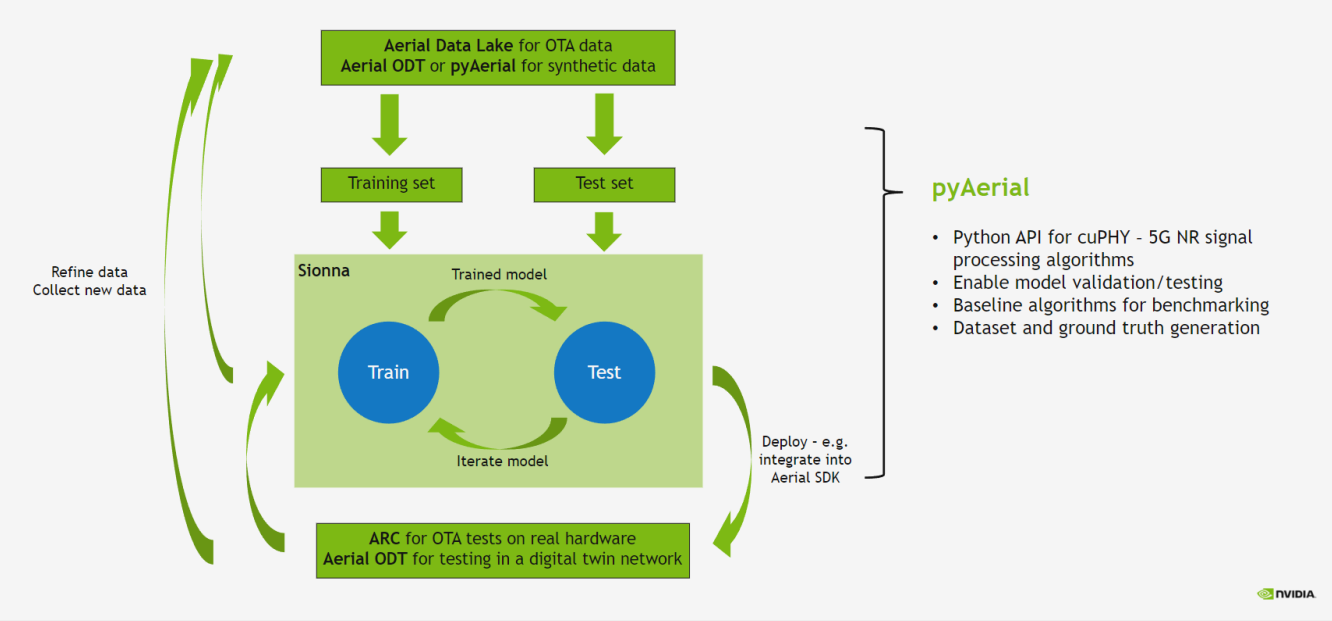
LLRNet: Dataset generation

Table of contents

The wireless ML design flow using Aerial is depicted in the figure below.



In this notebook, we take data generated in the Using pyAerial for data generation by simulation example and generate a dataset for training LLRNet using pyAerial. **Note that the data is assumed to have been generated prior to running this notebook.**

LLRNet, published in

15. Shental, J. Hoydis, "*'Machine LLRning': Learning to Softly Demodulate*", https://arxiv.org/abs/1907.01512

is a simple neural network model that takes equalizer outputs, i.e. the complex-valued equalized symbols, as its input and outputs the corresponding log-likelihood ratios (LLRs) for each bit. This model is used to demonstrate the whole ML design flow using Aerial, from capturing the data to deploying the model into 5G NR PUSCH receiver, replacing the conventional soft demapper in cuPHY. In this notebook a dataset is generated. We use pyAerial to call cuPHY functionality to get equalized symbols out for pre-captured/-simulated Rx data, as well as the corresponding log-likelihood ratios from a conventional soft demapper.

Imports

[1]:

```
import os os.environ["CUDA_VISIBLE_DEVICES"] = "0" from cuda import cudart import numpy as np import pandas as pd from tqdm.notebook import tqdm from IPython.display import Markdown from IPython.display import display from aerial.phy5g.algorithms import ChannelEstimator from aerial.phy5g.algorithms import NoiseIntfEstimator from aerial.phy5g.algorithms import ChannelEqualizer from aerial.phy5g.algorithms import Demapper from aerial.phy5g.ldpc import LdpcDeRateMatch from aerial.phy5g.ldpc import LdpcDecoder from aerial.phy5g.ldpc import code_block_desegment from aerial.util.data import PuschRecord from aerial.util.data import load_pickle from aerial.util.data import save_pickle from aerial.util.fapi import dmrs_fapi_to_bit_array import warnings warnings.filterwarnings("error")
```

Load the source data

The source data can be either real data collected from an over the air setup, or synthetic data generated by simulation.

Note: This notebook uses data generated using this notebook: Using pyAerial for data generation by simulation, which needs to be run before this notebook.

[2]:

```
# This is the source data directory which is assumed to contain the source data.
DATA_DIR = "data/" source_dataset_dir = DATA_DIR + "example_simulated_dataset/QPSK/" # This is the target dataset directory. It gets created if it does not exist. target_dataset_dir = DATA_DIR + "example_llrnet_dataset/QPSK/" os.makedirs(target_dataset_dir, exist_ok=True) # Load the main data file. try: df = pd.read_parquet(source_dataset_dir + "l2_metadata.parquet", engine="pyarrow") except FileNotFoundError: display(Markdown("**Data not found - has example_simulated_dataset.ipynb been run?**")) print(f"Loaded{df.shape[0]}PUSCH records.")
```

```
Loaded 12000 PUSCH records.
```

Dataset generation

Here, pyAerial is used to run channel estimation, noise/interference estimation and channel equalization to get the equalized symbols, corresponding to the LLRNet input, as well as the log-likelihood ratios, corresponding to the LLRNet target output.

[3]:

```
cuda_stream = cudart.cudaStreamCreate()[1] # Take modulation order from the first record. The assumption is that all # entries have the same modulation order here.
mod_order = df.loc[0].qamModOrder # These hard-coded too. num_rx_ant = 2
enable_pusch_tdi=1 eq_coeff_algo=1 # Create the PUSCH Rx components for extracting the equalized symbols and log-likelihood ratios. channel_estimator =
ChannelEstimator( num_rx_ant=num_rx_ant, cuda_stream=cuda_stream )
noise_intf_estimator = NoiseIntfEstimator( num_rx_ant=num_rx_ant,
eq_coeff_algo=eq_coeff_algo, cuda_stream=cuda_stream ) channel_equalizer =
ChannelEqualizer( num_rx_ant=num_rx_ant, eq_coeff_algo=eq_coeff_algo,
enable_pusch_tdi=enable_pusch_tdi, cuda_stream=cuda_stream) derate_match =
LdpcDeRateMatch(enable_scrambling=True, cuda_stream=cuda_stream) demapper
= Demapper(mod_order=mod_order) decoder =
LdpcDecoder(cuda_stream=cuda_stream) # Loop through the PUSCH records and create new ones. pusch_records = [] tb_errors = [] snrs = [] for pusch_record in (pbar
:= tqdm(df.itertuples(index=False), total=df.shape[0])):
pbar.set_description("Running cuPHY to get equalized symbols and log-likelihood ratios...") num_ues = 1 start_prb = pusch_record.rbStart num_prbs =
pusch_record.rbSize start_sym = pusch_record.StartSymbolIndex num_symbols =
pusch_record.NrOfSymbols dmrs_sym =
dmrs_fapi_to_bit_array(pusch_record.ulDmrsSymbPos) dmrs_scrm_id =
pusch_record.ulDmrsScramblingId dmrs_max_len = 1 dmrs_add_ln_pos = 1
num_dmrs_cdm_grps_no_data = pusch_record.numDmrsCdmGrpsNoData
num_layers = pusch_record.nrOfLayers scid = pusch_record.SCID dmrs_ports =
pusch_record.dmrsPorts slot = pusch_record.Slot tbs = len(pusch_record.macPdu)
code_rate = pusch_record.targetCodeRate / 10240. rv = 0 ndi = 1 rnti =
pusch_record.RNTI data_scrm_id = pusch_record.dataScramblingId ref_tb =
pusch_record.macPdu # Just making sure the hard-coded value is correct. assert
mod_order == pusch_record.qamModOrder # Load received IQ samples.
```

rx_iq_data_filename = source_dataset_dir + pusch_record.rx_iq_data_filename rx_slot = load_pickle(rx_iq_data_filename) num_rx_ant = rx_slot.shape[2] *# Load user data.* user_data_filename = source_dataset_dir + pusch_record.user_data_filename user_data = load_pickle(user_data_filename) *# Run the channel estimation (cuPHY).* ch_est = channel_estimator.estimate(rx_slot=rx_slot, num_ues=num_ues, slot=slot, num_dmrs_cdm_grps_no_data=num_dmrs_cdm_grps_no_data, dmrs_scrm_id=dmrs_scrm_id, start_prb=start_prb, num_prbs=num_prbs, dmrs_syms=dmrs_sym, dmrs_max_len=dmrs_max_len, dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym, num_symbols=num_symbols, scids=[scid], layers=[num_layers], dmrs_ports= [dmrs_ports]) *# Run noise/interference estimation (cuPHY), needed for equalization.* lw_inv, noise_var_pre_eq = noise_intf_estimator.estimate(rx_slot=rx_slot, channel_est=ch_est, num_ues=num_ues, slot=slot, num_dmrs_cdm_grps_no_data=num_dmrs_cdm_grps_no_data, dmrs_scrm_id=dmrs_scrm_id, start_prb=start_prb, num_prbs=num_prbs, dmrs_syms=dmrs_sym, dmrs_max_len=dmrs_max_len, dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym, num_symbols=num_symbols, scids=[scid], layers=[num_layers], dmrs_ports= [dmrs_ports]) *# Run equalization and mapping to log-likelihood ratios.* llr, equalized_sym = channel_equalizer.equalize(rx_slot=rx_slot, channel_est=ch_est, lw_inv=lw_inv, noise_var_pre_eq=noise_var_pre_eq, num_ues=num_ues, num_dmrs_cdm_grps_no_data=num_dmrs_cdm_grps_no_data, start_prb=start_prb, num_prbs=num_prbs, dmrs_syms=dmrs_sym, dmrs_max_len=dmrs_max_len, dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym, num_symbols=num_symbols, layers=[num_layers], mod_orders=[mod_order]) ree_diag_inv = channel_equalizer.ree_diag_inv[0] *# Just pick one (first) symbol from each PUSCH record for the LLRNet dataset. # This is simply to reduce the size of the dataset - training LLRNet does not # require a lot of data.* user_data["llrs"] = llr[0] [:mod_order, 0, :, 0] user_data["eq_syms"] = equalized_sym[0][0, :, 0] map_llrs = demapper.demap(equalized_sym[0][0, :, 0], ree_diag_inv[0, ...]) user_data["map_llrs"] = map_llrs *# Save pickle files for the target dataset.* rx_iq_data_fullpath = target_dataset_dir + pusch_record.rx_iq_data_filename user_data_fullpath = target_dataset_dir + pusch_record.user_data_filename save_pickle(data=rx_slot, filename=rx_iq_data_fullpath) save_pickle(data=user_data, filename=user_data_fullpath) pusch_records.append(pusch_record) ##

```
# Run through the rest of the receiver pipeline to verify that this was legit LLR data. # De-rate matching and descrambling.
cinit = (rnti << 15) + data_scrm_id
num_data_sym = (np.array(dmrs_sym[start_sym:start_sym + num_symbols]) == 0).sum()
rate_match_len = num_data_sym * mod_order * num_prbs * 12 * num_layers
coded_blocks = derate_match.derate_match( input_data=llr[0], tb_size=tbs * 8, code_rate=code_rate, rate_match_len=rate_match_len, mod_order=mod_order, num_layers=num_layers, redundancy_version=rv, ndi=ndi, cinit=cinit )
# LDPC decoding of the derate matched blocks.
code_blocks = decoder.decode( input_llr=coded_blocks, tb_size=tbs * 8, code_rate=code_rate, redundancy_version=rv, rate_match_len=rate_match_len )
# Combine the code blocks into a transport block.
tb = code_block_desegment( code_blocks=code_blocks, tb_size=tbs * 8, code_rate=code_rate, return_bits=False )
tb_errors.append(not np.array_equal(tb[:tbs], ref_tb[:tbs]))
snrs.append(user_data["snr"])
```

[4]:

```
print("Saving...")
df_filename = os.path.join(target_dataset_dir, "l2_metadata.parquet")
df = pd.DataFrame.from_records(pusch_records, columns=PuschRecord._fields)
df.to_parquet(df_filename, engine="pyarrow")
print("All done!")
```

```
Saving...
All done!
```