

Holoscan and USRP: A GPU and SDR Case Study

Martin Braun

Ettus Research / NI /
Emerson

Sage Trudeau

UT Austin

Holoscan and DAQIRI

What's a Daqiri? Is it already that time
of day?

What are Holoscan and DAQIRI?

- Holoscan: A data movement and GPU processing framework by
 - Open Source, Permissively Licensed
 - Composable architecture, block-based design (“one block p
 - All right, let’s be real, this is GNU Radio for GPUs
 - Ideal for Ethernet-based I/O, good for streaming application
 - Developers focus on building IP blocks (GPU algorithms), d
 - memory management is handled by the framework
 - Can be used with Ethernet-capable devices, USRP is a gre
 - interoperability
 - Has a Python and a C++ API, those are not (yet?) interoper
- Daqiri: The I/O library that handles data movement
 - “Data Acquisition for Integrated Real-time Instruments”
 - Was originally part of Holoscan, and they connect well

Holoscan

NVIDIA’s Streaming Sensor Platform

What is Holoscan?

- High-performance platform for sensor data processing and AI inferencing.
- Leverages the power of NVIDIA GPUs for efficient data movement and computation.

Benefits

- Scalability: Handles high data volumes and scales with additional hardware.
- Simplicity: Easier to build and deploy sensor processing applications.

Advanced Network Operator (ANO):

- Abstracts system tuning and GPUDirect RDMA implementation.
- Enables data to bypass CPU and directly reach GPUs for faster processing.

[Luigi Cruz, GRCon ‘24]



Nvidia Holoscan

GNU Radio <-> Holoscan Comparison

GNU Radio

- Scheduler stems from the early 2000s, but very stable API
- CPU focused, very portable
- Focus on radio/DSP
- Switch between C++ API / Python API / GRC, combine components from all language domains
- Very mature and comprehensive block library, both in-tree and out-of-tree
- Between CGRAN and github: Easy to find out-of-tree modules

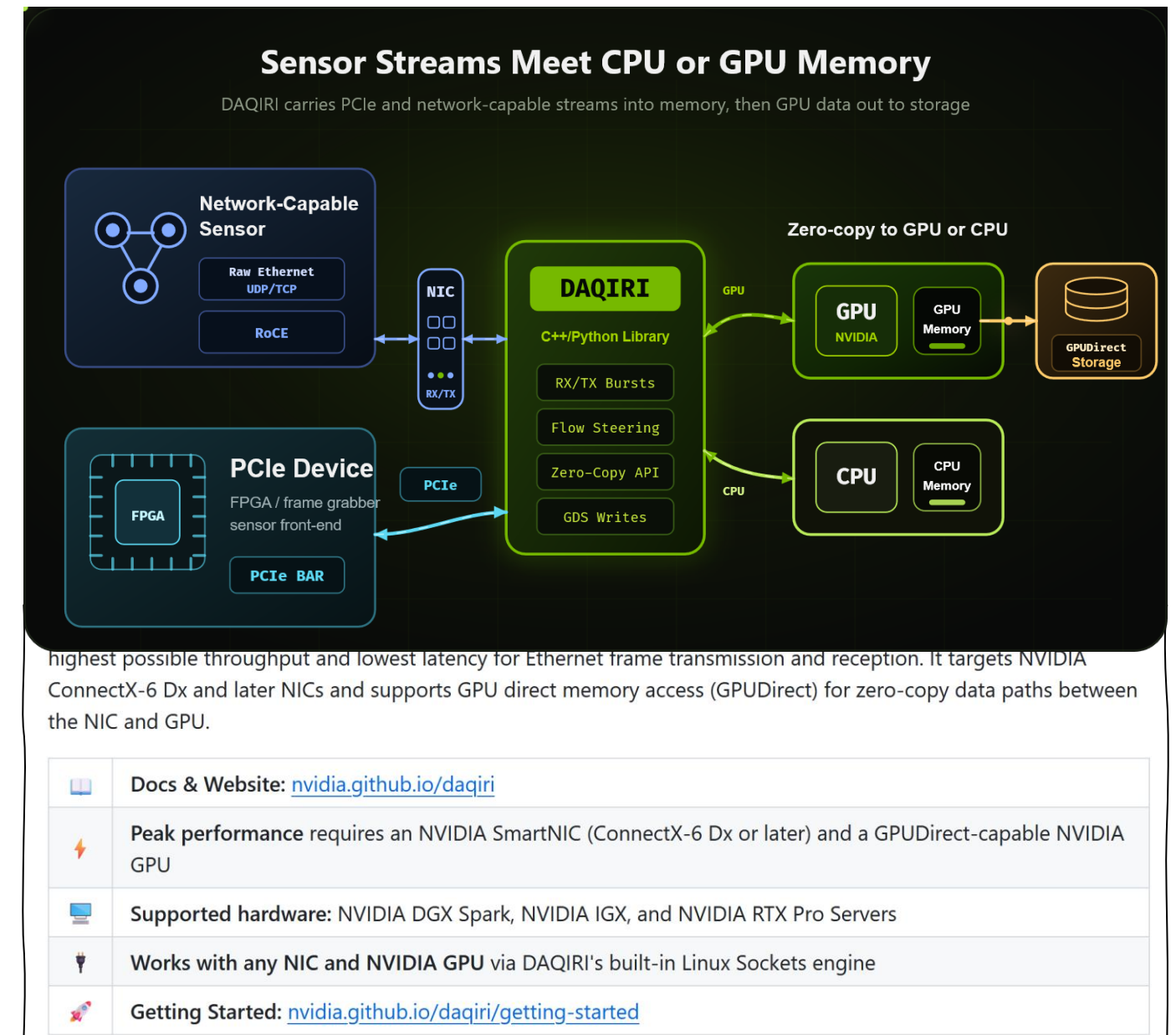
Holoscan

- Modern architecture, but APIs change frequently
- Made for use with Nvidia GPUs
- Comes from medical imaging (?), applications very varied
- No GUI whatsoever, even visualizations are highly limited, C++/Python not interoperable
- Extension community not well established, Nvidia not focused on building out open-source block library
- Holohub is the main extension repository, where all users upstream their extensions/applications/examples

Daqiri: Abstracting the I/O



- Assumption: You are using lots of Nvidia-compatible hardware (GPU, ConnectX NIC, ideally DGX/IGX/RTX machines)
- Daqiri abstracts different transport modes (e.g., RDMA, raw UDP + DPDK, ...) to easily enable GPUDirect support for your external sensor (e.g., your SDR)
- If you get it installed on your system, and the hardware allows it, then you can easily stream data rates of 2000 Msps (8 Gbyte/s) or more directly into your GPU memory space
- Runtime configuration options are provided through a YAML file



Examples: config.yml and main app

```

1 # SPDX-FileCopyrightText: 2026 National Instruments Corporation
2 #
3 # SPDX-License-Identifier: Apache-2.0
4 ---
5 num_runs: -1 # Number of iterations the application will run, -1 is infinite
6
7 scheduler:
8   worker_thread_number: 4
9   stop_on_deadlock: true
10  stop_on_deadlock_timeout: 500
11  enable_worker_postcheck_fastpath: true
12  enable_queue_stealing: true
13
14 usrp_rx:
15   # Need to be set to true to enable data transfer from USRP.
16   # Make sure that the "<device_addr>", "<dest_addr>", "<dest_mac_addr>" below are set correctly to your target setup
17   enabled: false
18   # Set your real device args before enabling, e.g.:
19   # args: "addr=10.88.136.10,master_clock_rate=500e6"
20   args: "addr=<1.2.3.4>,master_clock_rate=500e6"
21   rate: 500e6 # Hz (500 MHz); recommended to match spectrum_viz.ref_bandwidth_hz
22   freq: 1000e6 # Hz (1000 MHz); recommended to match spectrum_viz.ref_center_hz
23   gain: 0.0
24   channels: [0, 1]
25   # Set the DPDK host IP before enabling.
26   dest_addr: <5.6.7.8> # Destination IP address
27   dest_ports: [1234, 1235]
28   adapter: "sfp0"
29   dest_mac_addr: <00:00:00:00:00:00> # Destination MAC address
30   keep_hdr: true
31   spp: 1024
32   mtu: 8064
33   start_delay_seconds: 0.05
34
35 daqiri:
36   cfg:
37     version: 1
38     stream_type: "raw"
39     master_core: 9 # Master CPU core
40     debug: false
41     log_level: "info"
42
43   memory_regions:
44     # Memory pool for frame headers
45     - name: "Headers_RX_CPU"
46       kind: "huge"
47       affinity: 0
48       access:
49         - local
50       num_bufs: 12500
51     # Size of the Ethernet header + IPv4 header + UDP header
52     buf_size: 42
53     # Memory pool for channel 1 CHDR packet headers
54     - name: "CH1_CHDR_Headers_RX_CPU"
55       kind: "huge"
56       affinity: 0
57       access:
58         - local
59       num_bufs: 12500
60     # Size of the CHDR header (depends on CHDR block size)

```

```

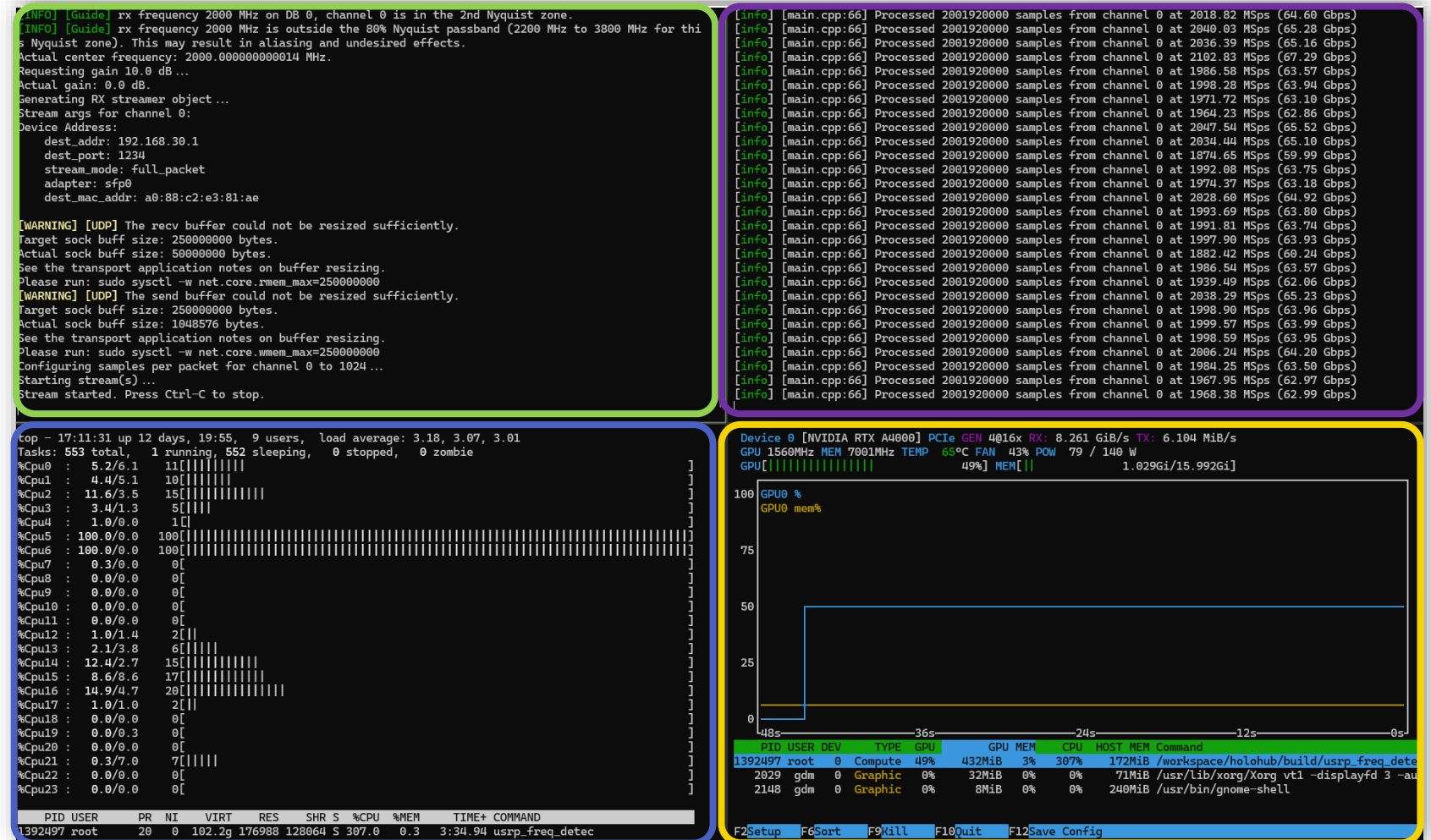
4 #include "usrp_rx/usrp_rx.hpp"
5 #include "uhd_chdr_rx/uhd_chdr_rx.hpp"
6 #include "log/log.hpp"
7 #include "spectrum_overlay/spectrum_overlay.hpp"
8 #include "spectrum_visualizer/spectrum_visualizer.hpp"
9 #include "spectrum_magnitude/spectrum_magnitude.hpp"
10 #include <fft.hpp>
11 #include <holoscan/operators/holoviz/holoviz.hpp>
12 #include <memory>
13
14 class UsrpSpectrumViewerApp : public holoscan::Application {
15 public:
16   void compose() override {
17     using namespace holoscan;
18
19     // Shared state couples the Holoviz overlay controls with the spectrum
20     // visualizer's pan/zoom and peak reporting logic.
21     auto view_state = std::make_shared<ops::SpectrumViewState>();
22     auto overlay_config = apps::SpectrumOverlayConfig{
23       .db_min = from_config("spectrum_viz.db_min").as<float>(),
24       .db_max = from_config("spectrum_viz.db_max").as<float>(),
25       .initial_center_freq_mhz =
26         from_config("spectrum_viz.ref_center_hz").as<float>() * 1.0e-6F,
27       .initial_bandwidth_mhz =
28         from_config("spectrum_viz.ref_bandwidth_hz").as<float>() * 1.0e-6F,
29       .num_channels = from_config("spectrum_viz.num_channels").as<int>();
30
31     auto usrp_rx = make_operator<ops::UsrpRxOp>(
32       "usrp_rx",
33       from_config("usrp_rx"));
34
35     auto uhd_chdr_rx = make_operator<ops::UhdChdrRxOp>(
36       "uhd_chdr_rx",
37       from_config("uhd_chdr_rx"));
38
39     auto fft = make_operator<ops::FFT>(
40       "fft",
41       from_config("fft"));
42
43     auto log = make_operator<ops::LogOp>(
44       "log",
45       from_config("log"),
46       make_condition<CountCondition>(from_config("num_runs").as<int64_t>()));
47
48     auto spectrum_magnitude = make_operator<ops::SpectrumMagnitudeOp>(
49       "spectrum_magnitude",
50       from_config("spectrum_magnitude"));
51
52     // Managed stream pool lets the framework event-sync with Holoviz (no host block).
53     auto spectrum_viz = make_operator<ops::SpectrumVisualizerOp>(
54       "spectrum_viz",
55       from_config("spectrum_viz"),
56       make_resource<CudaStreamPool>("spectrum_viz_stream_pool", 0, 0, 0, 1, 5));
57     // The visualizer reads live pan/zoom and peak-toggle state from the UI.
58     spectrum_viz->set_view_state(view_state);
59
60     auto holoviz = make_operator<ops::HolovizOp>(

```

Holoscan-Based Applications

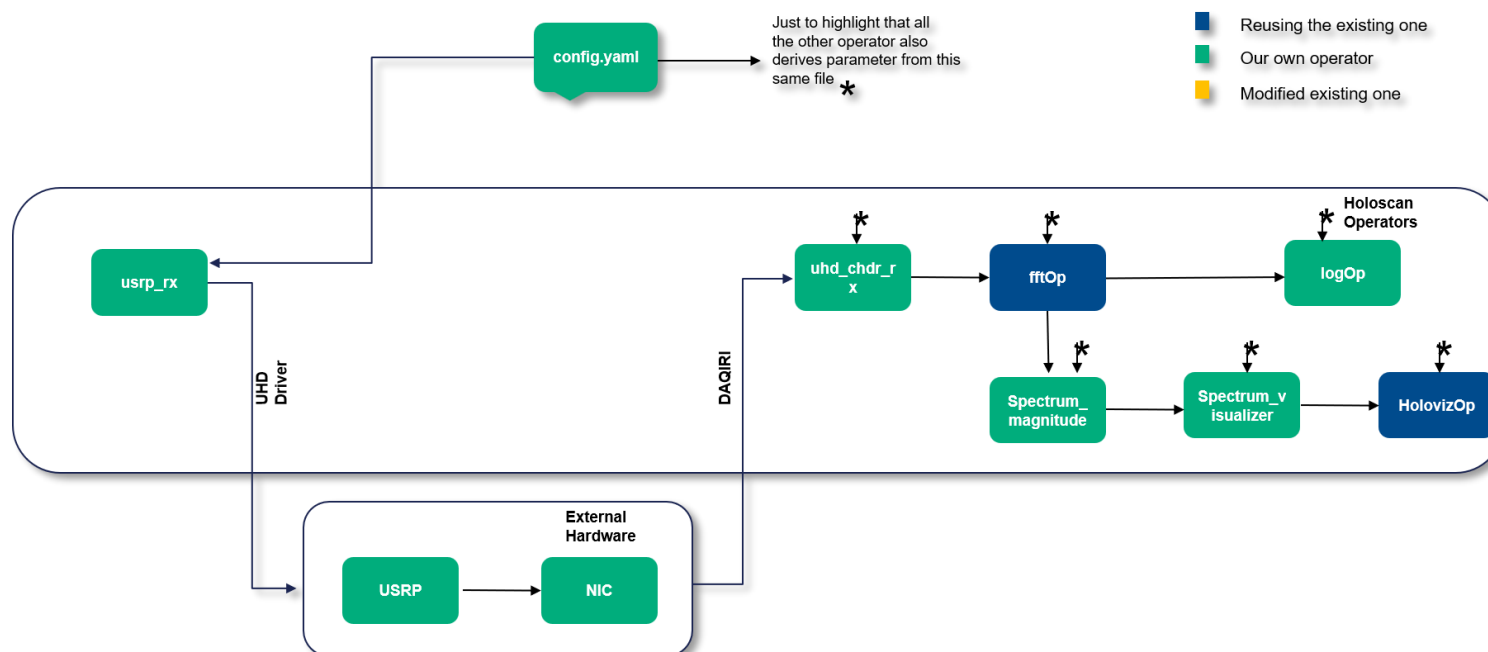
It all started with a PoC...

- We wanted to see if Holoscan gives us anything useful
- We used the raw UDP streaming feature of the USRP to produce a high-rate output that could be ingested by Holoscan
- Various platforms were tested
- As a stand-in for an SDR app, we simply ran a 20k FFT
- Full rate of X440 (2000 Msps) could be streamed without losing data
- Uses DPDK for data transfer, blocks CPU cores
- This could be useful!



USRP Spectrum Viewer

- Spectrum Viewer: Super-simple demo that enables SDR users to get started with Holoscan
- Uses Holoviz for visualization
- No sample ever leaves GPU memory!
- Available on Github/Holohub! (Thanks to our intern Abhiram!)



USRP Frequency Spectrum

NVIDIA | **HoloHub - Explore The Holoscan Ecosystem** | Search

Home | Holoscan SDK | HoloHub | Holoscan Sensor Bridge | Support | Blog

Home > Applications > USRP Spectrum Viewer

DPDK | HOLOVIZ | SDR | Signal Processing | USRP

USRP Spectrum Viewer

Authors: Andrew Lynch andrew.lynch@emerson.com (National Instruments Corporation.), Johannes Lange johannes.lange@emerson.com (National Instruments Corporation.), Abhiram Anil abhiram.anil@emerson.com (National Instruments Corporation.)

Supported platforms: x86_64, aarch64

Language: C++

Last modified: September 9, 2026

Latest version: 1.0.0

Minimum Holoscan SDK version: 4.4.0

Tested Holoscan SDK versions: 4.4.0, 4.5.0, 4.6.0

Contribution metric: Level 3 - Developmental

[Run Locally](#)

Limitations

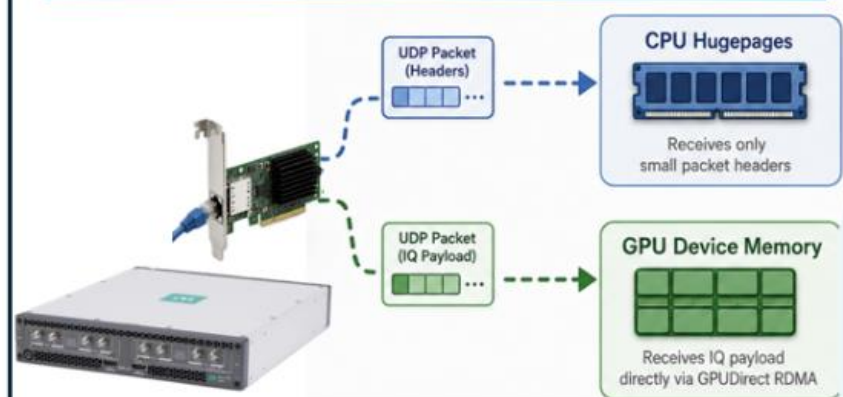
- Currently, lots of limitations:
 - No flow control support: You must make sure the GPU algorithms can handle all incoming data, or data is lost and bad things happen
 - No Tx support: Consequence of no flow control

ARGUS

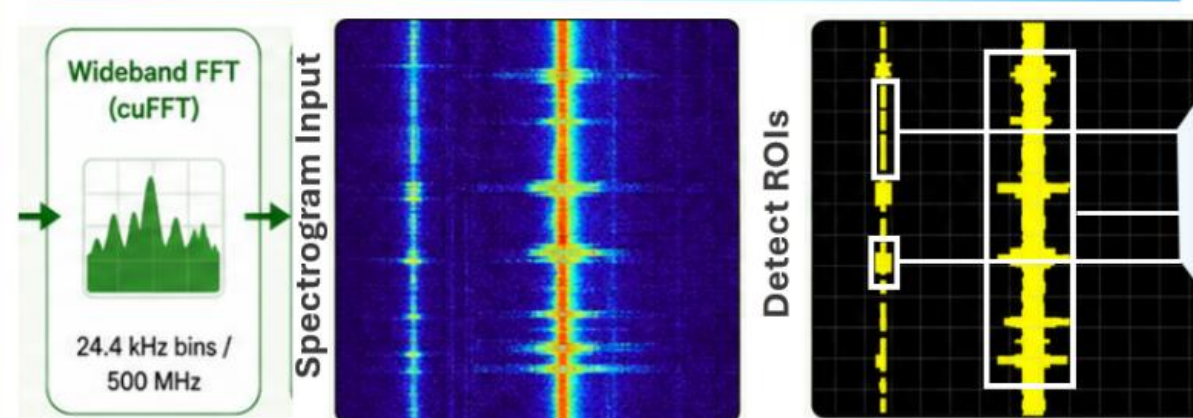
SAGE TRUDEAU, BRANDON NGUYEN, DIVYADHARSHINI MURUGANANDHAM, KAUSHIK CHOWDHURY

How do we deploy real time RF-AI models in the wild?

Direct to GPU Data Ingest



Wideband Signal Detection and Extraction

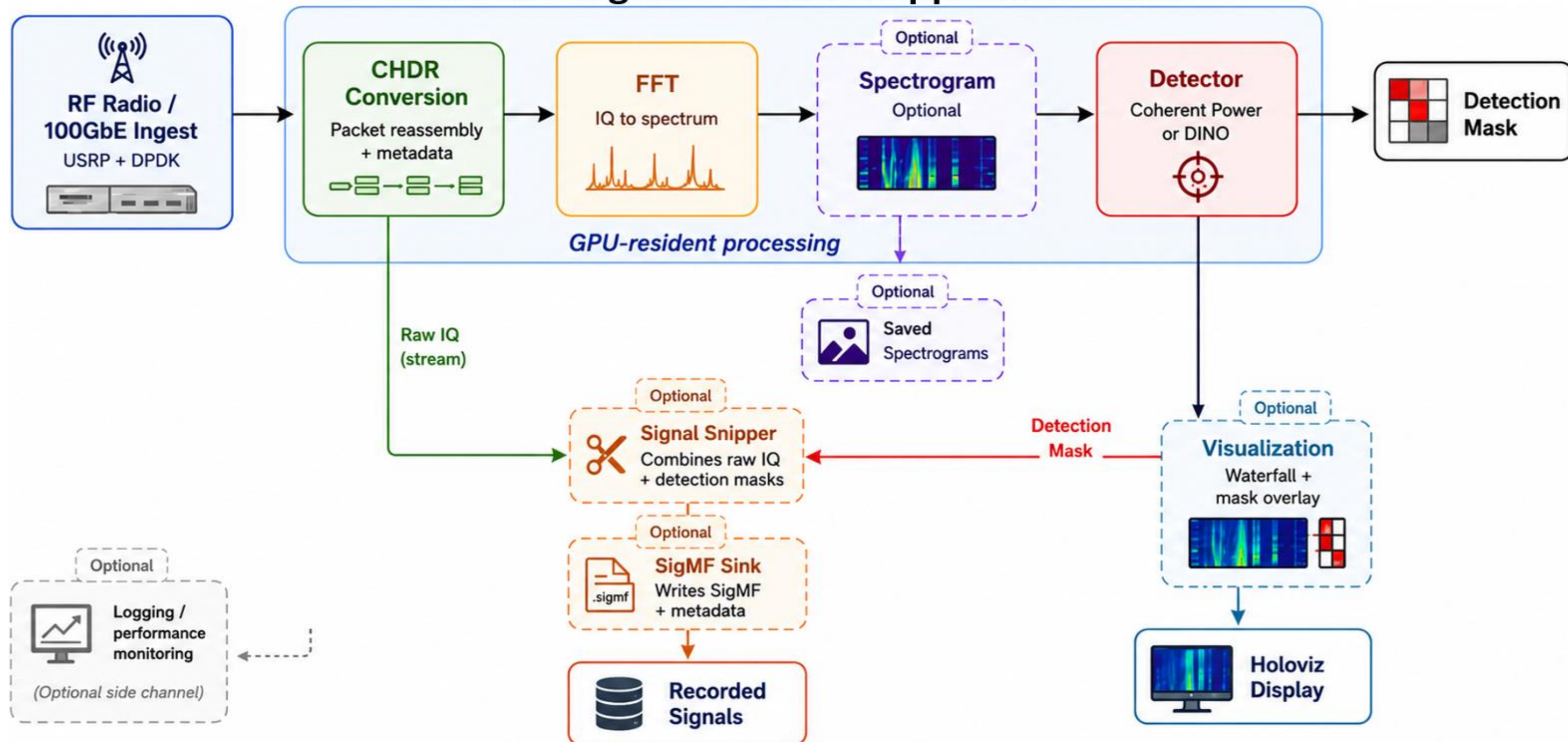


Downstream RF AI



ARGUS addresses whether the end-to-end wideband system can keep up with data delivery to RF AI models at all

Holoscan Signal Detection Application Flow



Control Setup



USRP X410

Waveform Library

RF Coax



USRP X410

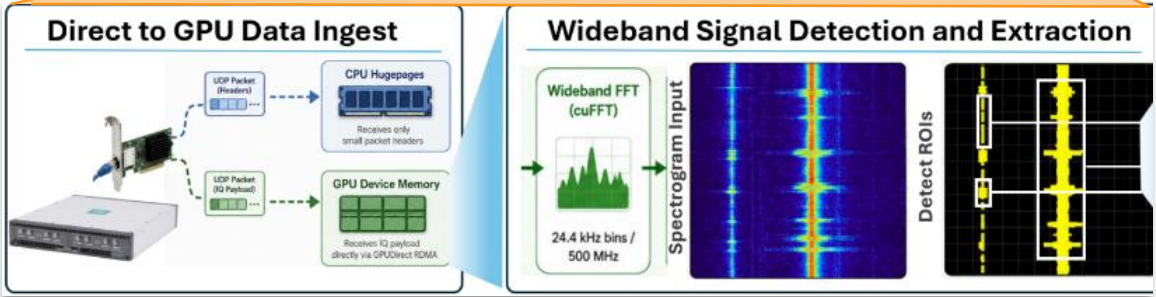
QSFP+ 100GE



DGX Spark

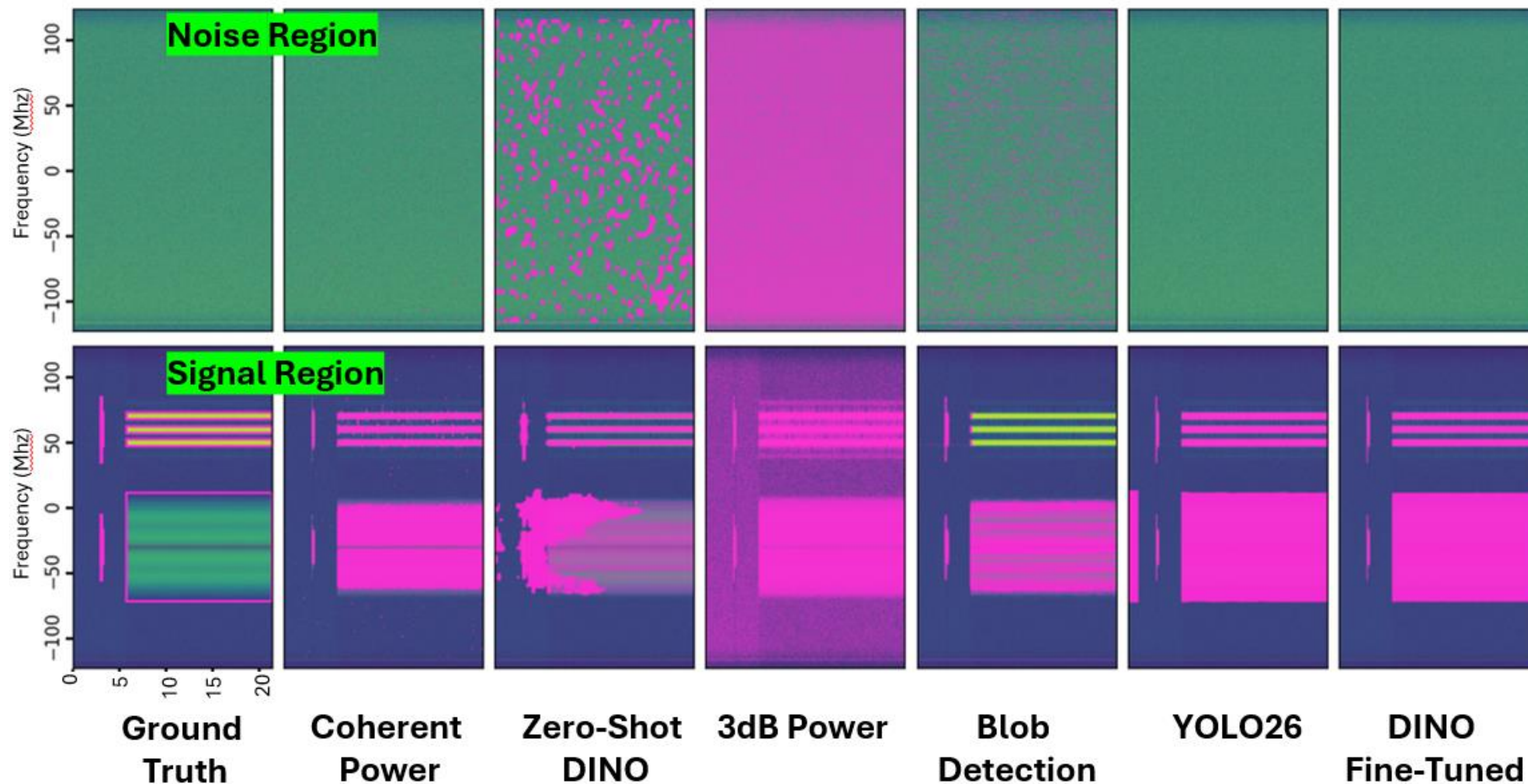
Table 2: Representative coverage of the controlled ARGUS evaluation library.

Family	Approx. occupied bandwidth	Detection challenge
BPSK/QPSK/16QAM	0.29–82.94 MHz	Single-carrier PSK/QAM over a broad bandwidth range
OFDM	4.50–98.28 MHz	Structured multicarrier emissions
5G NR downlink	3.96–98.28 MHz	Standardized wideband multicarrier traffic
802.11ax	20–160 MHz	Very wide Wi-Fi emissions
Bluetooth	1–2 MHz	Narrow frequency-hopping/GFSK activity
Broadband FM	0.13–61.47 MHz	Analog FM over highly variable bandwidth
Narrowband FM	0.01–0.03 MHz	Extremely thin spectral support

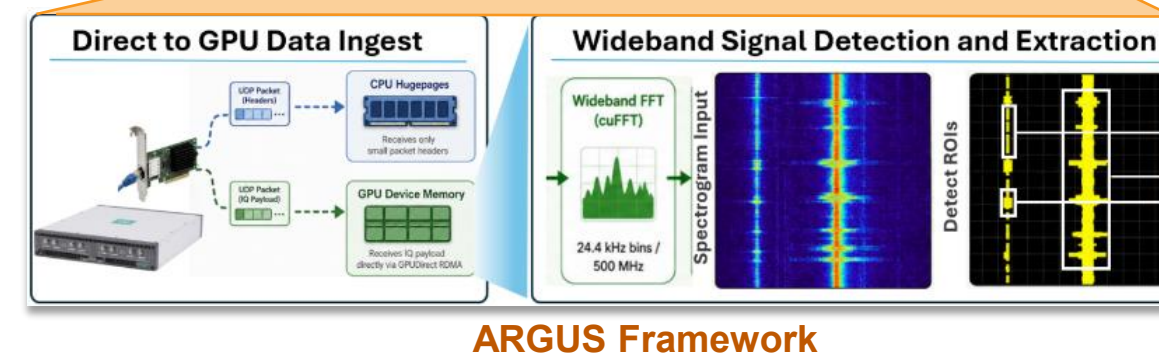


ARGUS Framework

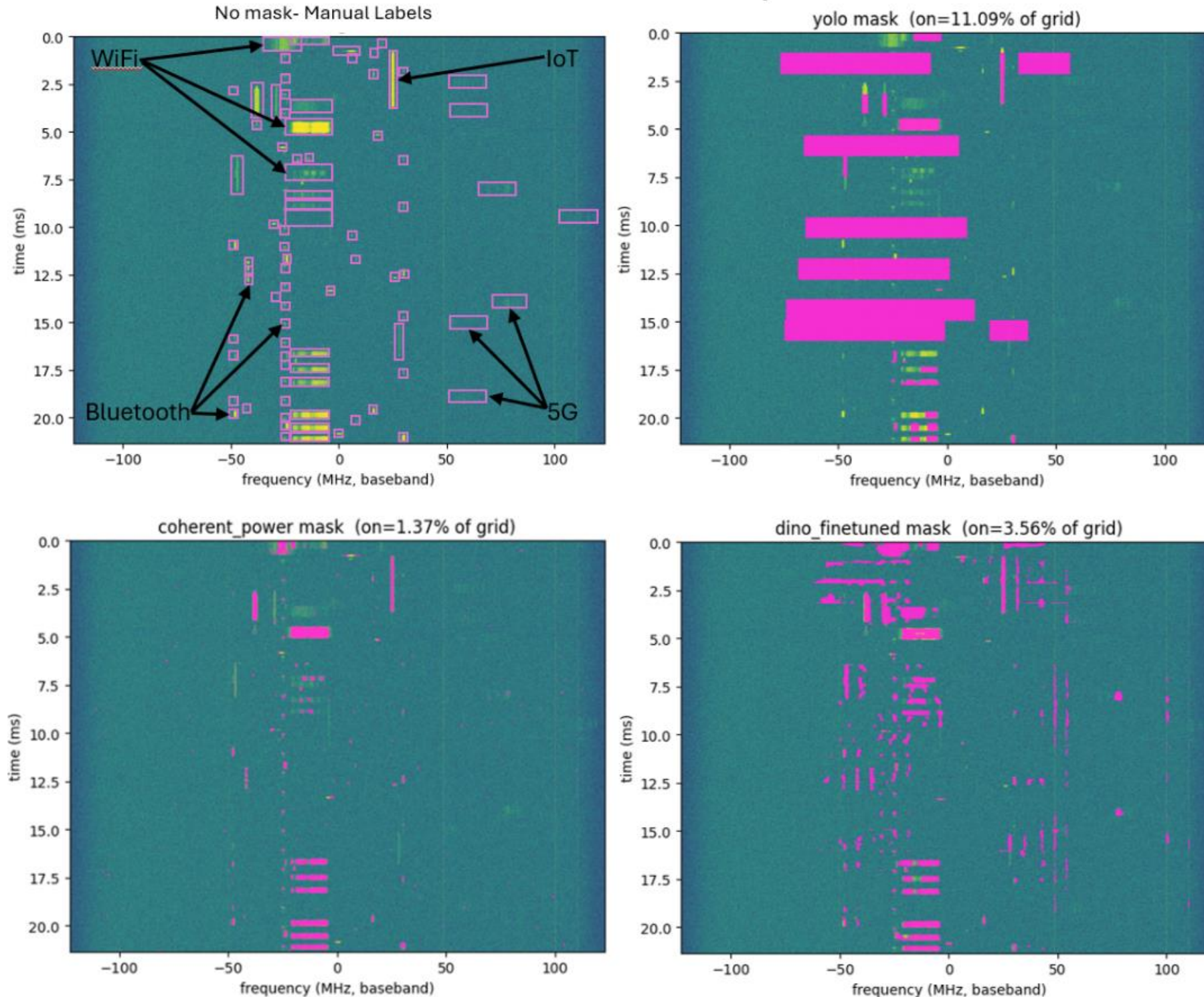
Qualitative Detector Comparison



Deployment Setup



2.4Ghz ISM OTA Comparison

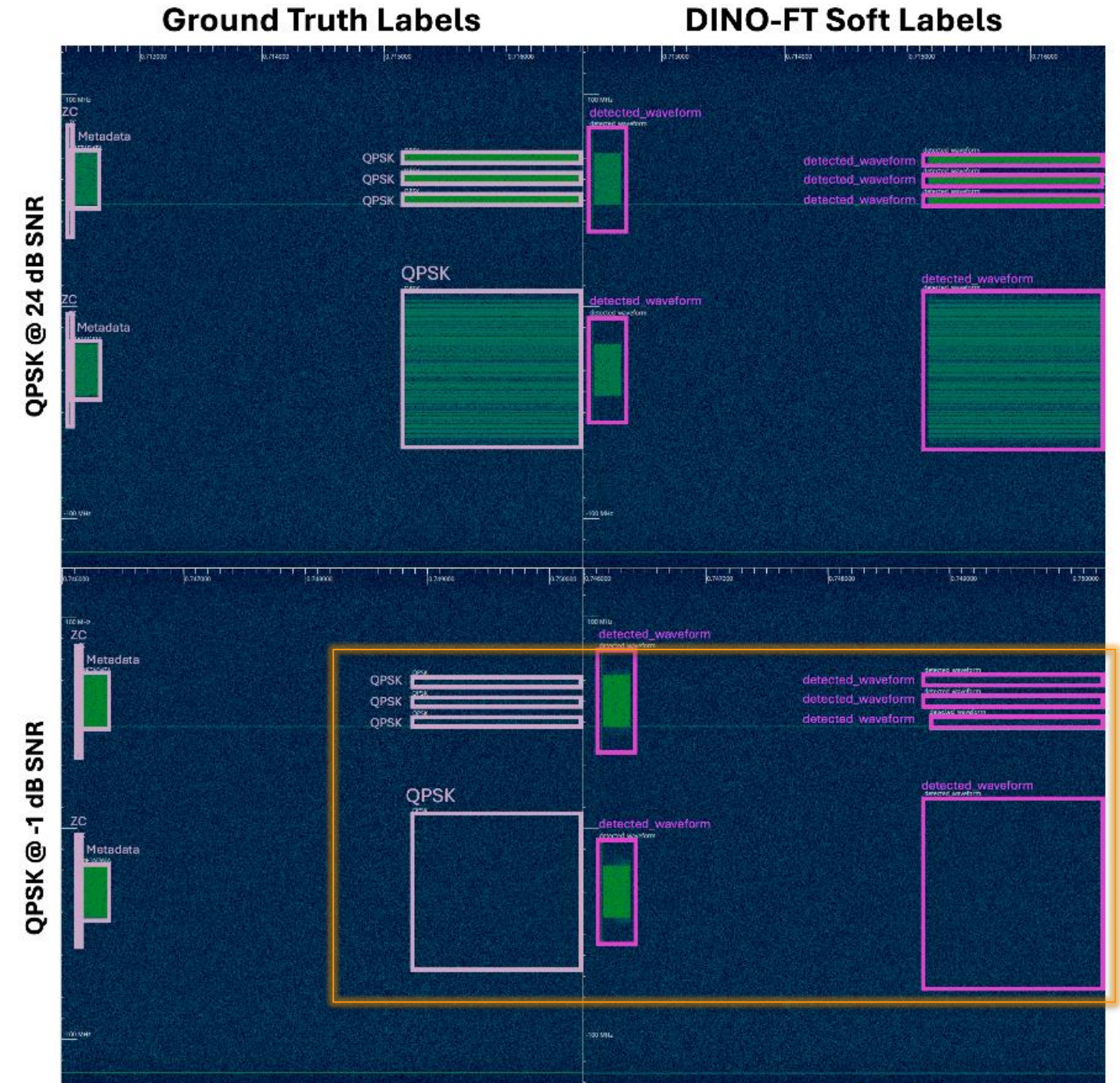


Out of Distribution Environment

- Fine-tuned DINO localizes several weaker structures near the visible noise floor
- Coherent power primarily captures strong emissions
- YOLO produces larger box-shaped regions and misses several narrowband structures.

Low SNR Conditions

- Can you see the QPSK signal in the orange region?
- DINO can!



DINO-FT shows impressive capability to localize signal region below the noise

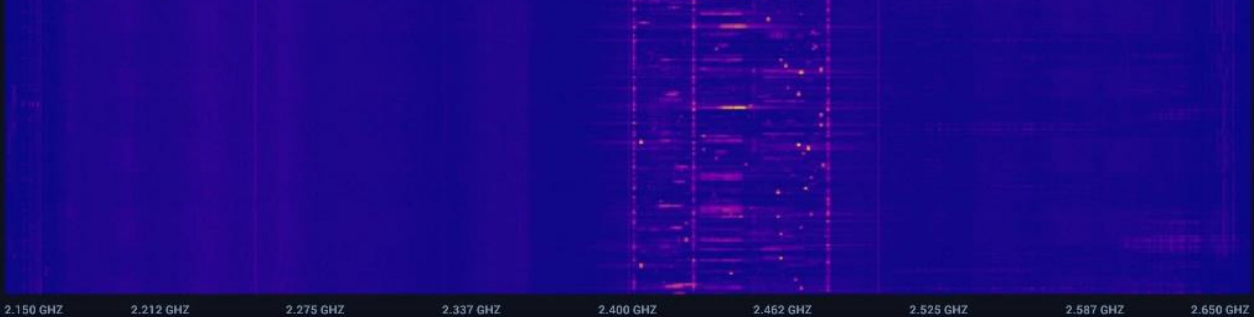
Enabling RF AI for Ultra Wideband Real Time Applications

Holoscan enables GPU native real time signal processing and computer vision models to detect and extract regions of interest from dual 500 MSps streams, reducing RF AI workloads to only the regions that matter.

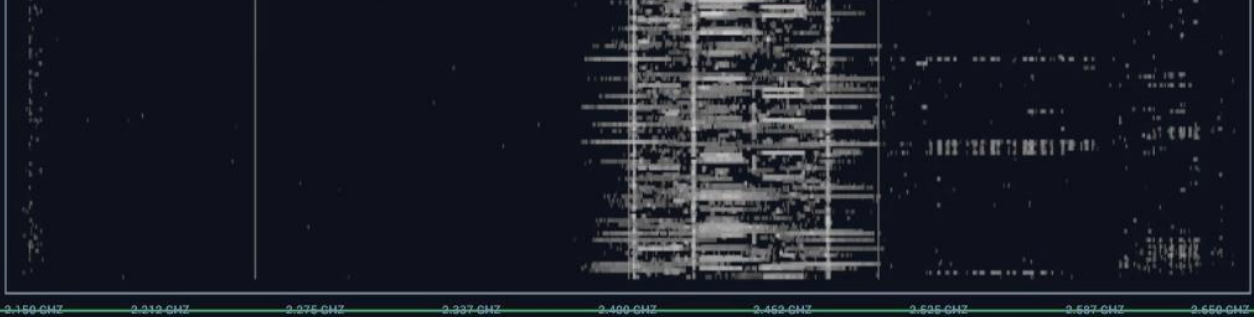
CH-0 2.400 GHZ 500.0 MSPS



Raw Spectrogram



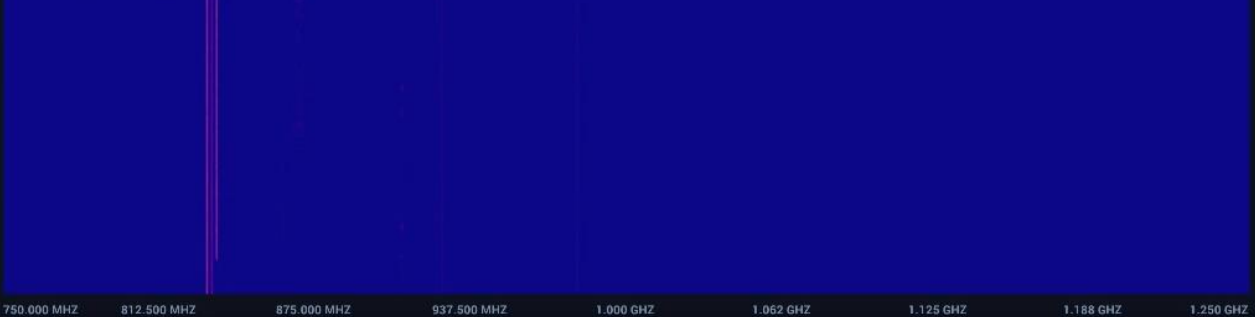
Detected Regions of Interest



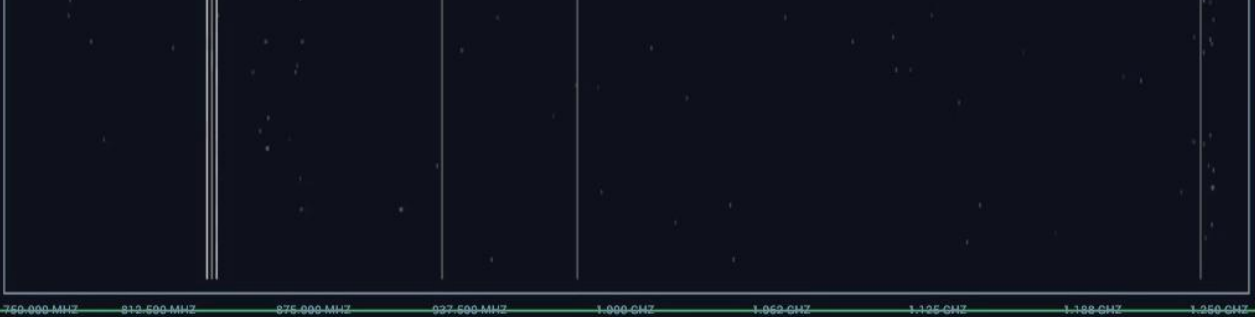
CH-1 1.000 GHZ 500.0 MSPS



Raw Spectrogram



Detected Regions of Interest



Channel Info

CH-0
Center 2.400 GHZ
Span 500.000 MHz
Det Bin 24.4 KHZ
Disp Px 260.4 KHZ
Time Bin 5.12 US
Processing Ratio 1:1 (100.0%)
Vis Ratio 8/512 (1.6%)
FFT 20480
History 161 frames

CH-1
Center 1.000 GHZ
Span 500.000 MHz
Det Bin 24.4 KHZ
Disp Px 260.4 KHZ
Time Bin 5.12 US
Processing Ratio 1:1 (100.0%)
Vis Ratio 8/512 (1.6%)
FFT 20480
History 161 frames

Display Controls

Overlay Detection Mask

GNU Radio and GPUs

A brief history...

- GPUs and SDRs go way back, but the first “big bang” was probably fosphor (and gr-fosphor) by Sylvain (OpenCL-based)
- Mike P. (among others!) talked about accelerating blocks with GPUs as early as 2017
- DARPA + BlackLynx bring us custom buffers, and with that comes gr-cuda
- Adam T. and his gang have been coming to GRCon for a while now, showing off new tools for use with SDRs
- Luigi keeps building crazy cool apps that build on GPUs (among other things)
- JohnS and Beecubed provide tutorials for gr-cuda
- I probably forgot loads of people

- Kudos, shoutouts, respect, and all the appropriate credit to all these folks!

The screenshot shows a GitHub repository page for 'gr_cudaCourse' by user 'jsallay'. The repository is public and has 1 branch and 0 tags. The commit history shows an initial commit by 'jsallay' last year, with 2 commits in total. The file list includes 'docker', 'gr-cudaCourse', 'lessons', 'LICENSE', and 'README.md', all with initial commits from last year. The repository description mentions 'Performance Killer #1: Data Copy' and 'CUDA Acceleration for GNU Radio'. A video thumbnail shows a person speaking, with text overlays: 'Performance Killer #1: Data Copy', 'Have to move the data to the card then retrieve the results', 'CUDA Acceleration for GNU Radio', and 'Data Processing Holoscan + CyberEther Pipeline'.

Performance Killer #1: Data Copy

Have to move the data to the card then retrieve the results

CUDA Acceleration for GNU Radio

Data Processing
Holoscan + CyberEther Pipeline

jsallay Initial Commit 5a934d8 · last year 2 Commits

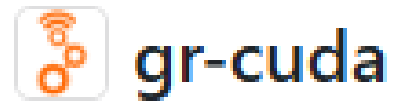
File	Commit	Time
docker	Initial Commit	last year
gr-cudaCourse	Initial Commit	last year
lessons	Initial Commit	last year
LICENSE	Initial commit	last year
README.md	Initial Commit	last year

gr-cuda ❤️ Daqiri

Options
Title: USRP SPECTRUM
Output Language: Python
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 500M

Variable
ID: center_freq
Value: 1G



<https://github.com/gnuradio/gr-cuda>



uhd_chdr_rx
Daqiri Configuration: ...yaml
Daqiri Interface: sdr_data
Complex Samples per Packet: 1.02...4k
Packets per Output Vector: 20
Output Vectors per Daqiri Burst: 125
RX Queue (-1 for all): 0
Empty Poll Interval (ms): 0

gpufft
FFT Size: 20.48k
Batch Size: 125
Direction: Forward

spectrum_magnitude
FFT Size: 20.48k
Number of Averages: 125

spectrum_log10
FFT Size: 20.48k
Epsilon: 1e-20

GPU | CPU

CUDA
Interoperability



Goal

- Move, process and visualize I/Q data (500 Msps). All on GPU!

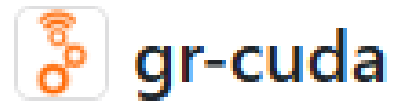
Current State

- Integrated DAQIRI into GNU Radio ecosystem
- Created FFT, magnitude and log conversion blocks (similar to Holoscan)
- Able to achieve 90% full functional GPU pipeline, “only” visualization missing

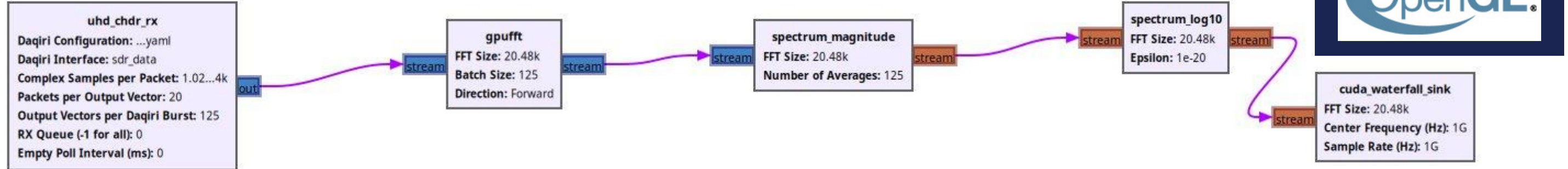
What's missing?

- Idea: visualization using CUDA interop (CUDA + OpenGL)
- Add remote streaming configuration as GNU radio block

gr-cuda ❤️ Daqiri



<https://github.com/gnuradio/gr-cuda>



Goal

- Move, process and visualize I/Q data (500 Msps). All on GPU!

Current State

- Integrated DAQIRI into GNU Radio ecosystem
- Created FFT, magnitude, log conversion and visualization blocks (similar to Holoscan)
- Able to achieve 100% full functional GPU pipeline with maximum I/Q data (1500Msps)

What's missing?

- Add remote streaming configuration as GNU radio block



That's all for now

- Bottom line: There are great ways for SDR/GPU integration
- This is all pretty new
- gr-cuda blocks: We still need to figure out open-sourcing mechanics
- Please pester us with questions!