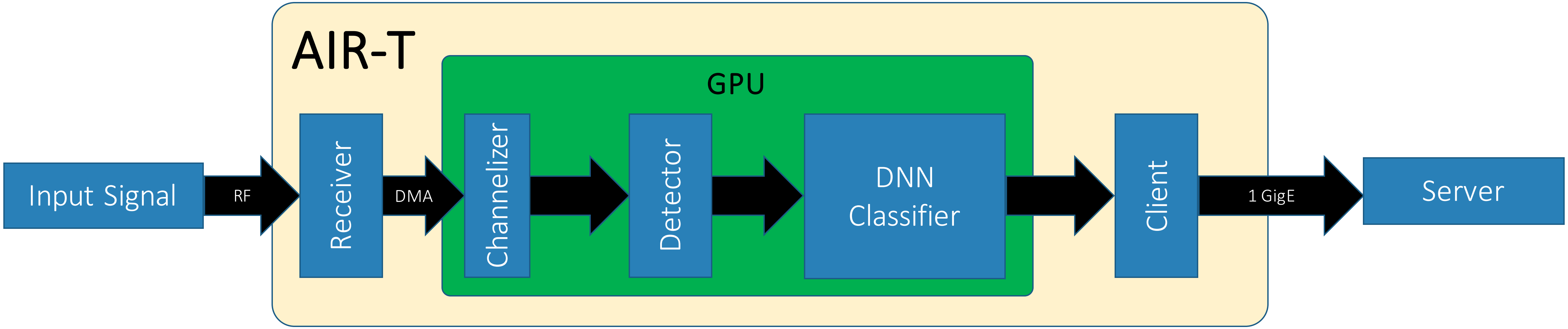
Synthetic Data → **Laboratory Data** → **Over the Air Data**

Prove model with synthetic data (TRL-3)

Validation in controlled environment (TRL-6)

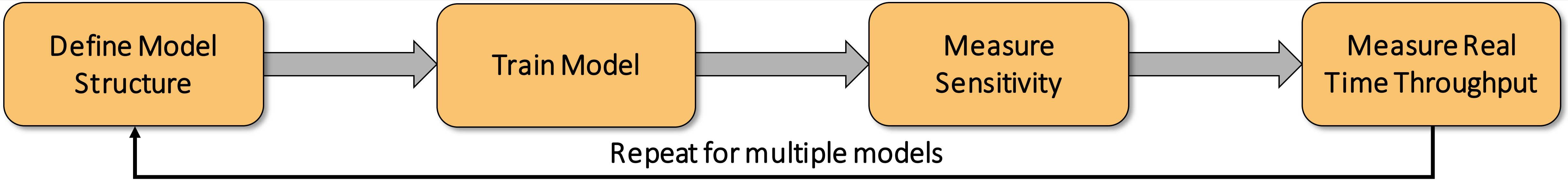
Validation in deployed environment (TRL-9)

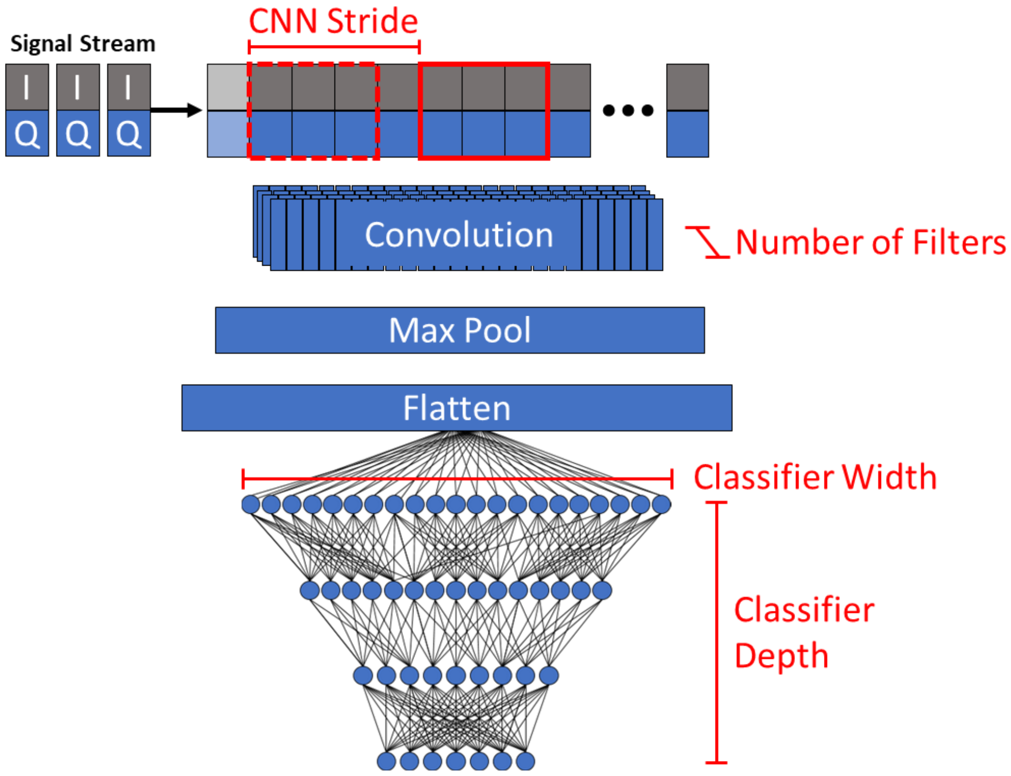
Processing Utilization on AIR-T



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

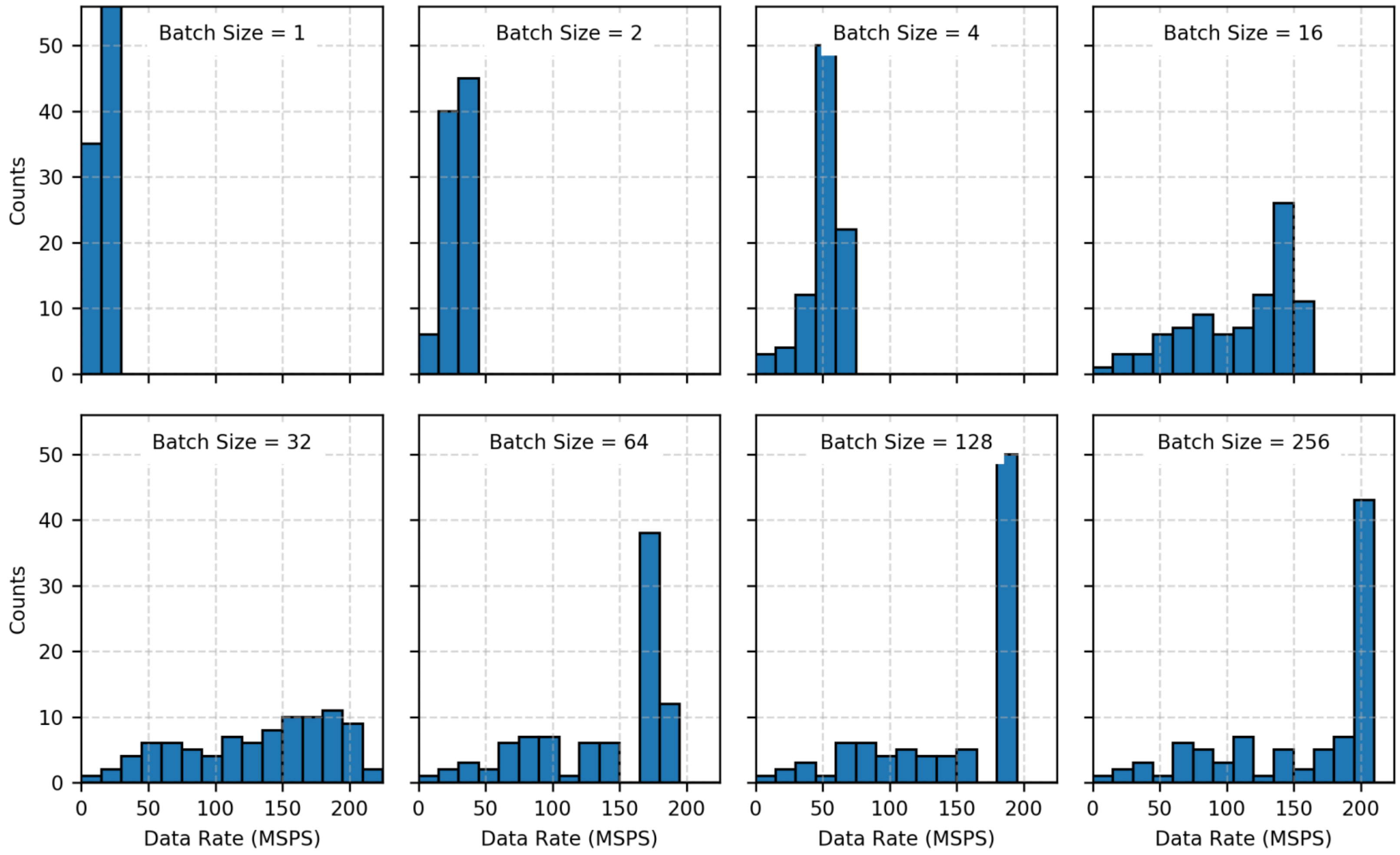
Performance Benchmarking Test Setup





- Stream unthrottled data to network
- Measure data rate at two locations:
  1. Aggregate data rate for entire process
     - Number of bytes processed / wall time
  2. Computation data rate in work() function
     - Number of bytes processed / computation time

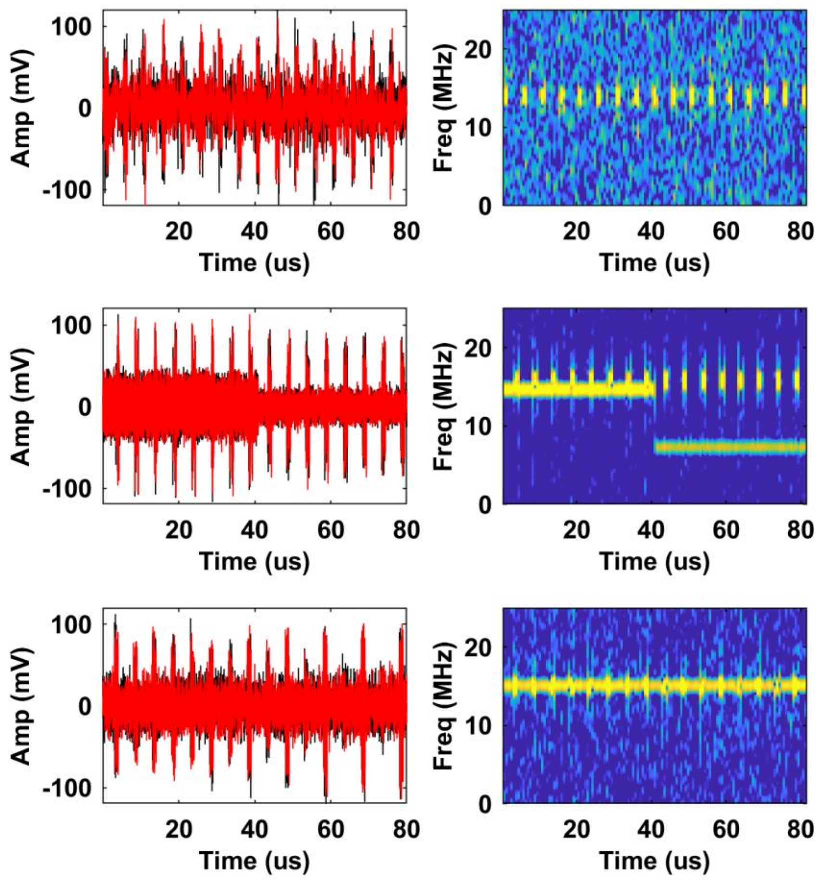
Data Rate Benchmark for AIR-T (Jetson TX2)

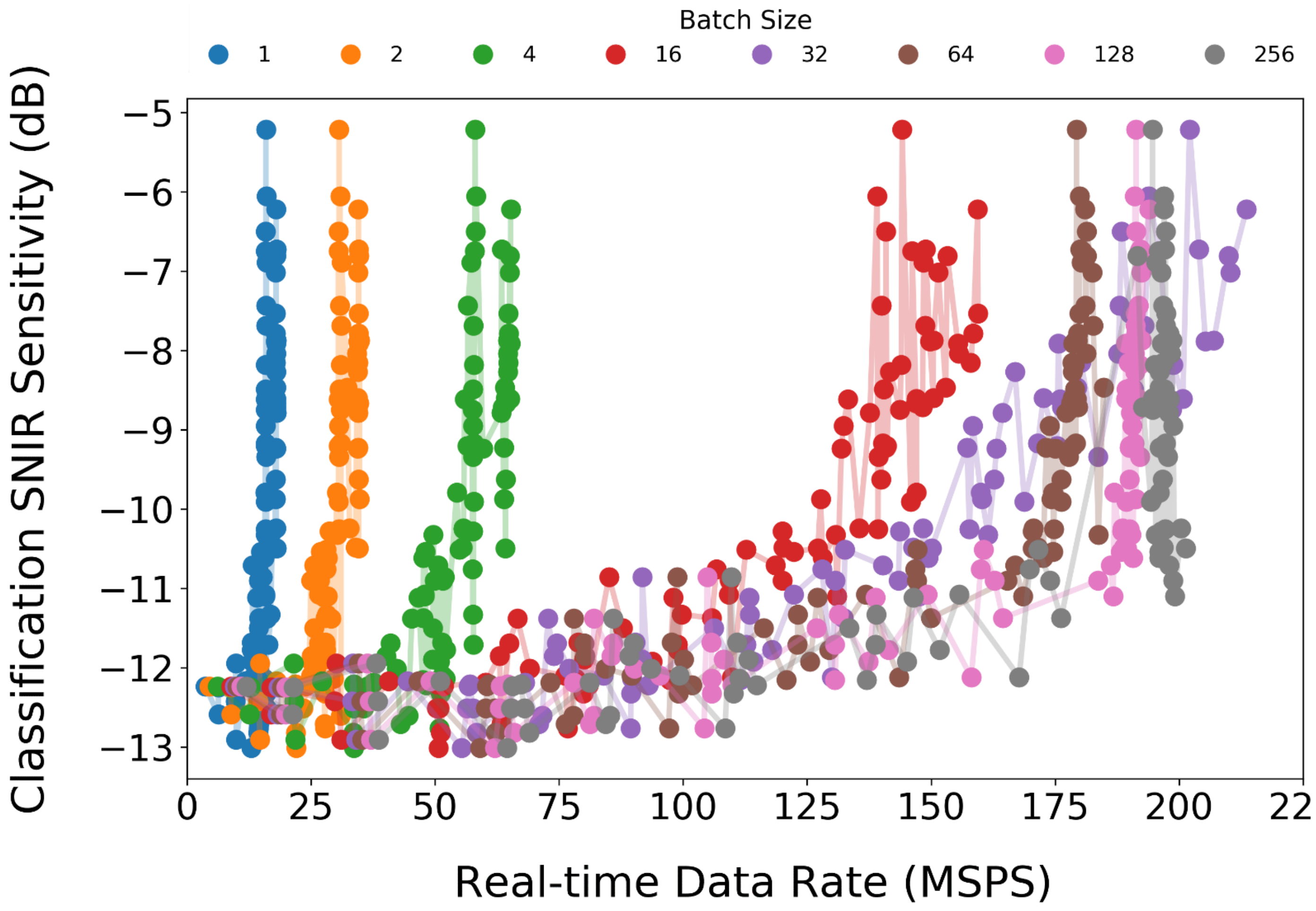


- Tested 91 different CNN classifier models with 8 batch sizes
  - 728 models tested
- Able to achieve 200 MSPS (real samples) with AIR-T

AIR-T

Model Accuracy Benchmarks

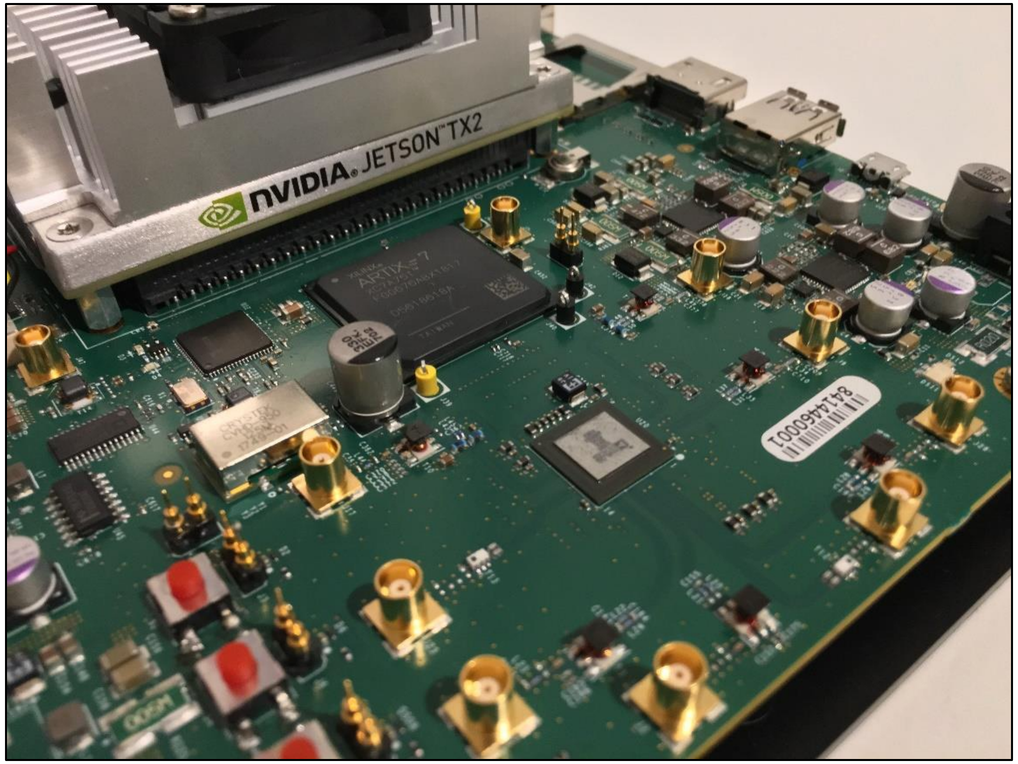




Summary

- Deep learning within signal processing is emerging
- High bandwidth requirements driving edge solutions
  - Embedded GPUs now suitable for signal processing
- Deepwave developed AIR-T
  - Edge-compute inference engine with MIMO transceiver
  - FPGA, CPU, GPU
- Open-source Python ecosystem for deep learning with signals is improving daily: give it a try!
- Benchmarking demonstrates real-time signal classification inference at rates approaching 200 MSPS with AIR-T

More info at www.deepwavedigital.com/sdr



DEEPWAVE

DIGITAL

info@deepwavedigital.com