Aerial Data Lake

Table of contents

List of Figures

6G will be artificial intelligence (AI) native. AI and machine learning (ML) will extend through all aspects of next generation networks from the radio, baseband processing, the network core including system management, orchestration and dynamic optimization processes. GPU hardware, together with programming frameworks will be essential to realize this vision of a software defined native-AI communication infrastructure.

The application of AI/ML in the physical layer has particularly been a hot research topic.

There is no AI without data. While the synthetic data generation capabilities of Aerial Omniverse Digital Twin (AODT) and Sionna/SionnaRT are essential aspects of a research project, availability of over-the-air (OTA) waveform data from real-time systems is equally important. This is the role of Aerial Data Lake. It is a data capture platform supporting the capture of OTA radio frequency (RF) data from virtual radio access network (vRAN) networks built on the Aerial CUDA-Accelerated RAN. Aerial Data Lake consists of a data capture application (app) running on the base station (BS) distributed unit (DU), a database of samples collected by the app, and an application programming interface (API) for accessing the database.

Target Audience

Industry and university researchers and developers looking to bring ML to the physical layer with the end goal of benchmarking on OTA testbeds like NVIDIA ARC-OTA or other GPU-based BSs.

Key Features

Aerial Data Lake has the following features:

Real-time capture of RF data from OTA testbed

- Aerial Data Lake is designed to operate with gnBs built on the Aerial CUDA-Accelerated RAN and that employ the Small Cell Forum FAPI interface between L2 and L1. One example system being the NVIDIA ARC-OTA network testbed. I/Q samples from O-RUs connected to the GPU platform via a O-RAN 7.2x split fronthaul interface are delivered to the host CPU and exported to the Aerial Data Lake database.

Aerial Data Lake APIs to access the RF database

- The data passed to the layer-2 via RX_Data.Indication and UL_TTI.Request are exported to the database. The fields in these data structures form the basis of the database access APIs.

Scalable and time coherent over arbitrary number of BSs

- The data collection app runs on the same CPU that supports the DU. It runs on a single core, and the database runs on free cores. Because each BS is responsible for collecting its own uplink data, the collection process scales as more BSs are added to the network testbed. Database entires are time-stamped so data collected over multiple BSs can be used in a training flow in a time-coherent manner.

Use in conjunction with pyAerial to generate training data for neural network physical layer designs

- Aerial Data Lake can be used in conjunction with the NVIDIA pyAerial CUDA-Accelerated Python L1 library. Using the Data Lake database APIs, pyAerial can access RF samples in a Data Lake database and transform those samples into training data for all the signal processing functions in an uplink or downlink pipeline.

Design

Aerial Data Lake sits beside the Aerial L1 and copies out data that would be useful for machine learning into an external database.

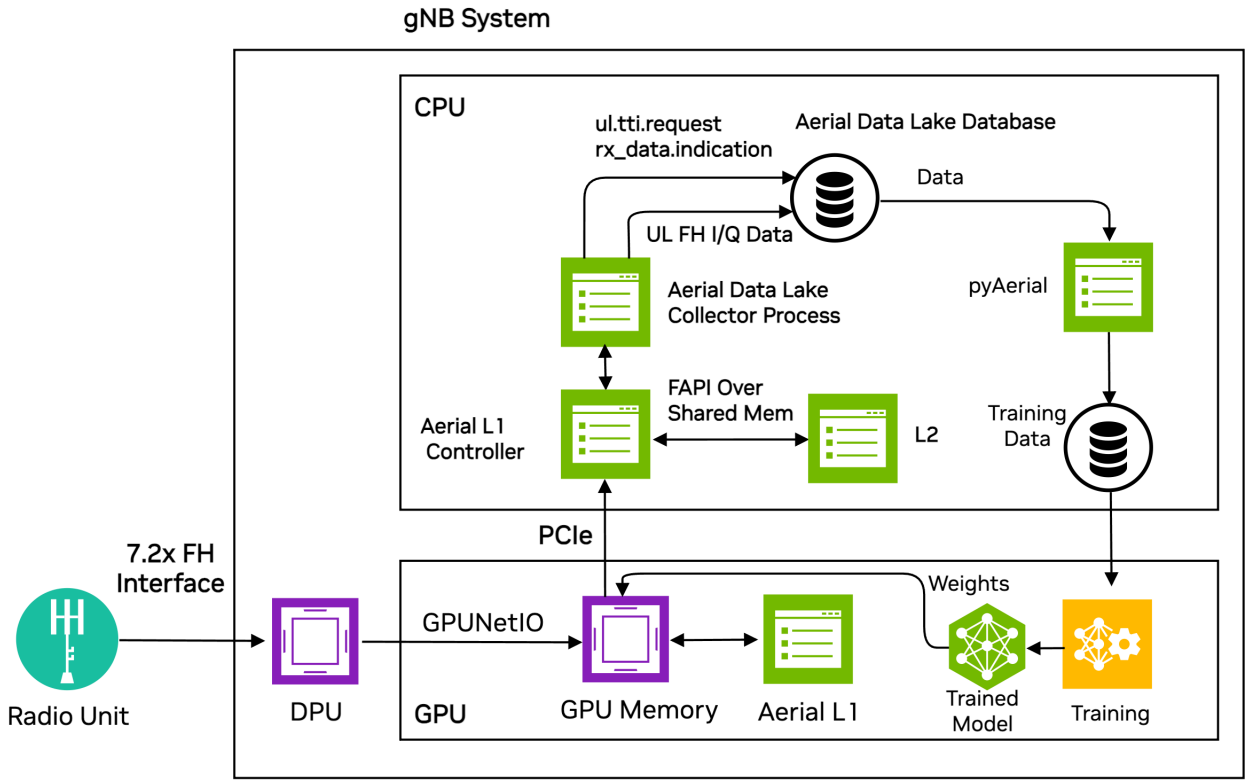

Figure 1: The Aerial Data Lake data capture platform as part of the gNB.

Uplink I/Q data from one or more O-RAN radio units (O-RUs) is delivered to GPU memory where it is both processed by the Aerial L1 PUSCH baseband pipeline and delivered to host CPU memory. The Aerial Data Lake collector process writes the I/Q samples to the Aerial Data Lake database in the *fh* table. The *fh* table has columns for SFN, Slot, IQ samples as *fhData*, and the start time of that SFN.slot as *TsTaiNs*.

The collector app saves data that the L2 sent to L1 to describe UL OTA transmissions in UL_TTI.Request messages as well as data returned to L2 the via RX_Data.Indication and CRC.Indication. This data is then written to the *fapi* database table. These messages and the fields within them are described in SCF 5G FAPI PHY Spec version 10.02, sections 3.4.3, 3.4.7, and 3.4.8.

Each gNB in a network testbed collects data from all O-RUs associated with it. That is, data collection over the span of a network is performed in a distributed manner, each gNB is building its own local database. Training can be performed locally at each gNB, and site-specific optimizations can be realized with this approach. Since the data in a database is time-stamped, the local databases can be consolidated at a centralized compute resource and training performed using the time aligned aggregated data. In cases where the aerial pusch pipeline was unable to decode due to channel conditions,

retransmissions can be used as ground truth as long as one of the retransmissions succeeds, allowing the user to test algorithms with better performance than the originals.

The Aerial Data Lake database storage requirements depend on the number of O-RUs, the antenna configuration of the O-RU, the carrier bandwidth, the TDD pattern and the number of samples to be collected. Collecting IQ samples of 1 million tranmissions from a single RU 4T4R O-RU employing a single 100MHz carrier will consume approximately 660 GB of storage.

Aerial Data Lake database comprises the fronthaul RF data. However, for many training applications access to data at other nodes in the receive pipeline is required. A pyAerial pipeline, together with the Data Lake database APIs, can access samples from an Aerial Data Lake database and transform that data into training data for any function in the pipeline.

Figure 2 illustrates data ingress from a Data Lake database into a pyAerial pipeline and using standard Python file I/O to generate training data for a soft de-mapper.

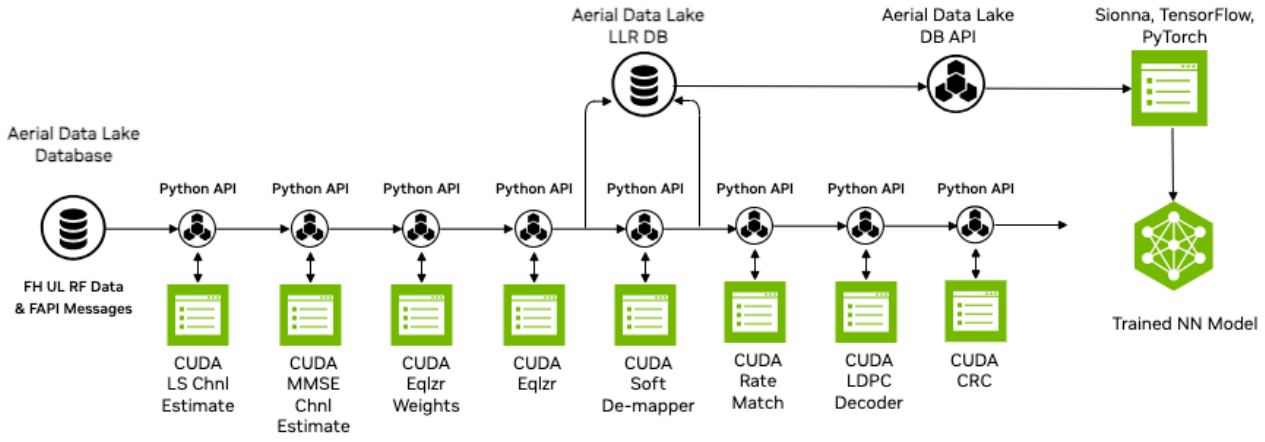

Figure 2: pyAerial is used in conjunction with the NVIDIA data collection platform, namely, Aerial Data Lake to build training data sets for any node in the layer-1 downlink or uplink signal processing pipeline. The example shows a Data Lake database of over-the-air samples transformed into training data for a neural network soft de-mapper.

Installation

Aerial Data Lake is compiled by default as part of cuphycontoller. If you would like to record fresh data every time cuphycontroller is started, see the section on Fresh Data.

Start by installing Clickhouse database on the server collecting the data. The command below will download and run an instance of the clickhouse server in a docker container.

```
docker run -d \ --network=host \ -v $(realpath ./ch_data):/var/lib/clickhouse/ \ -v $(realpath ./ch_logs):/var/log/clickhouse-server/ \ --cap-add=SYS_NICE --cap-add=NET_ADMIN --cap-add=IPC_LOCK \ --name my-clickhouse-server --ulimit nofile=262144:262144 clickhouse/clickhouse-server
```

By default clickhouse will not drop large tables, and will return an error if attempted. The clickhouse-cpp library does not return exceptions so to avoid what loooks like a cuphycontroller crash we recommend allowing it to drop large tables using the following command:

```
sudo touch './ch_data/flags/force_drop_table' && sudo chmod 666 './ch_data/flags/force_drop_table'
```

Usage

In the cuphycontoller adapter yaml configuration file, enable data collection by specifying a core then start cuphycontroller and usual. The core should be on the same NUMA node as the rest of cuphycontroller, i.e. should follow the same pattern as the rest of the cores An example of this can be found commented out in cuphycontroller_P5G_FXN_R750.yaml.

```
cuphydriver_config: # Fields added for data collection datalake_core: 19 # Core on which data collection runs. E.g isolated odd on R750, any isolated core on gigabyte datalake_address: localhost datalake_samples: 1000000 # Number of samples to collect for each UE/RNTI. Defaults to 1M
```

When enabled the *DataLake* object is created and *DataLake::dbInit()* initializes the two tables in the database. After cuphycontroller runs the PUSCH pipeline, cupycontroller calls *DataLake::notify()* with the addresses of the data to be saved, which *DataLake* then saves. When *DataLake::waitForLakeData* wakes up it calls *DataLake::dbInsert()* which appends data to respective *Clickhouse* columns, then sleeps waiting for more data. Once 20 PUSCH transmissions have been stored or a total of *datalake_samples* have been recived the columns are appended to a *Clickhouse::Block* and inserted into the respective table.

Using Data Lake in Notebooks

Follow pyAerial instructions and usual to build and launch that container. It must be run on a server with a GPU.

Two example notebooks for are included:
datalake_channel_estimation.ipynb performs channel estimation and plots the result
datalake_chan_estimation_decoding.ipynb goes futher and runs the full PUSCH decoding pipeline, both a fused version and a version build up up constituent parts

Notes

Database Administration

Clickhouse client

A clickhouse client is needed to interact with the server. To download it and run it do the following:

> curl https://clickhouse.com/ | sh ./clickhouse client aerial@aerial-gnb:~$ ./clickhouse client ClickHouse client version 24.3.1.1159 (official build). Connecting to localhost:9000 as user default. Connected to ClickHouse server version 24.3.1. aerial-gnb :)

You are now at the clickhouse client prompt. Commands starting with *aerial-gnb :)* are entered at this prompt and those with *$* are run on the host.

Database Import

There are example *fapi* and *fh* tables included in Aerial CUDA-Accelerated RAN. These tables can be imported into the clickhouse database by copying them to the clickhouse user_files folder, using the client to import them:

> $ docker cp c_aerial_$USER:/opt/nvidia/cuBB/pyaerial/notebooks/data/fh.parquet . $ docker cp c_aerial_$USER:/opt/nvidia/cuBB/pyaerial/notebooks/data/fapi.parquet . $ sudo cp *.parquet ./ch_data/user_files/ aerial-gnb :) create table fapi ENGINE =

```
MergeTree primary key TsTaiNs settings allow_nullable_key=1 as select * from file('fapi.parquet',Parquet) Ok. aerial-gnb :) create table fh ENGINE = MergeTree primary key TsTaiNs settings allow_nullable_key=1 as select * from file('fh.parquet',Parquet) Ok. aerial-gnb :) select table, formatReadableSize(sum(bytes)) as size from system.parts group by table SELECT `table`, formatReadableSize(sum(bytes)) AS size FROM system.parts GROUP BY `table` Query id: 95451ea7-6ea9-4eec-b297-15de78036ada
    table                                    size                fh    4.54 MiB       fapi    2.18 KiB
```

You now have five PUSCH transmissions loaded in the database and can run the example notebooks.

Database Queries

To show some information about the entries (rows) you can run the following:

```
# Show counts of transmissions for all RNTIs aerial-gnb :) select rnti, count(*) from fapi group by rnti SELECT rnti, count(*) FROM fapi GROUP BY rnti Query id: 76cf63d8-7302-4d73-972e-8ba7392da7ac       rnti      count()        55581    5
                                        # Show select information from all rows of the fapi table aerial-gnb :) from fapi select TsTaiNs,TsSwNs,SFN,Slot,pduData SELECT TsTaiNs, TsSwNs, SFN, Slot, pduData FROM fapi Query id: af362836-b379-46fd-85ae-0e9f62deb8ab
                                                 TsTaiNs                                                TsSwNs      SF
   2024-03-21 12:18:39.162000000    2024-03-21 12:18:39.162990534    192    4
[62,1,0,63,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,
      2024-03-21 12:18:39.187000000    2024-03-21 12:18:39.188086009    194    14
[1,36,192,1,0,1,58,12,191,0,166,41,62,128,2,191,0,46,0,6,128,128,120,136,120,127,128,
      2024-03-21 12:18:39.192000000    2024-03-21 12:18:39.194784691    195    4
[62,1,0,63,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,
      2024-03-21 12:18:39.252000000    2024-03-21 12:18:39.253086195    201    4
[1,3,0,2,0,61,0,57,63,51,63,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33,33
      2024-03-21 12:18:39.332000000    2024-03-21 12:18:39.332997301    209    4
[1,38,192,2,0,2,58,13,191,0,216,239,96,3,131,63,0,44,138,152,7,112,128,220,160,94,152
```

```
5 rows in set. Elapsed: 0.002 sec. #Show start times of fh table aerial-gnb :) from fh select TsTaiNs,TsSwNs,SFN,Slot SELECT TsTaiNs, TsSwNs, SFN, Slot FROM fh Query id: 078d451a-5db9-4f35-b890-96b2c561fdbe
                                    TsTaiNs                                    TsSwNs      SF
  2024-03-21 12:18:39.162000000    2024-03-21 12:18:39.162990534    192    4
2024-03-21 12:18:39.187000000    2024-03-21 12:18:39.188086009    194    14
2024-03-21 12:18:39.192000000    2024-03-21 12:18:39.194784691    195    4
2024-03-21 12:18:39.252000000    2024-03-21 12:18:39.253086195    201    4
2024-03-21 12:18:39.332000000    2024-03-21 12:18:39.332997301    209    4

5 rows in set. Elapsed: 0.002 sec.
```

Fresh Data

The database of IQ samples grows quite quickly. If you want fresh data every run the tables can be dropped automatically by uncommenting these lines in cuPHY-CP/data_lakes/data_lakes.cpp:

```
//dbClient->Execute("DROP TABLE IF EXISTS fapi"); //dbClient->Execute("DROP TABLE IF EXISTS fh");
```

Dropping Data

You can manually drop all of the data from the database with these commands:

```
aerial-gnb :) drop table fh Ok. aerial-gnb :) drop table fapi Ok.
```

Jupyter notebooks

Exceptions are not always displayed in jupyter notebooks the way that it would be if a python script had been run, so in some cases it can be easier to convert the notebook to a script and run that.

```
jupyter nbconvert --to script <notebook_name>.ipynb
```

To interact with the data and code in place, specific lines can be debugged by adding *breakpoint()* inline

Known Limitations

Currently datalakes records the first UE per TTI and has been tested with a single cell per gNB as supported by the Open Air Interface L2+ stack.