Using pyAerial for PUSCH decoding on Aerial Data Lake data

Table of contents

List of Figures

This example shows how to use the pyAerial bindings to run cuPHY GPU accelerated PUSCH decoding for 5G NR PUSCH. The 5G NR PUSCH data is read from an example over the air captured PUSCH dataset collected and stored using Aerial Data Lake. Building a PUSCH receiver using pyAerial is demonstrated in two ways, first by using a fully fused, complete, PUSCH receiver called from Python using just a single function call. The same is then achieved by building the complete PUSCH receiver using individual separate Python function calls to individual PUSCH receiver components.

Note: This example requires that the clickhouse server is running and that the example data has been stored in the database. Refer to the Aerial Data Lake documentation on how to do this.

Imports

[1]:

```
import math import os os.environ["CUDA_VISIBLE_DEVICES"] = "0" import numpy as np import pandas as pd from IPython.display import Markdown from IPython.display import display # Connecting to clickhouse on remote server import clickhouse_connect # Plotting with Bokeh. import matplotlib.pyplot as plt # pyAerial imports from aerial.phy5g.algorithms import ChannelEstimator from aerial.phy5g.algorithms import ChannelEqualizer from aerial.phy5g.algorithms import NoiseIntfEstimator 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.phy5g.pusch import PuschRx from aerial.util.cuda import get_cuda_stream from aerial.util.fapi import dmrs_fapi_to_bit_array # Hide log10(10) warning _ = np.seterr(divide='ignore', invalid='ignore')
```

Create the PUSCH pipelines

This is a PUSCH receiver pipeline made up of separately called pyAerial PUSCH receiver components.

[2]:

```
# Whether to plot intermediate results within the PUSCH pipeline, such as channel
estimates and equalized symbols. plot_figures = True num_ues = 1 num_tx_ant = 2 #
UE antennas num_rx_ant = 4 # gNB antennas cell_id = 41 # Physical cell ID
enable_pusch_tdi = 0 # Enable time interpolation for equalizer coefficients
eq_coeff_algo = 1 # Equalizer algorithm # The PUSCH receiver chain built from
separately called pyAerial Python components is defined here. class PuschRxSeparate:
"""PUSCH receiver class. This class encapsulates the whole PUSCH receiver chain
built using pyAerial components. """ def __init__(self, num_rx_ant, enable_pusch_tdi,
eq_coeff_algo, plot_figures): """Initialize the PUSCH receiver.""" self.cuda_stream =
get_cuda_stream() # Build the components of the receiver. self.channel_estimator =
ChannelEstimator( num_rx_ant=num_rx_ant, cuda_stream=self.cuda_stream)
self.channel_equalizer = ChannelEqualizer( num_rx_ant=num_rx_ant,
enable_pusch_tdi=enable_pusch_tdi, eq_coeff_algo=eq_coeff_algo,
cuda_stream=self.cuda_stream) self.noise_intf_estimator = NoiseIntfEstimator(
num_rx_ant=num_rx_ant, eq_coeff_algo=eq_coeff_algo,
cuda_stream=self.cuda_stream) self.derate_match = LdpcDeRateMatch(
enable_scrambling=True, cuda_stream=self.cuda_stream) self.decoder =
LdpcDecoder(cuda_stream=self.cuda_stream) # Whether to plot the intermediate
results. self.plot_figures = plot_figures def run( self, rx_slot, num_ues, slot,
num_dmrs_cdm_grps_no_data, dmrs_scrm_id, start_prb, num_prbs, dmrs_syms,
dmrs_max_len, dmrs_add_ln_pos, start_sym, num_symbols, scids, layers,
dmrs_ports, rntis, data_scids, code_rates, mod_orders, tb_sizes ): """Run the
receiver.""" # Channel estimation. ch_est = self.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_syms, dmrs_max_len=dmrs_max_len,
dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym,
num_symbols=num_symbols, scids=scids, layers=layers, dmrs_ports=dmrs_ports ) #
Noise and interference estimation. lw_inv, noise_var_pre_eq =
self.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_syms, dmrs_max_len=dmrs_max_len,
dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym,
num_symbols=num_symbols, scids=scids, layers=layers, dmrs_ports=dmrs_ports ) #
Channel equalization and soft demapping. The first return value are the LLRs, # second
are the equalized symbols. We only want the LLRs now. llr,sym =
self.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_syms, dmrs_max_len=dmrs_max_len,
dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym,
num_symbols=num_symbols, layers=layers, mod_orders=mod_orders ) if
self.plot_figures: fig, axs = plt.subplots(1,4) for ant in range(4):
axs[ant].imshow(10*np.log10(np.abs(rx_slot[:, :, ant]**2)), aspect='auto')
axs[ant].set_ylim([pusch_record.rbStart * 12, pusch_record.rbSize * 12])
axs[ant].set_title('Ant ' + str(ant)) axs[ant].set(xlabel='Symbol', ylabel='Resource
Element') axs[ant].label_outer() fig.suptitle('Power in RU Antennas') fig, axs =
plt.subplots(1,2) axs[0].scatter(rx_slot.reshape(-1).real, rx_slot.reshape(-1).imag)
axs[0].set_title("Pre-Equalized samples") axs[0].set_aspect('equal')
axs[1].scatter(np.array(sym).reshape(-1).real, np.array(sym).reshape(-1).imag)
axs[1].set_title("Post-Equalized samples") axs[1].set_aspect('equal') fig, axs =
plt.subplots(1) axs.set_title("Channel estimates from the PUSCH pipeline") for ant in
range(4): axs.plot(np.abs(ch_est[0][ant, 0, :, 0])) axs.legend(["Rx antenna 0,
estimate", "Rx antenna 1, estimate", "Rx antenna 2, estimate", "Rx antenna 3,
estimate"]) axs.grid(True) plt.show() decoded_tbs = [] num_data_sym =
(np.array(dmrs_syms[start_sym:start_sym + num_symbols]) == 0).sum() for ue_idx in
range(num_ues): # De-rate matching and descrambling. cinit = (rntis[ue_idx] << 15) +
data_scids[ue_idx] # Scrambling init value. rate_match_len = num_data_sym *
mod_orders[ue_idx] * num_prbs * 12 * layers[ue_idx] coded_blocks =
self.derate_match.derate_match( input_data=llr[0][:, [ue_idx], ...],
tb_size=tb_sizes[ue_idx] * 8, code_rate=code_rates[ue_idx] / 1024.,
rate_match_len=rate_match_len, mod_order=mod_orders[ue_idx],
num_layers=layers[ue_idx], redundancy_version=0, ndi=1, cinit=cinit, ) # LDPC
decoding of the derate matched blocks. code_blocks = self.decoder.decode(
input_llr=coded_blocks, tb_size=tb_sizes[ue_idx] * 8, code_rate=code_rates[ue_idx] /
1024., redundancy_version=0, rate_match_len=rate_match_len, ) # Combine the code
blocks into a transport block. tb = code_block_desegment( code_blocks=code_blocks,
tb_size=tb_sizes[ue_idx] * 8, code_rate=code_rates[ue_idx] / 1024.,
return_bits=False, ) decoded_tbs.append(tb) return decoded_tbs pusch_rx_separate = PuschRxSeparate( num_rx_ant=num_rx_ant, enable_pusch_tdi=enable_pusch_tdi, eq_coeff_algo=eq_coeff_algo, plot_figures=plot_figures ) # This is the fully fused PUSCH receiver chain. pusch_rx = PuschRx( cell_id=cell_id, num_rx_ant=num_rx_ant, num_tx_ant=num_rx_ant, enable_pusch_tdi=enable_pusch_tdi, eq_coeff_algo=eq_coeff_algo )
```

Querying the database

Below shows how to connect to the clickhouse database and querying the data from it.

[3]:

```
# Connect to the local database client = clickhouse_connect.get_client(host='localhost') # Pick a packet from the database pusch_records = client.query_df('select * from fapi where mcsIndex != 0 order by TsTaiNs limit 10')
```

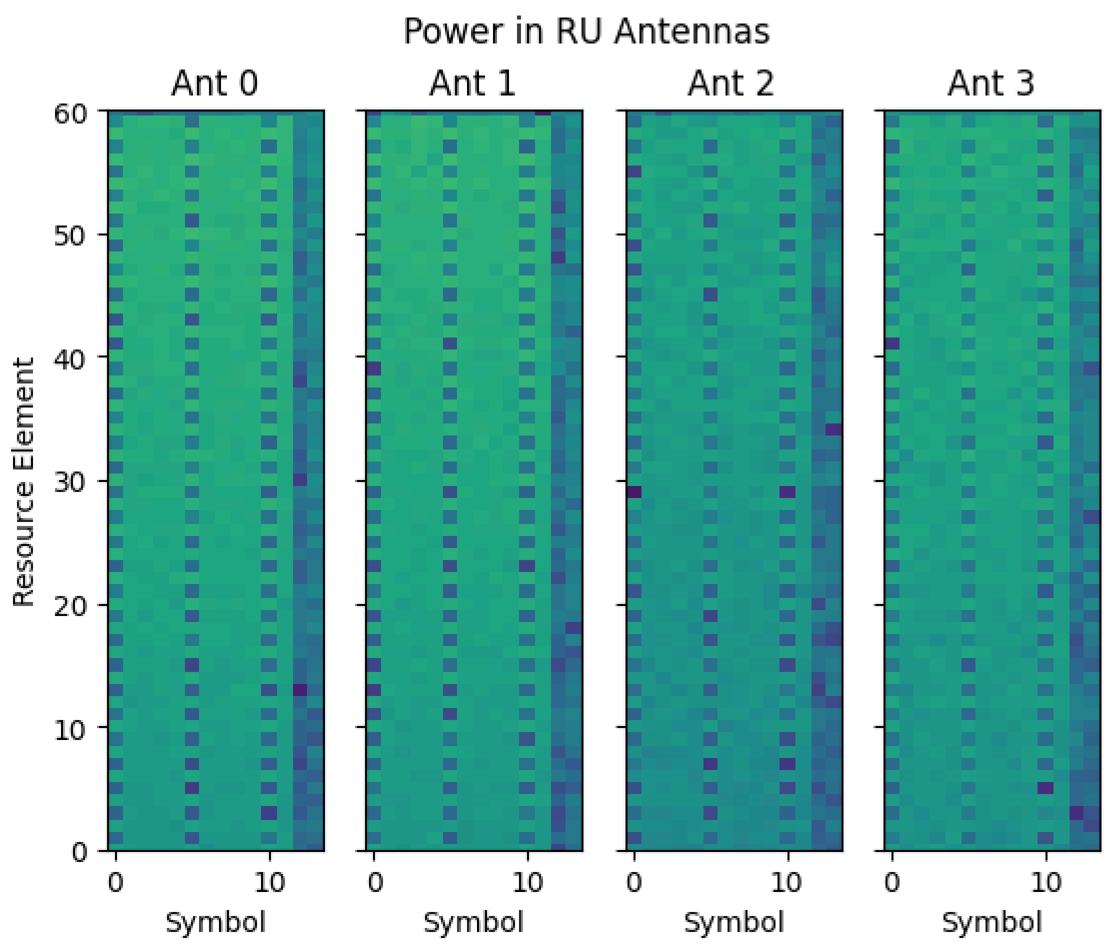
Extract the PUSCH parameters and run the pipelines

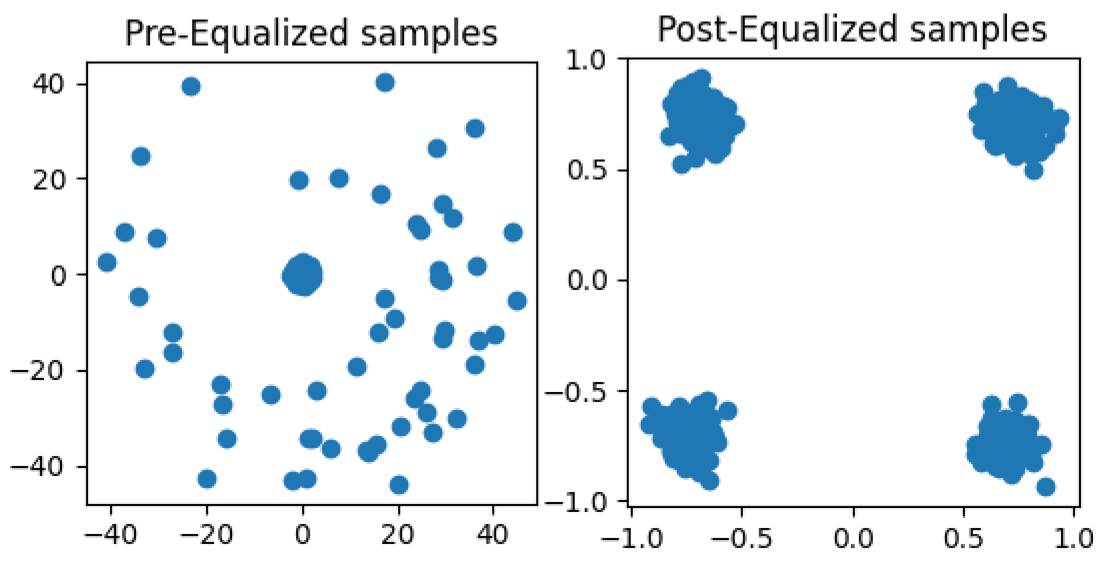
[4]:

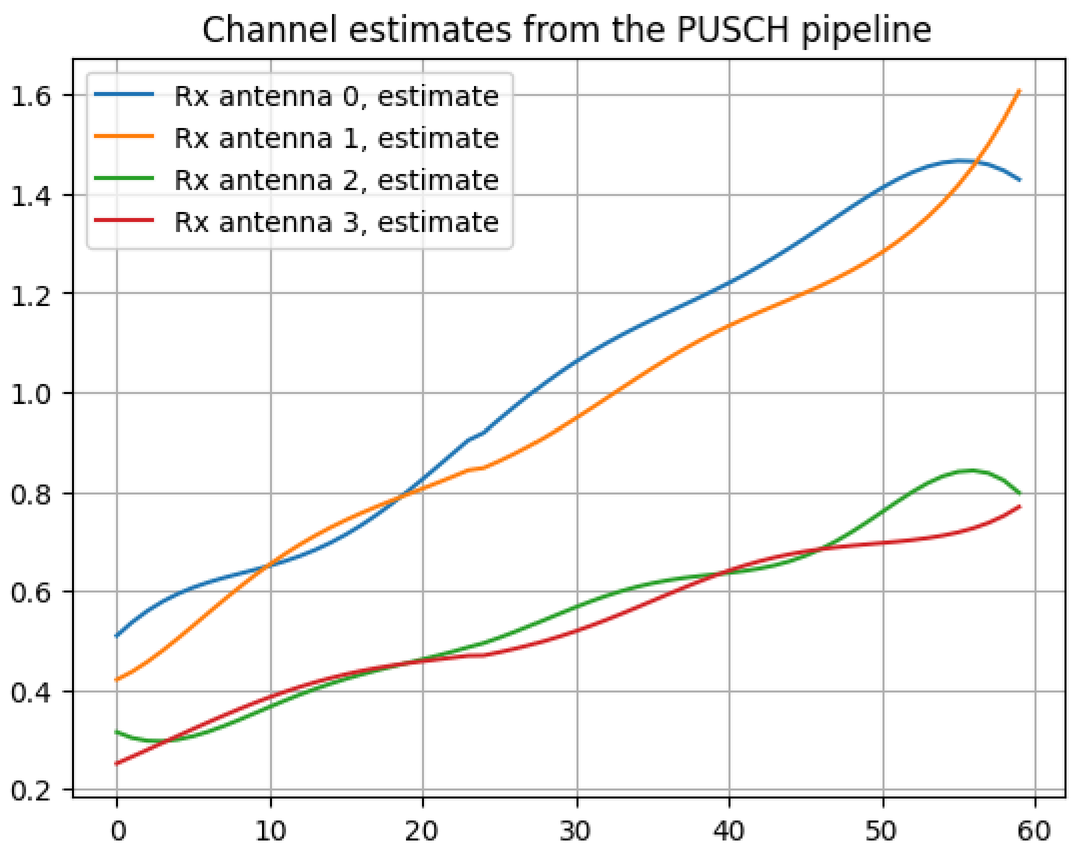
```
for index, pusch_record in pusch_records.iterrows(): query = f"""select TsTaiNs,fhData from fh where TsTaiNs =={pusch_record.TsTaiNs.timestamp()} """ fh = client.query_df(query) display(Markdown("### Example{}- SFN.Slot{}.{}from time{}" .format(index + 1, pusch_record.SFN, pusch_record.Slot, pusch_record.TsTaiNs ))) # Make sure that the fronthaul database is complete for the SFN.Slot we've chosen if fh.index.size < 1: pusch_records = pusch_records.drop(index) continue; fh_samp = np.array(fh['fhData'][0], dtype=np.float32) rx_slot = np.swapaxes(fh_samp.view(np.complex64).reshape(4, 14, 273 * 12), 2, 0) # Extract all the needed parameters from the PUSCH record. slot = int(pusch_record.Slot) rntis = [pusch_record.rnti] layers = [pusch_record.nrOfLayers] start_prb = pusch_record.rbStart num_prbs = pusch_record.rbSize start_sym = pusch_record.StartSymbolIndex num_symbols = pusch_record.NrOfSymbols scids = [int(pusch_record.SCID)] data_scids = [pusch_record.dataScramblingId]
```

```
dmrs_scrm_id = pusch_record.ulDmrsScramblingId num_dmrs_cdm_grps_no_data =
pusch_record.numDmrsCdmGrpsNoData dmrs_syms =
dmrs_fapi_to_bit_array(int(pusch_record.ulDmrsSymbPos)) dmrs_ports =
[pusch_record.dmrsPorts] dmrs_max_len = 1 dmrs_add_ln_pos = 2 mcs_tables =
[pusch_record.mcsTable] mcs_indices = [pusch_record.mcsIndex] coderates =
[pusch_record.targetCodeRate / 10.] tb_sizes = [pusch_record.TBSize] mod_orders =
[pusch_record.qamModOrder] tb_input = np.array(pusch_record.pduData) # Run the
receiver built from separately called components. tbs = pusch_rx_separate.run(
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_syms, dmrs_max_len=dmrs_max_len,
dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym,
num_symbols=num_symbols, scids=scids, layers=layers, dmrs_ports=dmrs_ports,
rntis=rntis, data_scids=data_scids, code_rates=coderates, mod_orders=mod_orders,
tb_sizes=tb_sizes ) if np.array_equal(tbs[0][:tb_input.size], tb_input):
display(Markdown("**Separated kernels PUSCH decoding success** for SFN.Slot{}.
{}from time{}".format(pusch_record.SFN, pusch_record.Slot, pusch_record.TsTaiNs)))
else: display(Markdown("**Separated kernels PUSCH decoding failure**"))
print("Output bytes:") print(tbs[0][:tb_input.size]) print("Expected output:")
print(tb_input) # Run the fused PUSCH receiver. # Note that this is where we set the
dynamically changing parameters. tb_crcs, tbs = pusch_rx.run( 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_syms, dmrs_max_len=dmrs_max_len,
dmrs_add_ln_pos=dmrs_add_ln_pos, start_sym=start_sym,
num_symbols=num_symbols, scids=scids, layers=layers, dmrs_ports=dmrs_ports,
rntis=rntis, data_scids=data_scids, code_rates=coderates, mod_orders=mod_orders,
tb_sizes=tb_sizes ) if np.array_equal(tbs[0][:tb_input.size], tb_input):
display(Markdown("**Fused PUSCH decoding success** for SFN.Slot{}.{}from
time{}".format(pusch_record.SFN, pusch_record.Slot, pusch_record.TsTaiNs))) else:
display(Markdown("**Fused PUSCH decoding failure**")) print("Output bytes:")
print(tbs[0][:tb_input.size]) print("Expected output:") print(tb_input)
```

Example 1 - SFN.Slot 192.4 from time 2024-03-21 12:18:39.162000



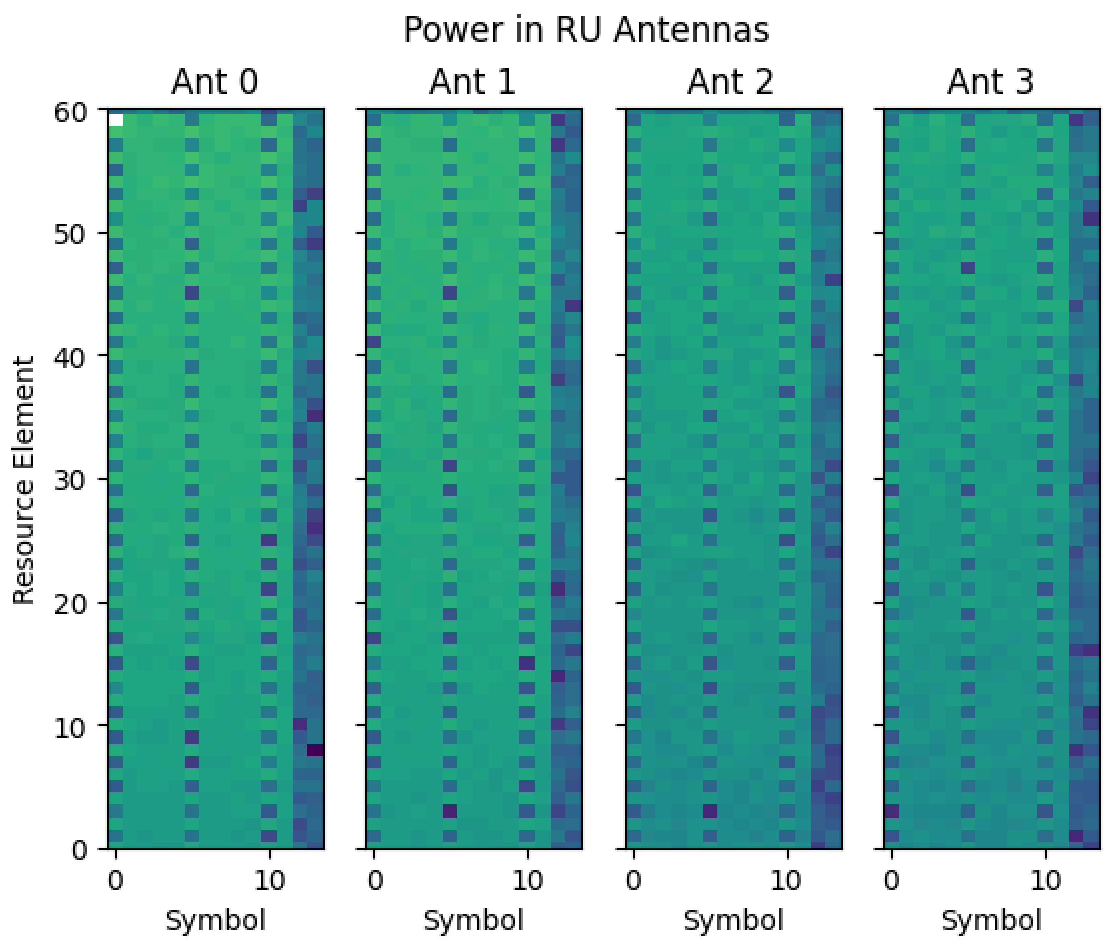


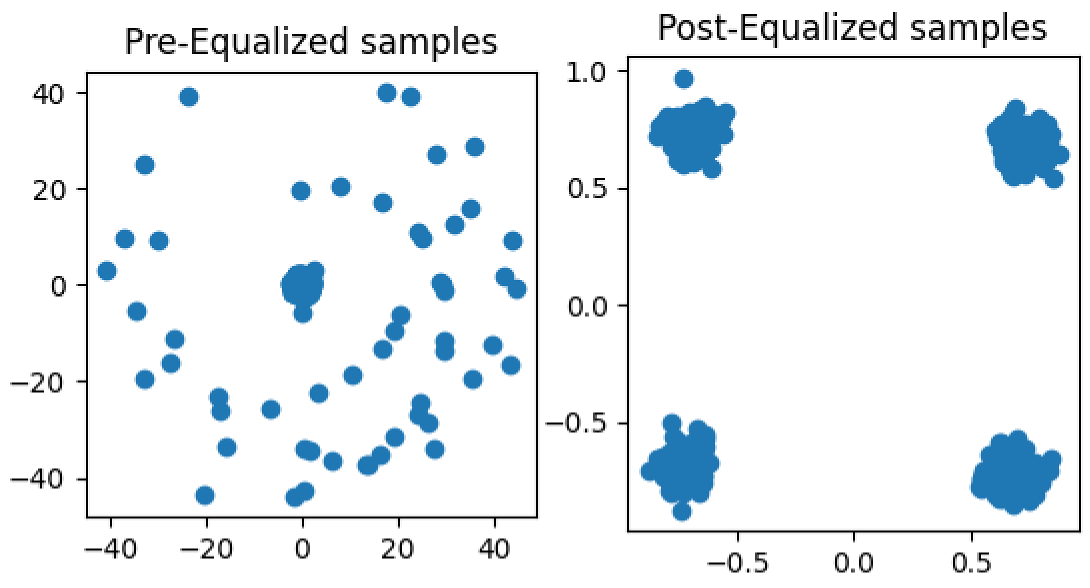


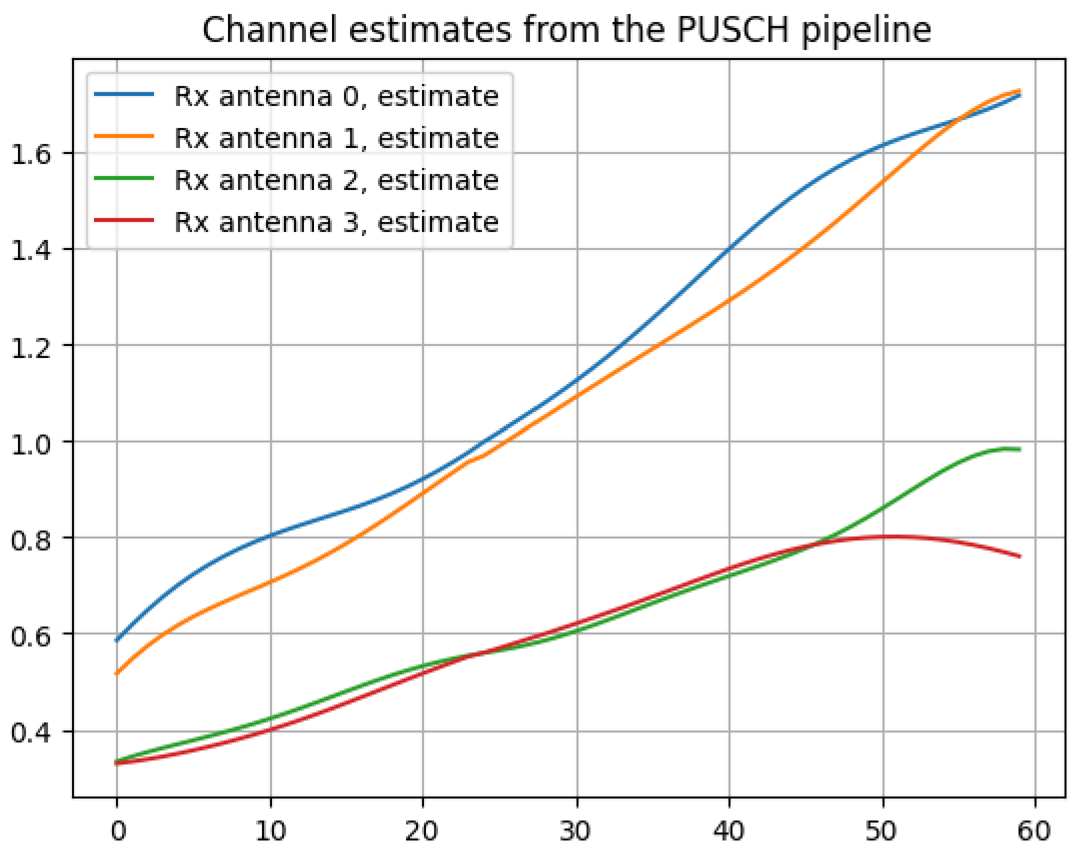
Separated kernels PUSCH decoding success for SFN.Slot 192.4 from time 2024-03-21 12:18:39.162000

Fused PUSCH decoding success for SFN.Slot 192.4 from time 2024-03-21 12:18:39.162000

Example 2 - SFN.Slot 194.14 from time 2024-03-21 12:18:39.187000


Power in RU Antennas
Ant 0
Ant 1
Ant 2
Ant 3
Resource Element
Symbol
0
10
20
30
40
50
60

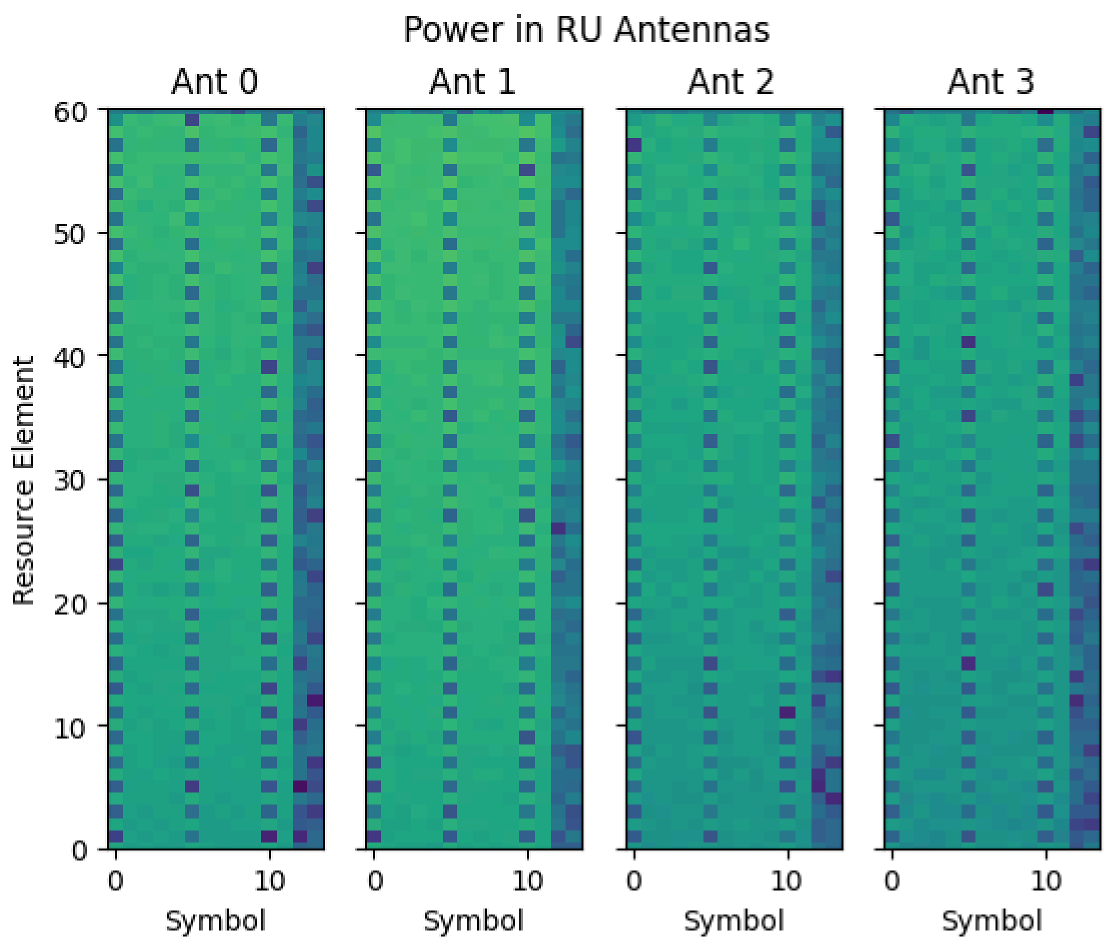

Pre-Equalized samples
Post-Equalized samples
40
20
0
−20
−40
1.0
0.5
0.0
−0.5



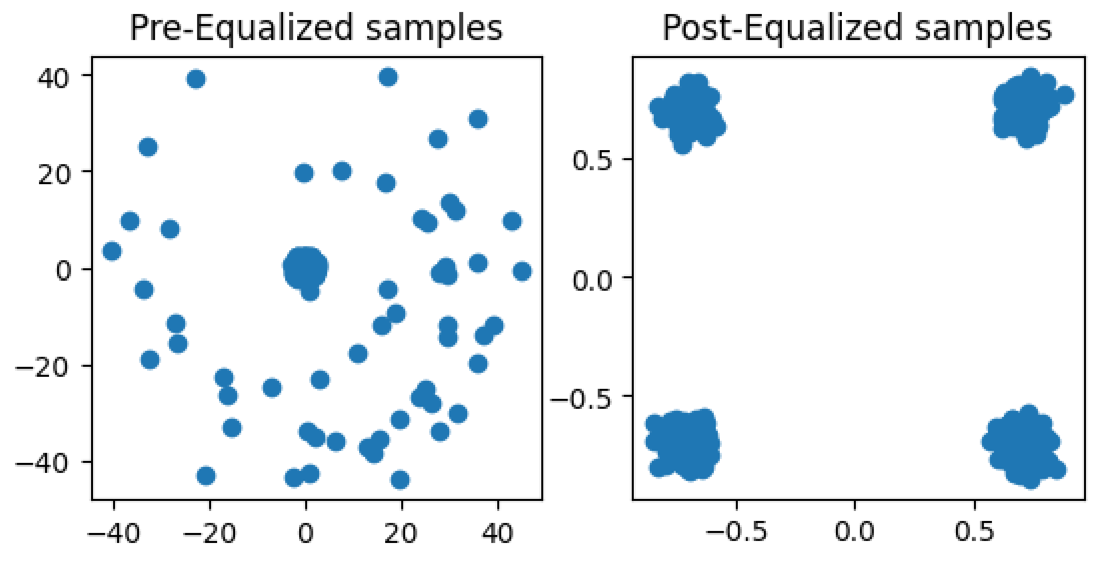
Separated kernels PUSCH decoding success for SFN.Slot 194.14 from time 2024-03-21 12:18:39.187000

Fused PUSCH decoding success for SFN.Slot 194.14 from time 2024-03-21 12:18:39.187000

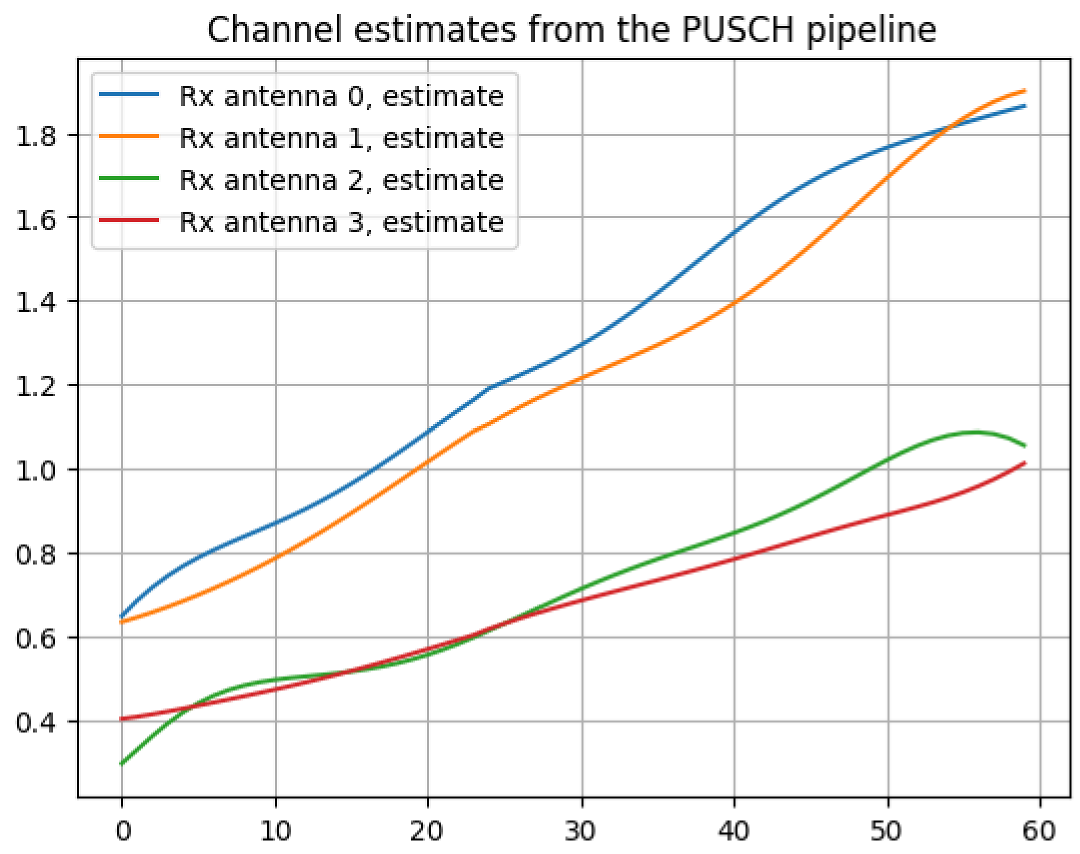
Example 3 - SFN.Slot 195.4 from time 2024-03-21 12:18:39.192000



Power in RU Antennas
Ant 0
Ant 1
Ant 2
Ant 3
Resource Element
Symbol
0
10
20
30
40
50
60



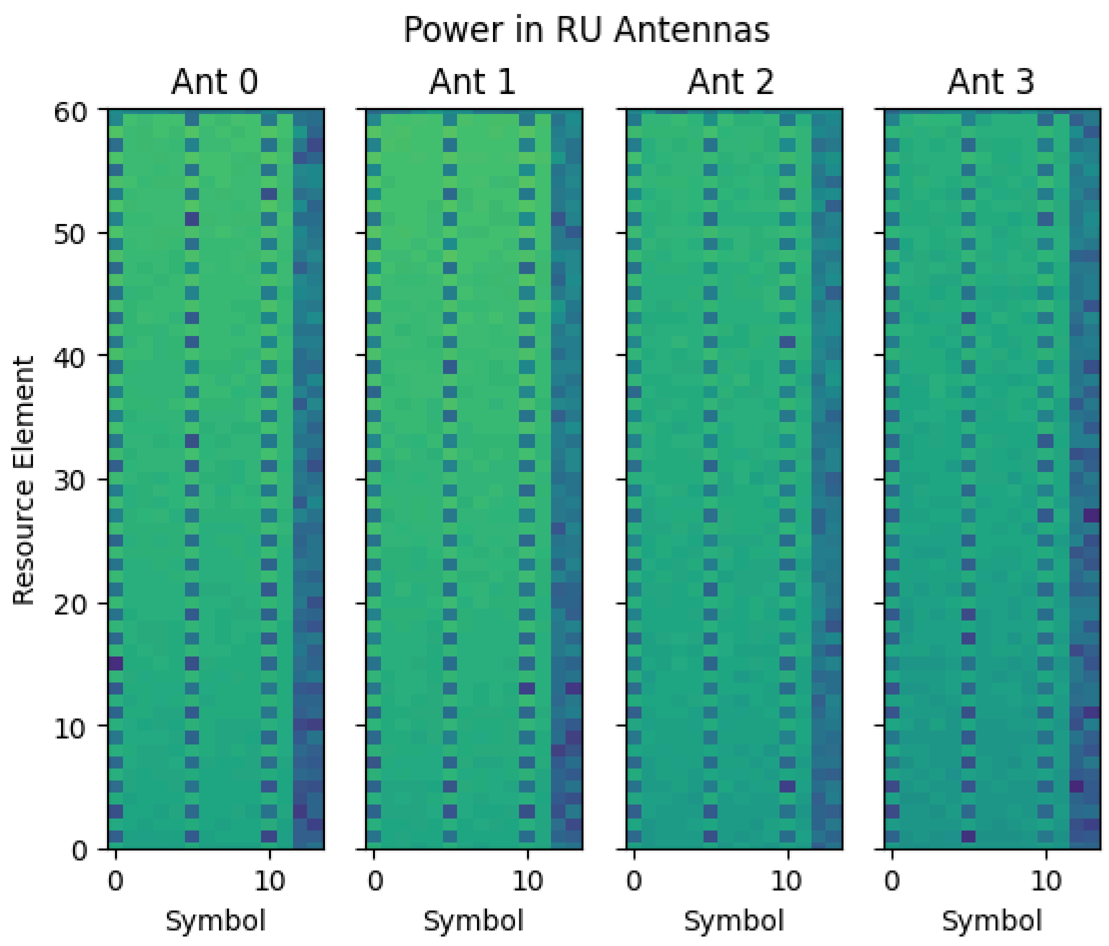
Pre-Equalized samples
Post-Equalized samples
40
20
0
−20
−40
0.5
0.0
−0.5

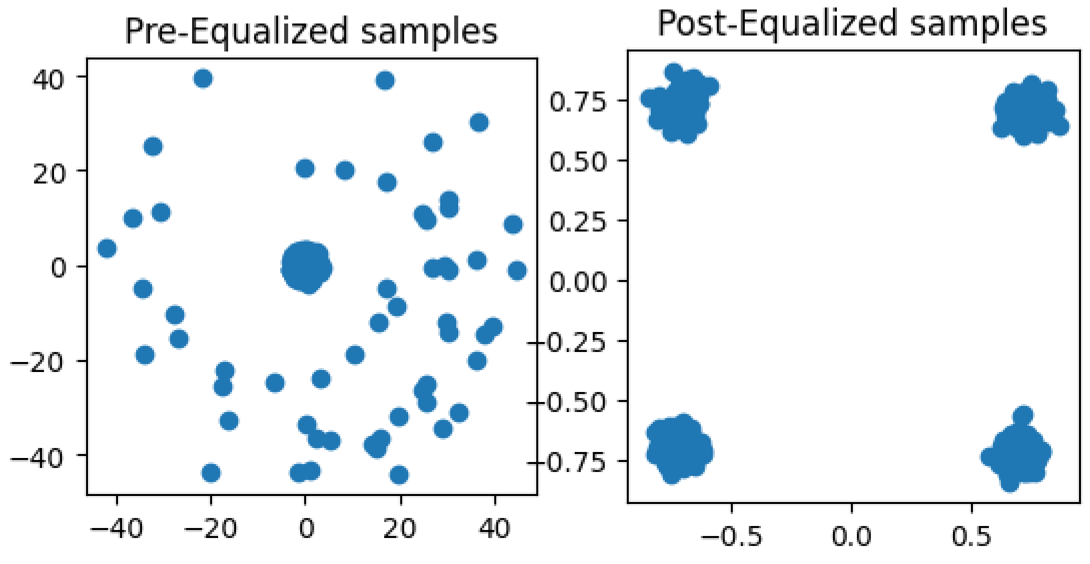


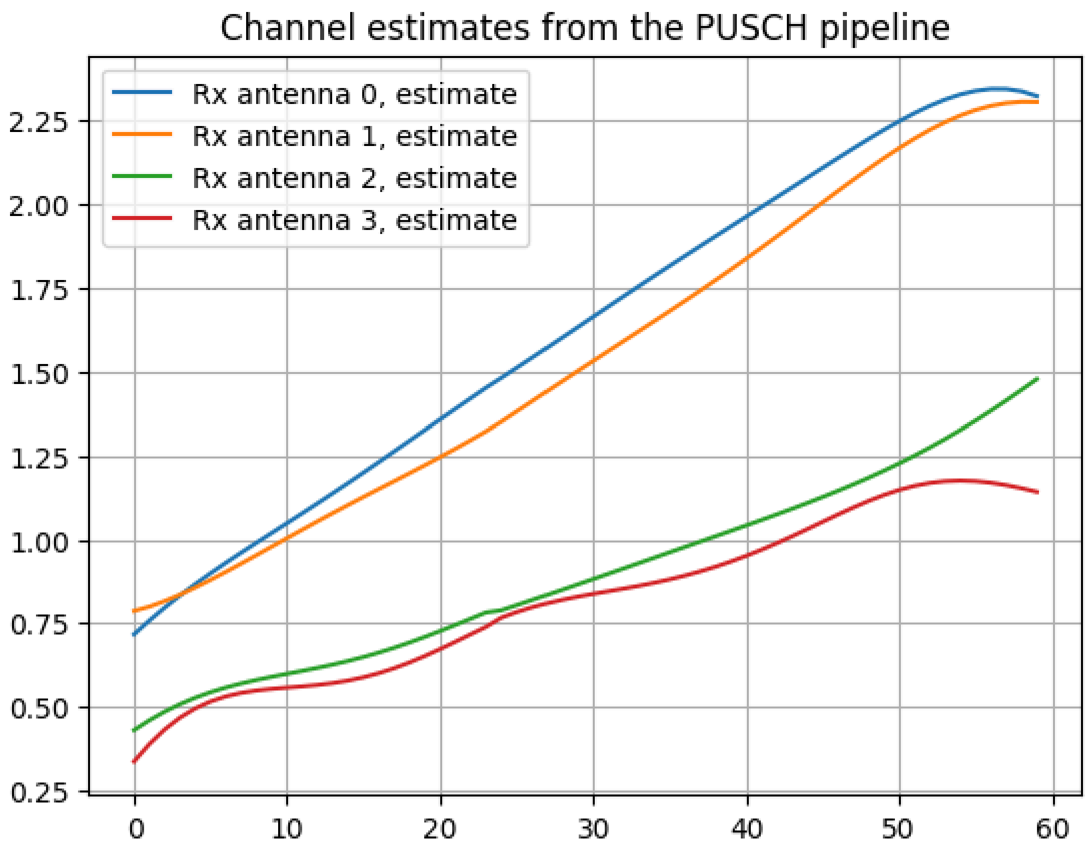
Separated kernels PUSCH decoding success for SFN.Slot 195.4 from time 2024-03-21 12:18:39.192000

Fused PUSCH decoding success for SFN.Slot 195.4 from time 2024-03-21 12:18:39.192000

Example 4 - SFN.Slot 201.4 from time 2024-03-21 12:18:39.252000


Power in RU Antennas
Ant 0
Ant 1
Ant 2
Ant 3
Resource Element
0
10
20
30
40
50
60
Symbol
0
10

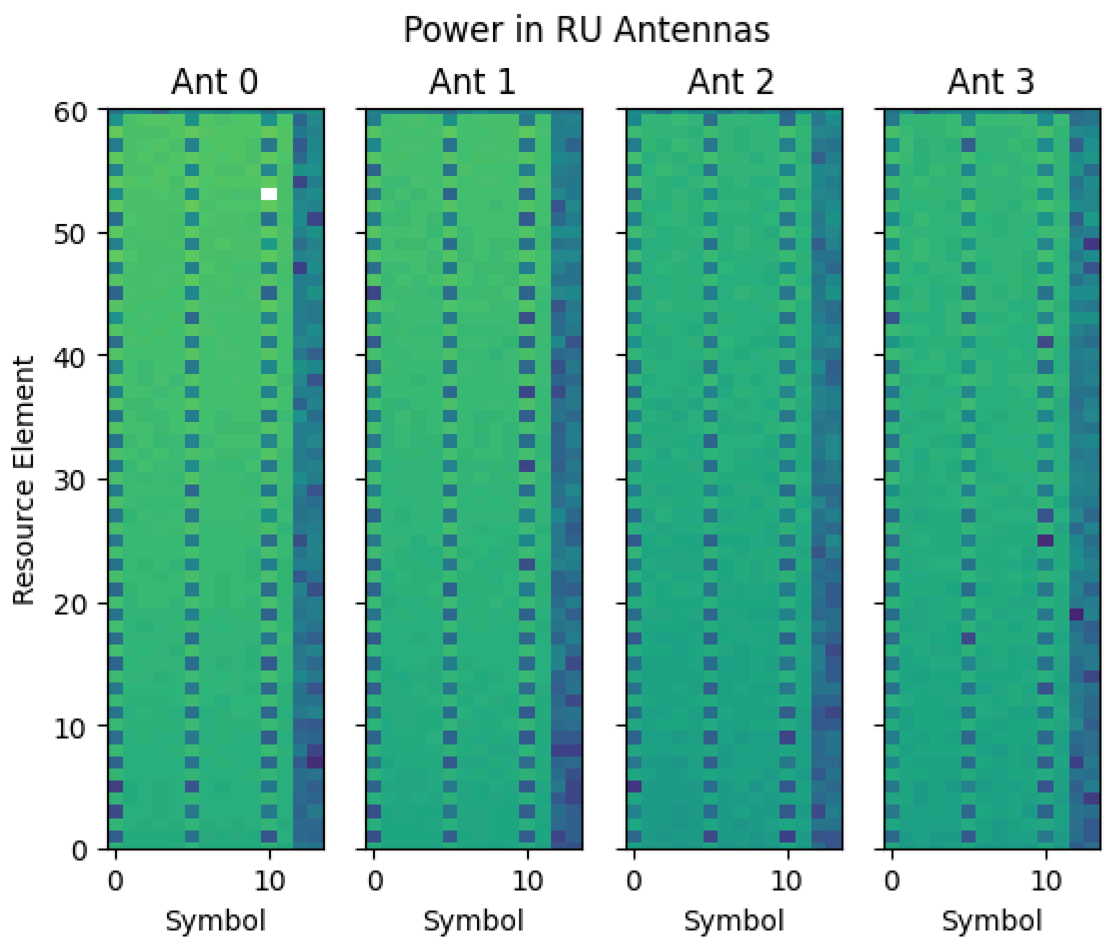

Pre-Equalized samples
Post-Equalized samples
40
20
0
−20
−40
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.5
0.0
0.5

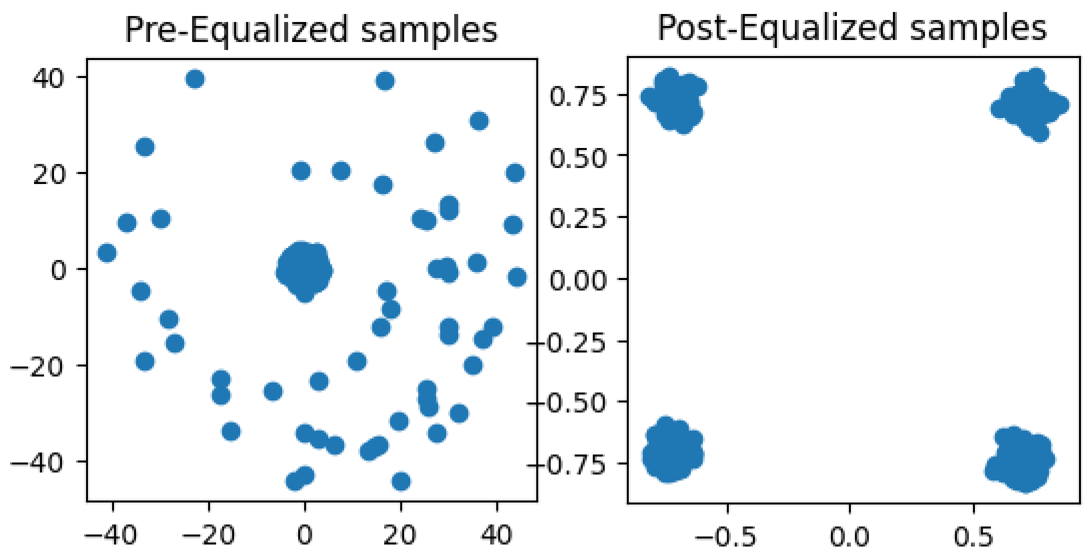


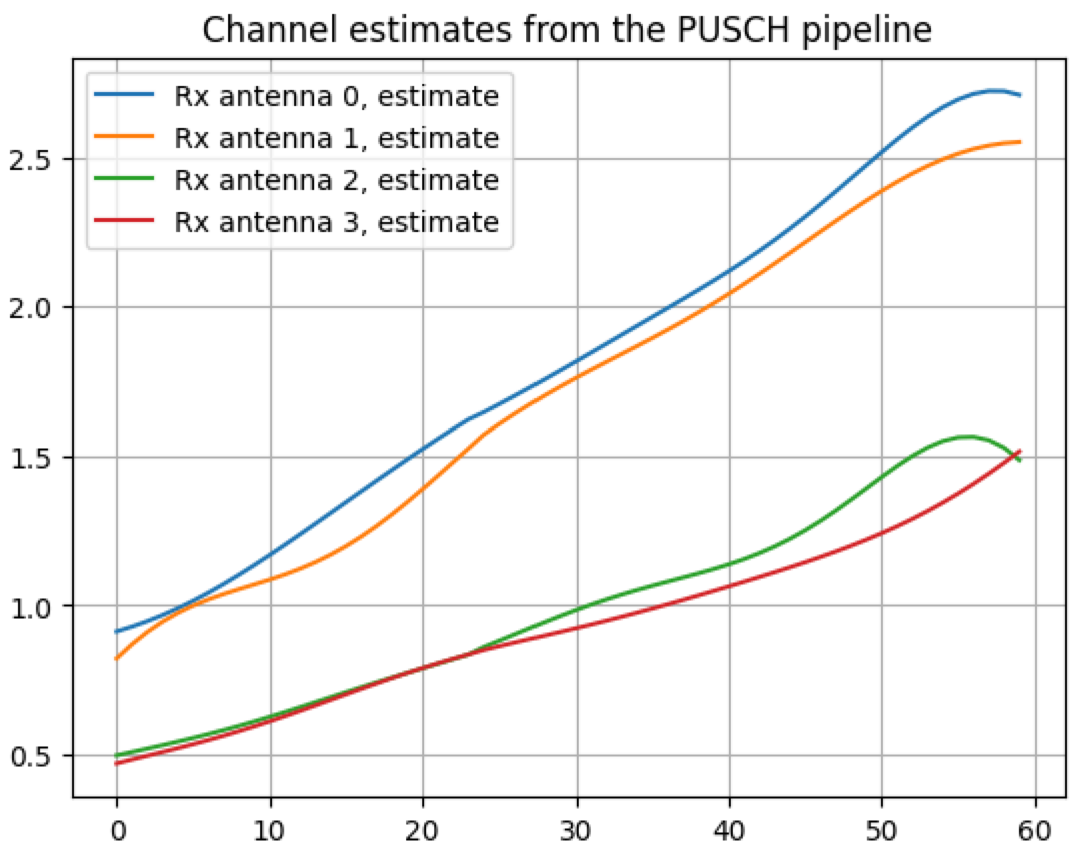
Separated kernels PUSCH decoding success for SFN.Slot 201.4 from time 2024-03-21 12:18:39.252000

Fused PUSCH decoding success for SFN.Slot 201.4 from time 2024-03-21 12:18:39.252000

Example 5 - SFN.Slot 209.4 from time 2024-03-21 12:18:39.332000


Power in RU Antennas
Ant 0
Ant 1
Ant 2
Ant 3
Resource Element
Symbol


Pre-Equalized samples
Post-Equalized samples



Separated kernels PUSCH decoding success for SFN.Slot 209.4 from time 2024-03-21 12:18:39.332000

Fused PUSCH decoding success for SFN.Slot 209.4 from time 2024-03-21 12:18:39.332000