ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

AI for Non-Terrestrial Networks

Bruno De Filippo, PhD student – bruno.defilippo2@unibo.it
Riccardo Campana, PhD student – riccardo.campana7@unibo.it

Digicomm Research group
Electrical, Electronic and Information Engineering Department (DEI)
University of Bologna

Introduction

- **Non-Terrestrial Networks** are gaining **broad interest**
  - Standardization, Industry, Researchers, End users
- **Artificial Intelligence** has reached an **all-time high popularity**
  - Advances in hardware, models, availability
  - Applications to countless fields

Perfect timeliness for AI-based NTNs!

- Basics for the development of Neural Networks to support NTNs
  - Theoretical
  - Practical (Python: Keras, TensorFlow)

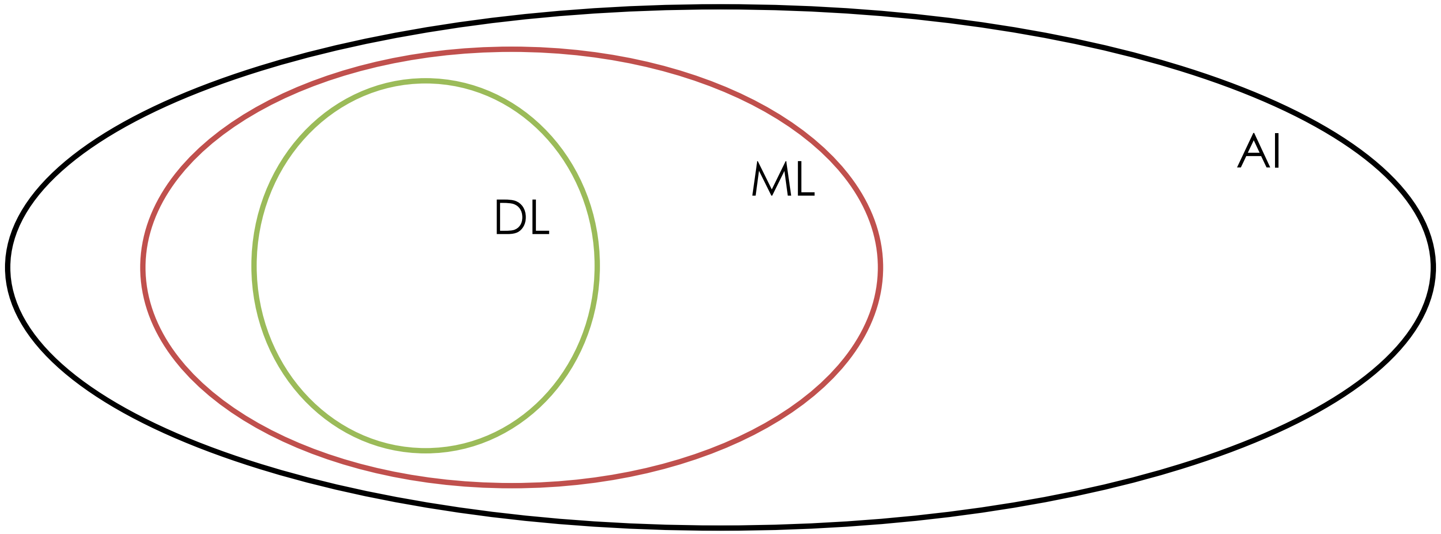
Outline

- A Primer on Artificial Intelligence
- AI in Telecommunications: From Literature to Standardization
- Non-Terrestrial Networks: Challenges and Impairments
- Use Case 1: AI-based Demodulator for Sparse Code Multiple Access
- Use Case 2: AI-based Channel Prediction
- The Way Forward for AI in NTN
- Q&A

A Primer on Artificial Intelligence

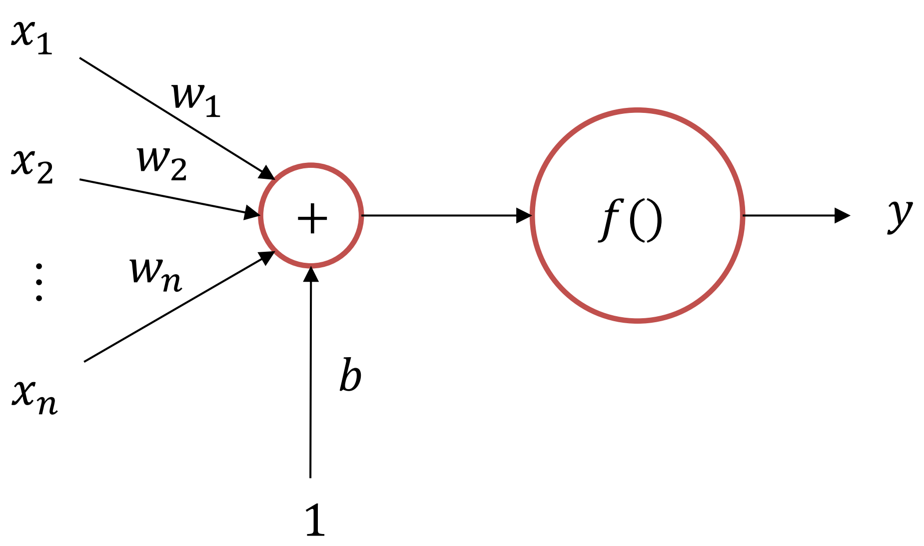
Artificial Intelligence: what it is, what it is not

- **Artificial Intelligence**: mimic human intelligence
  - General-purpose AI vs Specialized AI
- **Machine Learning**: develop AI by automatically learning from data
  - Decision Trees (classification), K-Means (clustering), …
- **Deep Learning**: usage of **Neural Networks** as ML algorithms



The Perceptron: the fundamental unit of Neural Networks

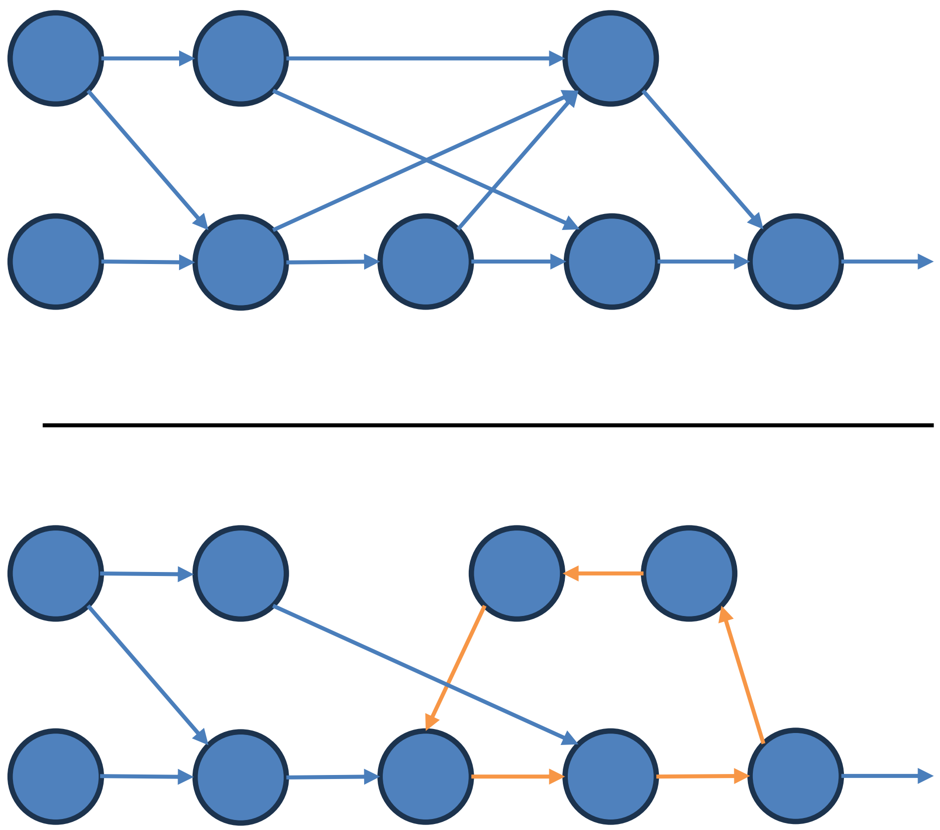
- Inspiration from behavior seen in **biological neurons**
- Three main operations:
  - **Weighted sum** of an input vector
  - Sum of weighted **bias**
  - **Activation function** (typically non-linear)
- The perceptron is a ML algorithm!
  - Classifier with sigmoid activation function
  - Linear regressor with linear activation function



$$\boxed{y = f(\boldsymbol{x} \cdot \boldsymbol{w} + b)}$$

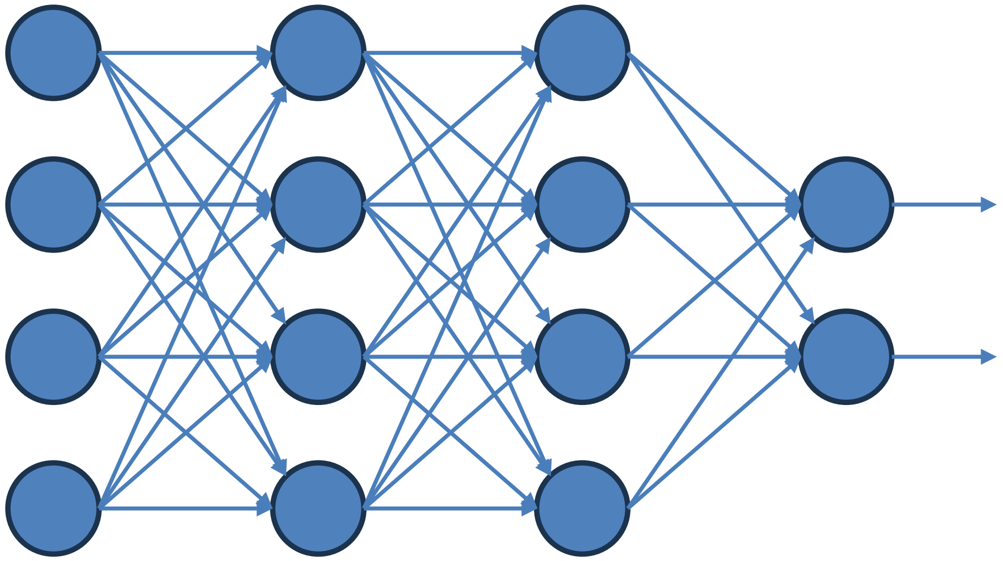
From the Perceptron to Neural Networks

- The outputs of multiple Perceptrons can be used as input to another Perceptron
- **Feedforward NNs** process the input data in **only one direction** – input to output
  - Fully-Connected NNs
  - Convolutional NNs
  - …
- **Recurrent NNs** allow **cycles and feedback loops** inside the network
  - Gated Recurrent Units
  - Long Short-Term Memory
  - …



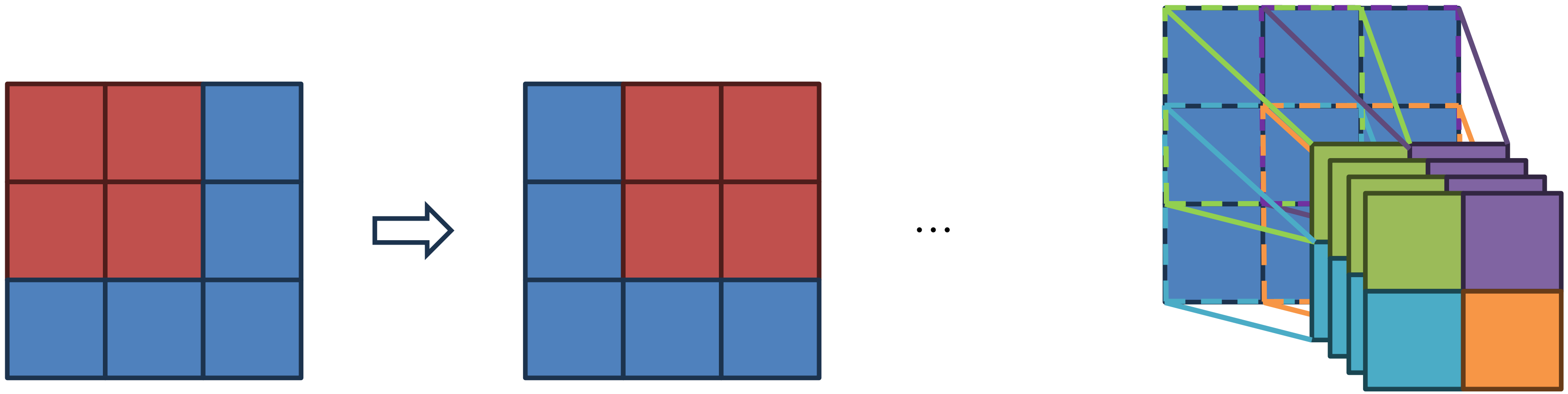
Fully-connected Neural Networks

- Multiple Perceptrons are deployed in **dense** layers
- Each Perceptron in layer *i* takes as input the output of **every** Perceptron in layer *i-1*
- If properly designed and trained, a FC NN can **approximate the data distribution**



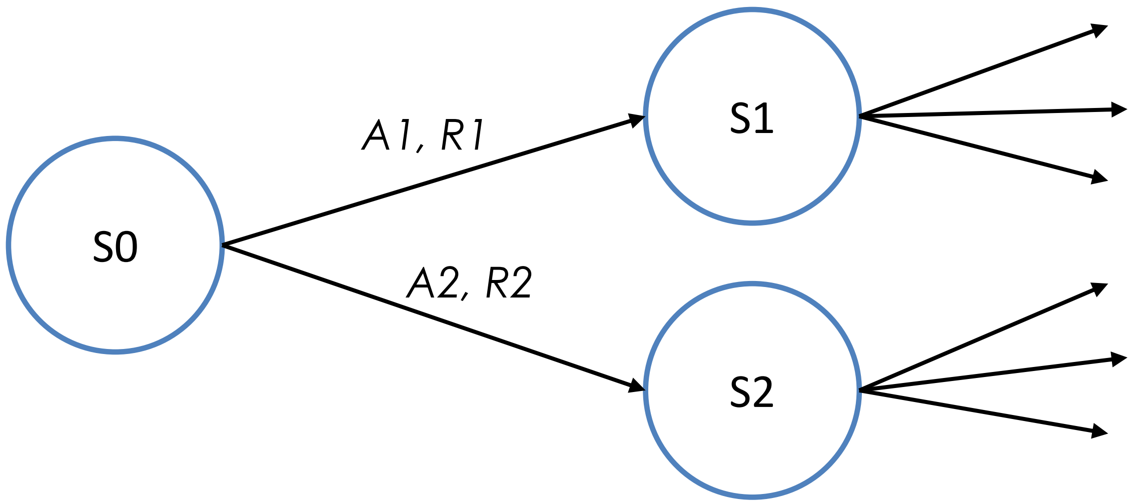
Convolutional Neural Networks

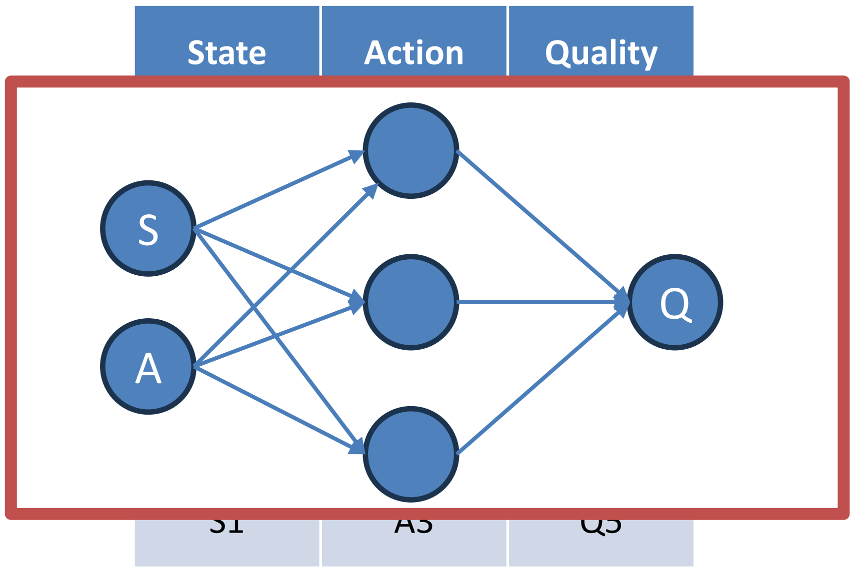
- Use of **convolutional** layers to extract spatial features
- Each Perceptron in layer *i* takes as input the output of **a specific subset** of Perceptrons in layer *i-1*
- Each **filter** act as a perceptron that process the entire input space by sliding over it



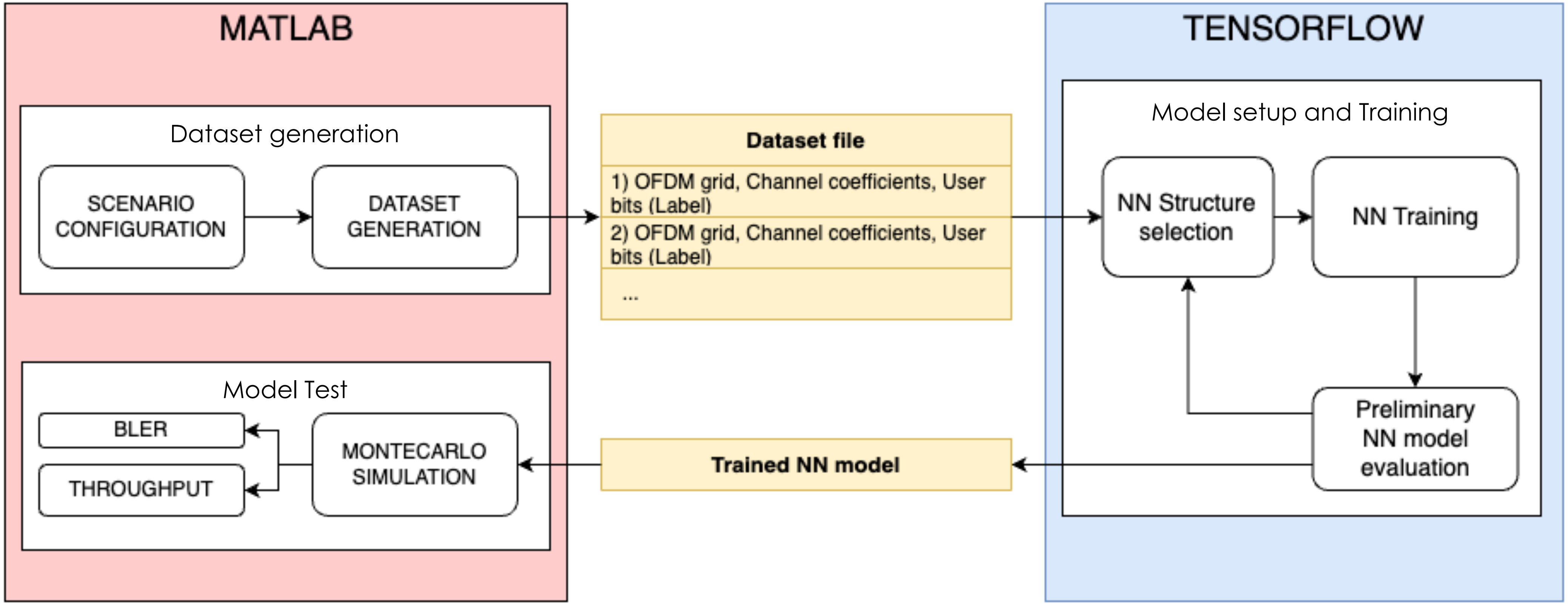
Deep Reinforcement Learning

- **RL**: Learn how to behave in an environment
  - Analyze the current state
  - Take an action (move to new state)
  - Earn a reward
- **Deep** RL: choose the action with a NN





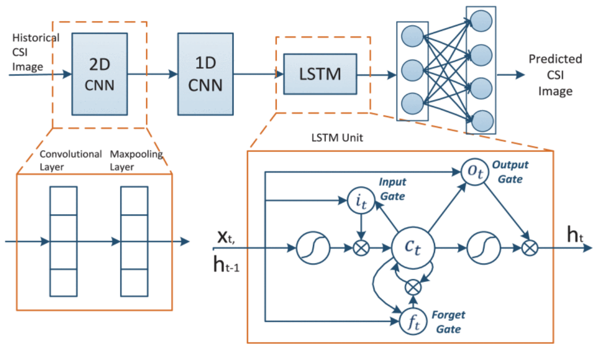
Deep Learning: Typical Workflow



AI in Telecommunications:
From Literature to Standardization

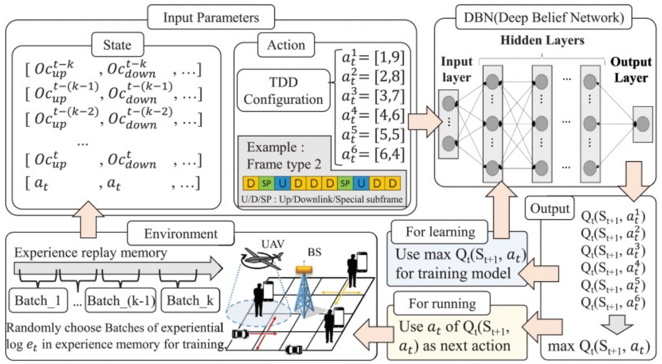
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

AI in the Telecommunications Literature: examples



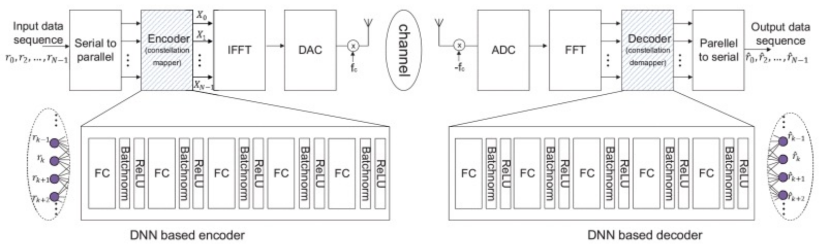
C. Luo, J. Ji, Q. Wang, X. Chen and P. Li, "Channel State Information Prediction for 5G Wireless Communications: A Deep Learning Approach," in *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 227-236, 1 Jan.-March 2020

Channel prediction



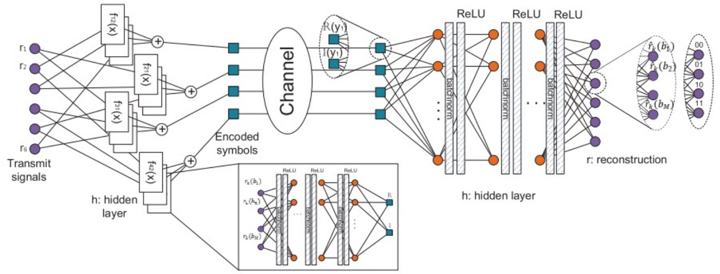
F. Tang, Y. Zhou and N. Kato, "Deep Reinforcement Learning for Dynamic Uplink/Downlink Resource Allocation in High Mobility 5G HetNet," in *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 12, pp. 2773-2782, Dec. 2020

Resource allocation



M. Kim, W. Lee and D. -H. Cho, "A Novel PAPR Reduction Scheme for OFDM System Based on Deep Learning," in *IEEE Communications Letters*, vol. 22, no. 3, pp. 510-513, March 2018

PAPR reduction in OFDM



M. Kim, N. -I. Kim, W. Lee and D. -H. Cho, "Deep Learning-Aided SCMA," in *IEEE Communications Letters*, vol. 22, no. 4, pp. 720-723, April 2018

Generation of optimized modulations

AI in 3GPP

- Study on AI-based **NG-RAN** in **Rel-17** (TR 37.817)
  - New study in **Rel-18** on AI for **Network Energy Savings**, **Load Balancing**, **Mobility Optimizations**

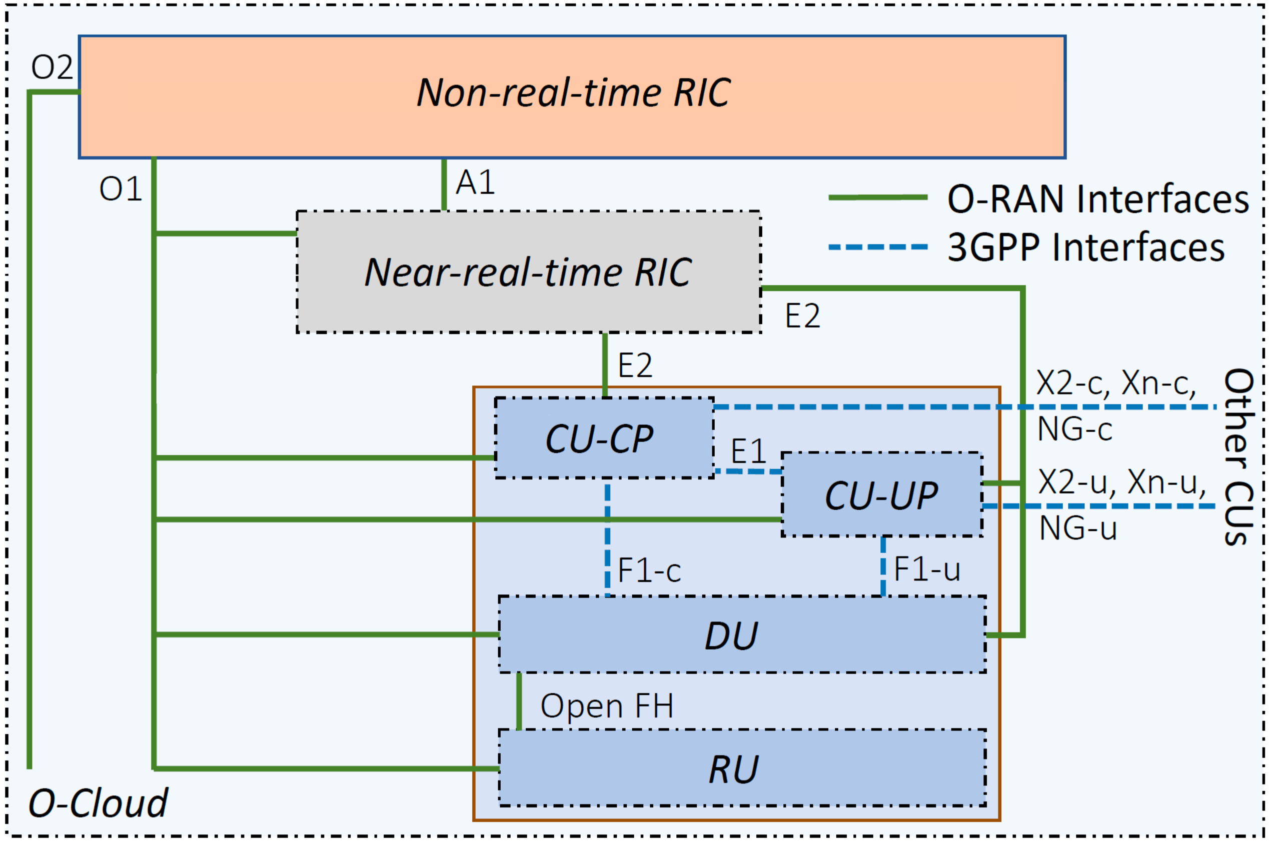
- Study on AI-based **NR Air Interface** in **Rel-18** (TR 38.843, draft)
  - **Channel State Information**
    - Frequency-domain **compression**
    - Time-domain **prediction**
    - Possibly with interplay between UE and gNB (*e.g.*, with AutoEncoders)
  - **Beam Management**
    - Spatial and temporal **prediction**
  - **Positioning**
    - **AI-based** (*e.g.*, fingerprinting) or **AI-aided** (*e.g.*, measurement enhancement)

AI in Open RAN (1/2)

It is based on the concepts of:

- Disaggregation
- Virtualization
- Open Interfaces
- RAN Intelligent Controllers (RIC)

Artificial intelligence enablers

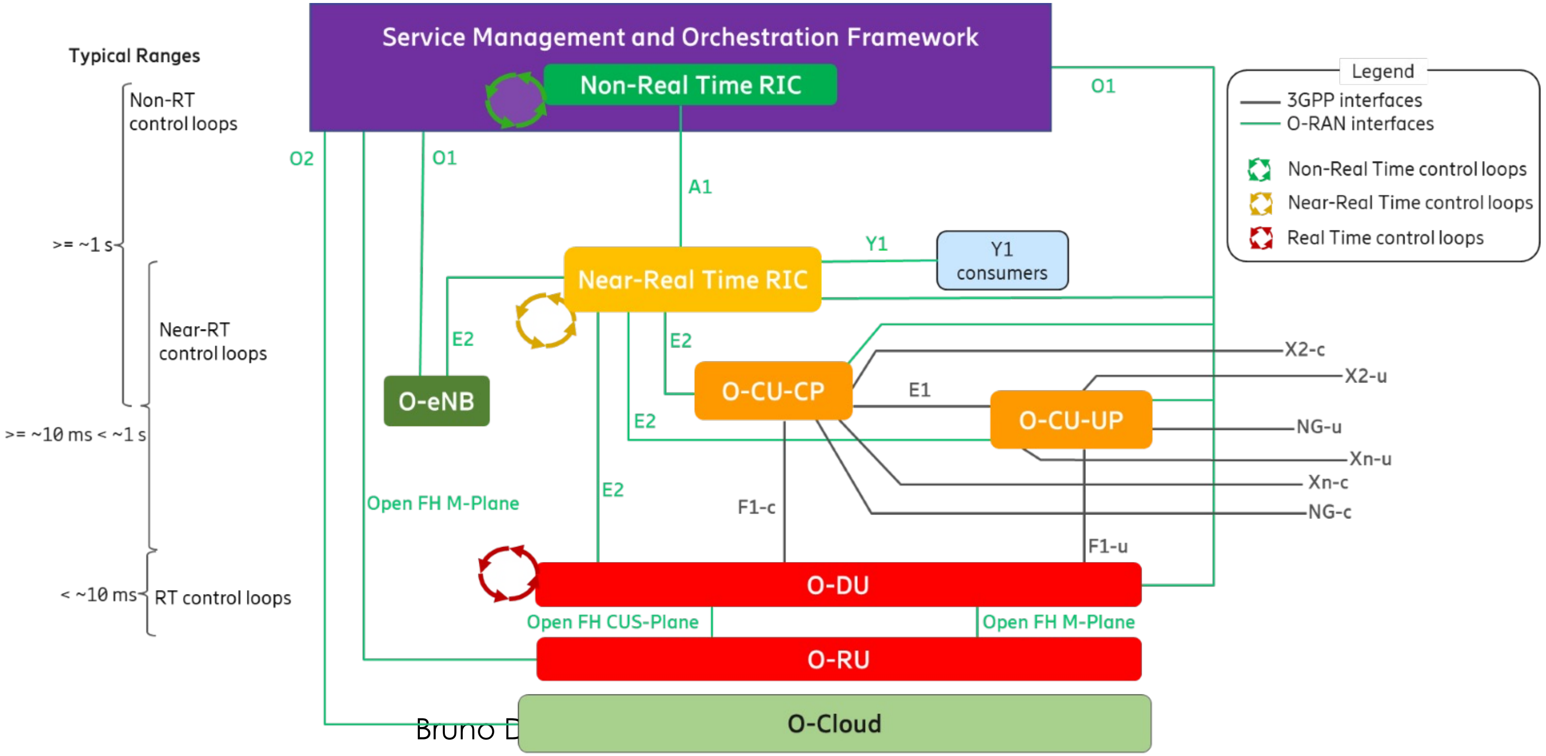


M. Polese et al., "Understanding O-RAN: Architecture, Interfaces, Algorithms, Security, and Research Challenges," Aug. 2022, arXiv:2202.01032

AI in Open RAN (2/2)

The O-RAN based NTN architecture enables:

- The **collection of KPIs from the network** nodes (E2 and O1) through the open interfaces;
- The exploitation of the collected data to **train the AI/ML models in the RICs**;
- The exploitation of the input KPI data and trained AI/ML models to **optimize the RAN configuration parameters**.



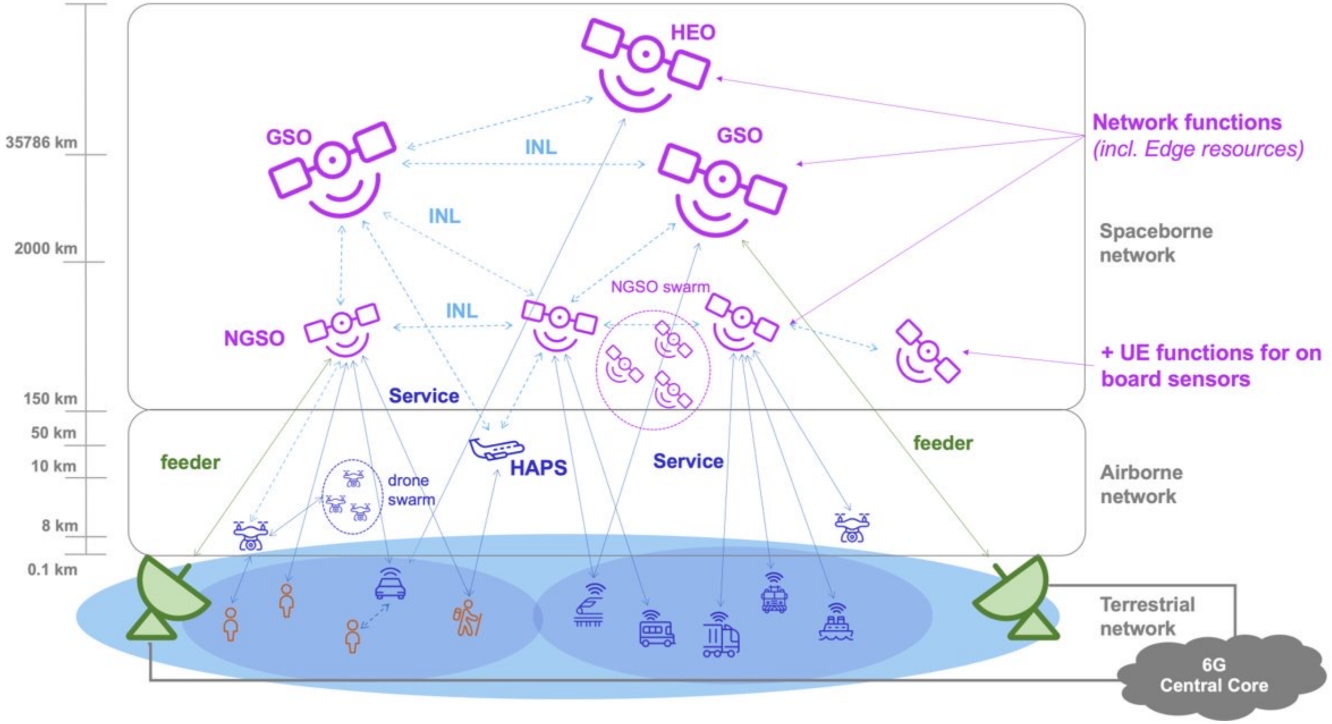
Bruno D

O-RAN architecture description 5.00, O-RAN, Alfter, Germany, Jul. 2021.

Non-Terrestrial Networks: Challenges and Impairments

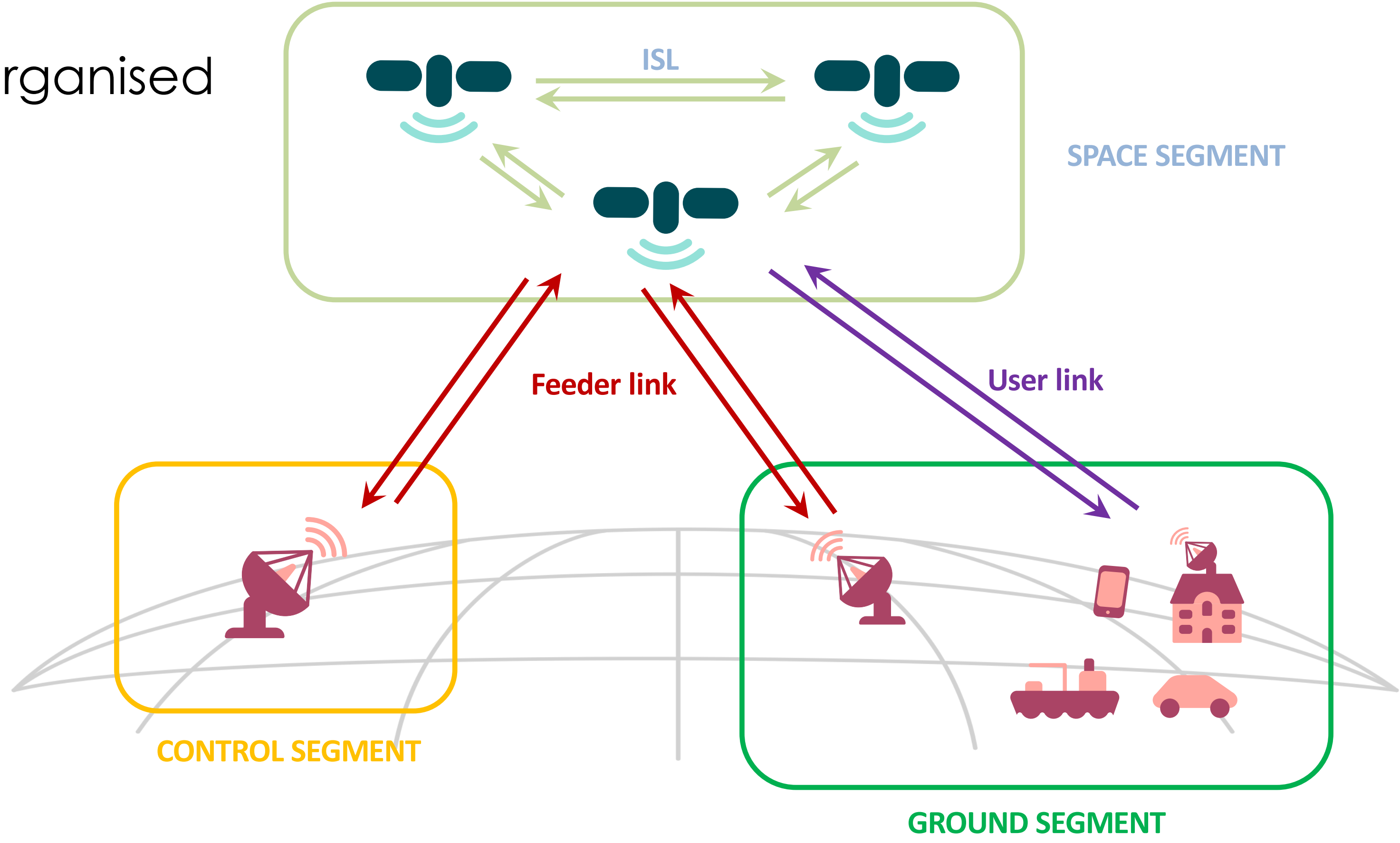
NTN architecture

- Non-terrestrial segment
  - A communication system encompassing flying communication elements
- The flying communication elements can be
  - Air-borne platforms
  - Space-borne platforms



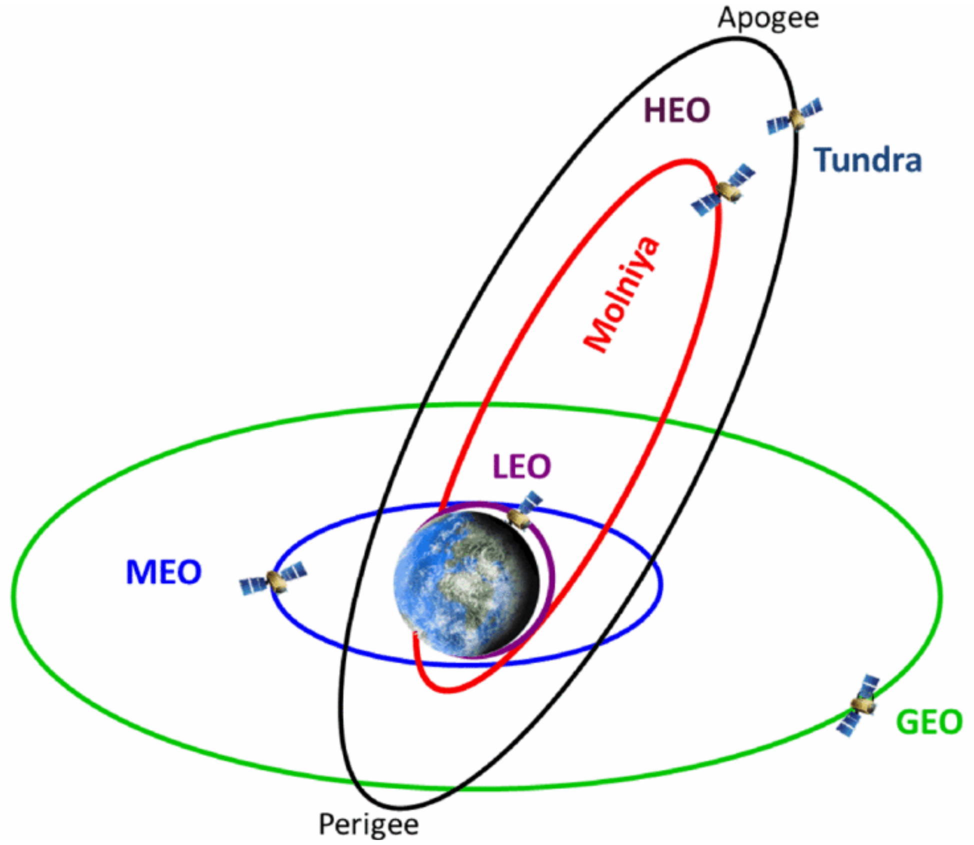
Satellite communications systems

- Space segment
  - 1+ communication satellites organised in a constellation
- Control segment
  - Network Control Center
  - Satellite Control Center
- Ground segment
  - Gateways
  - User Terminals



Satellite orbits

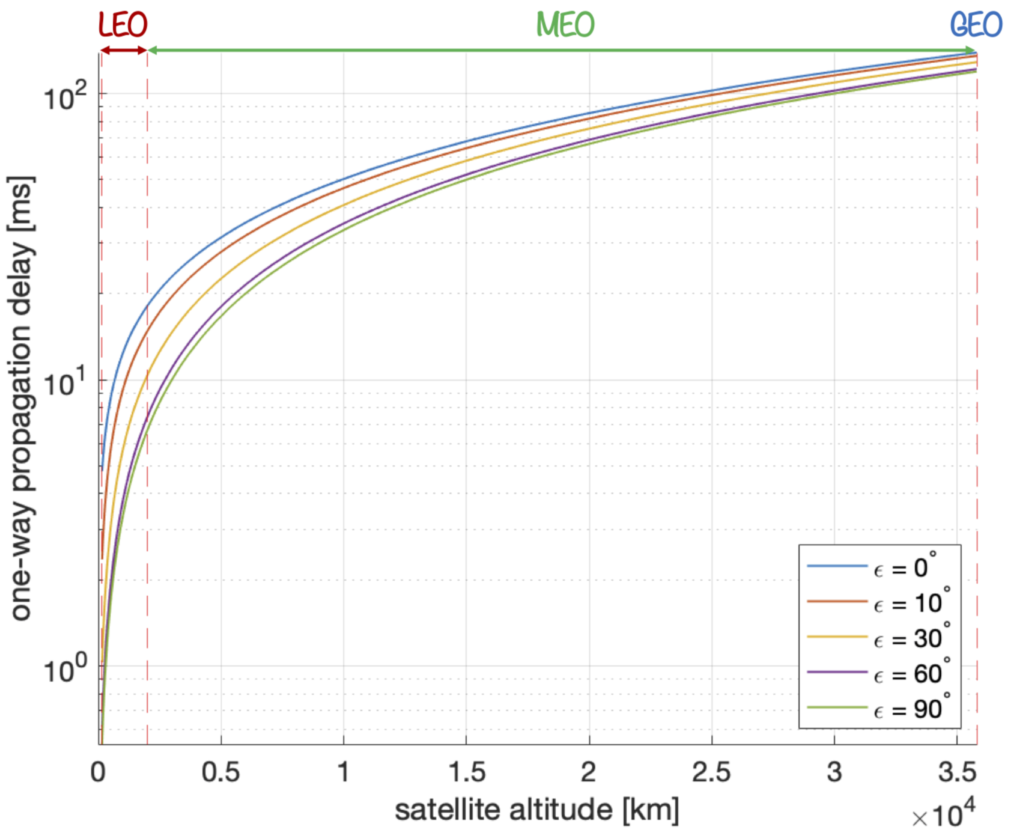
- **Geo-Synchronous Orbit (GSO)**
  - Period equal to one sidereal day
  - Geostationary Earth Orbit (GEO): GSO on the equatorial plane
    - The satellite appears as a fixed point in the sky
    - altitude ~36000 km
- **Non-GSO (NGSO)**
  - Medium Earth Orbit (MEO)
    - Typically around 20000 km
  - Low Earth Orbit (LEO)
    - 600-1200 km
  - vLEO
    - <500 km



Source: S. Plass et al., "Current Situation and Future Innovations in Arctic Communications," IEEE VTC Fall 2015, Sep. 2015

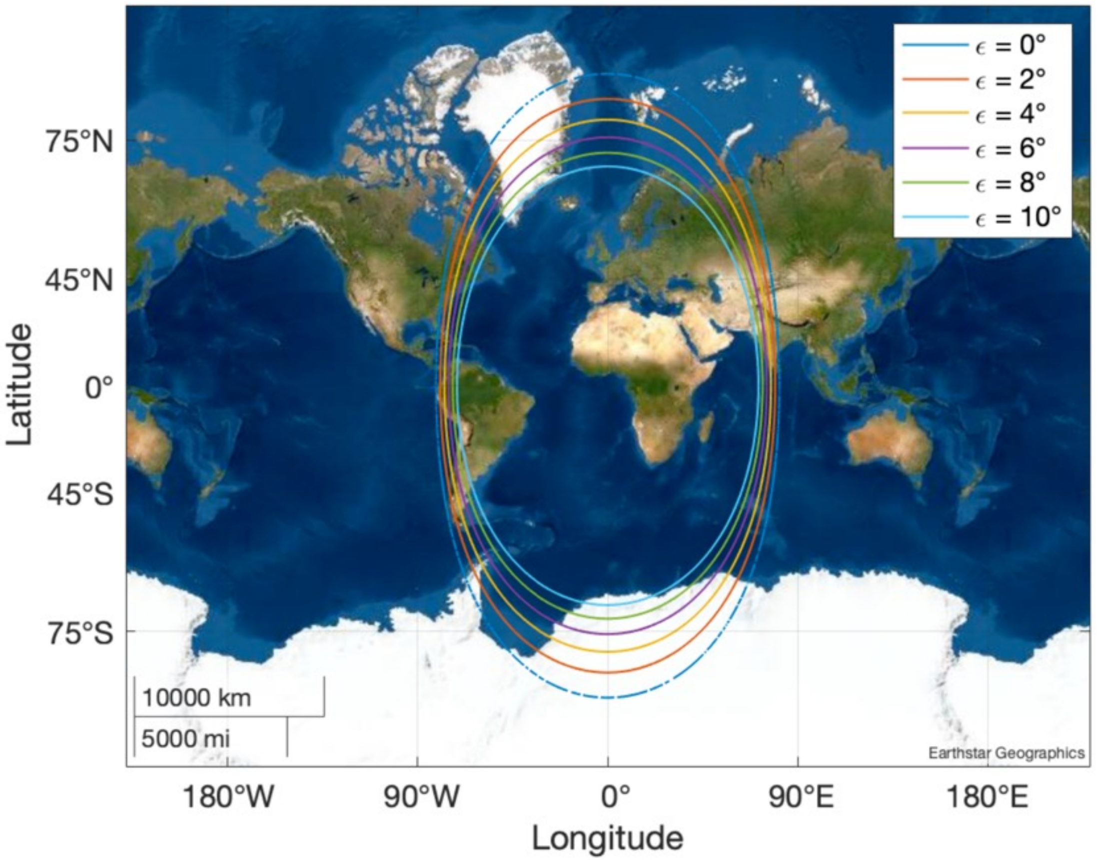
Satellite orbits impact: Latency

Latency sensibly increases when selecting higher altitude orbits

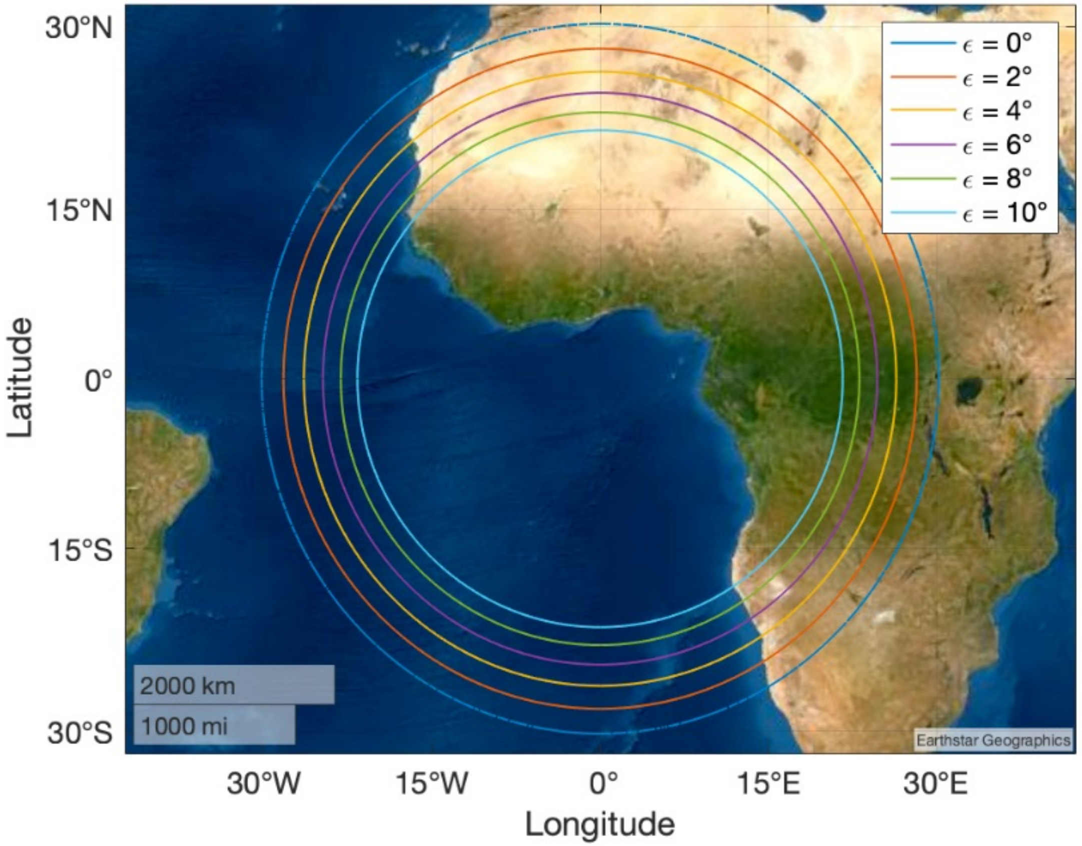


Bruno De Filippo – Riccardo Campana – WiSEE ‘23

Satellite orbits impact: Field of view



GEO

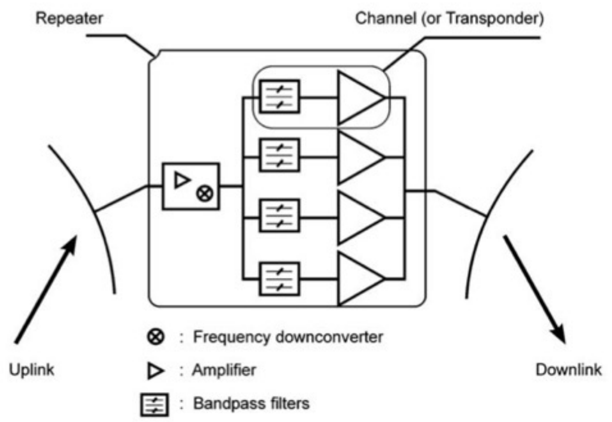


LEO

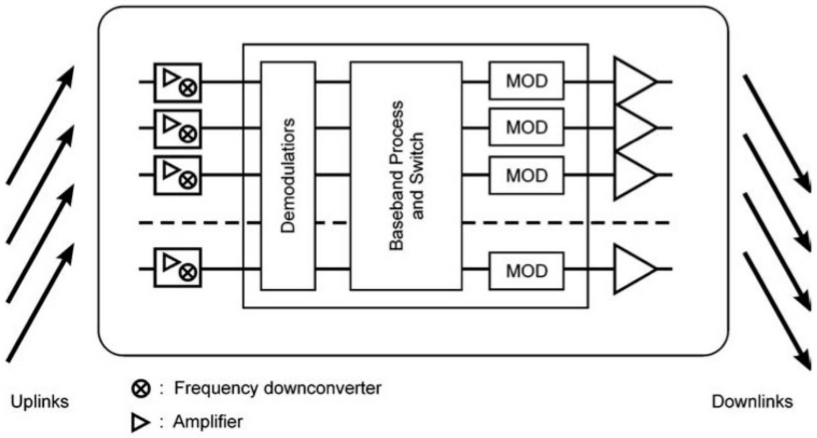
Main satellite components

A communication satellite consists of

- **Platform:** the subsystem permitting the satellite to operate
- **Payload:** antennas and Tx/Rx equipment
  - **Transparent** Tx/Rx: frequency conversion and amplification
  - **Regenerative** Tx/Rx: demodulation and modulation, protocol termination



Transparent



Regenerative

Source: G. Maral, M. Bousquet, "Satellite Communication Systems," 5th ed., Wiley, 2009

3GPP NTN Scenarios identified in TR 38.821

The targeted macro-scenarios are

- GEO with transparent payload (A)
- LEO with transparent payload and fixed/moving beams (C1/C2)
- LEO with regenerative payload and fixed/moving beams (D1/D2)

All of the above scenarios can be implemented by means of

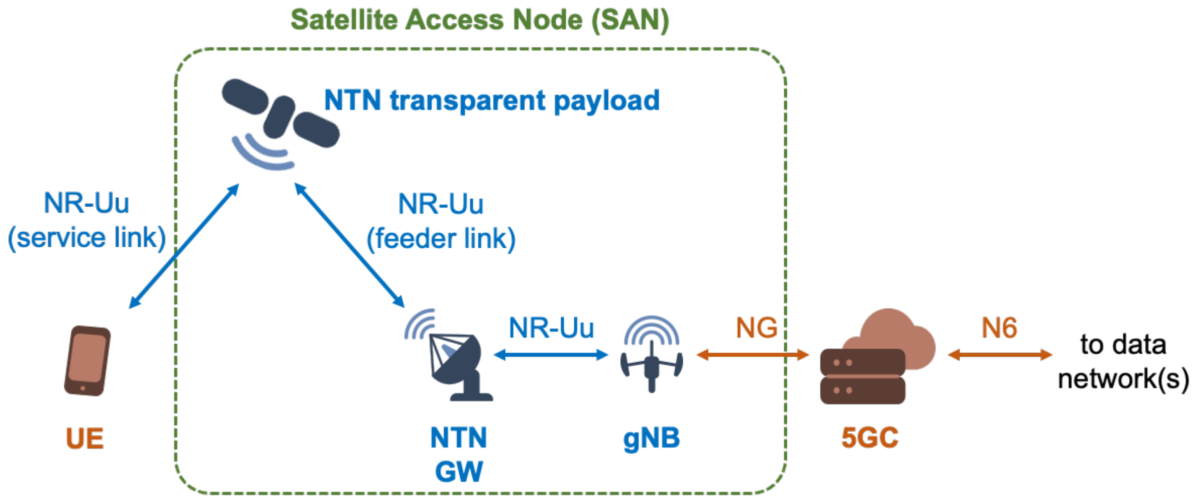
- Direct access (with/without functional split for regenerative payloads)
- Relay Nodes (RNs) or Integrated Access Backhaul (IAB) Nodes

	Transparent satellite	Regenerative satellite
GEO based non-terrestrial access network	Scenario A	Scenario B
LEO based non-terrestrial access network: steerable beams	Scenario C1	Scenario D1
LEO based non-terrestrial access network: the beams move with the satellite	Scenario C2	Scenario D2

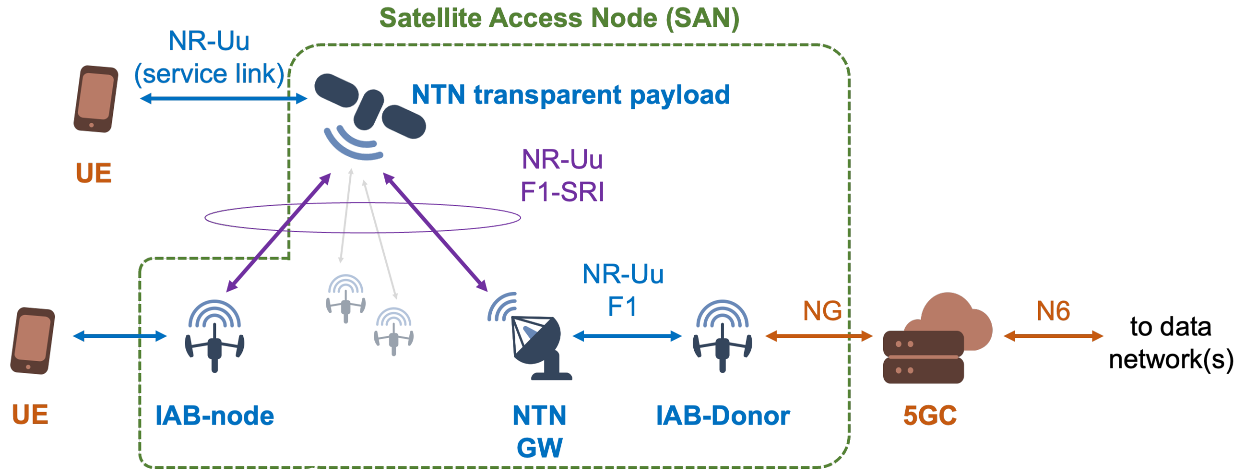
w/o ISL

w/ ISL

Transparent payload reference architecture



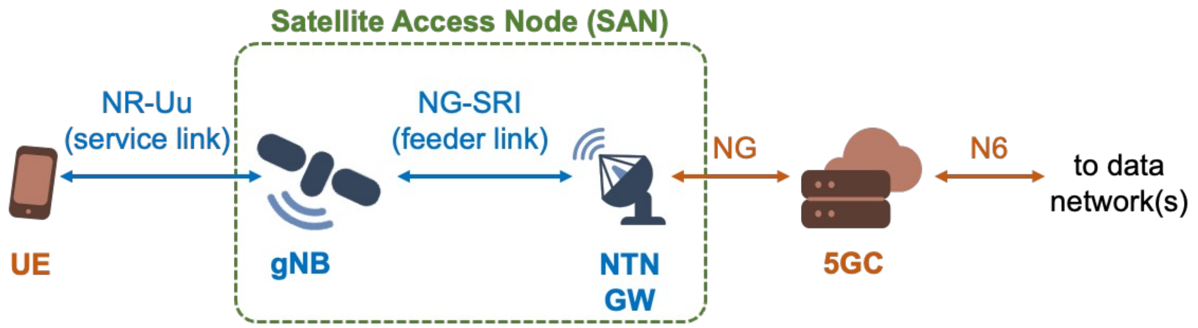
Direct Access



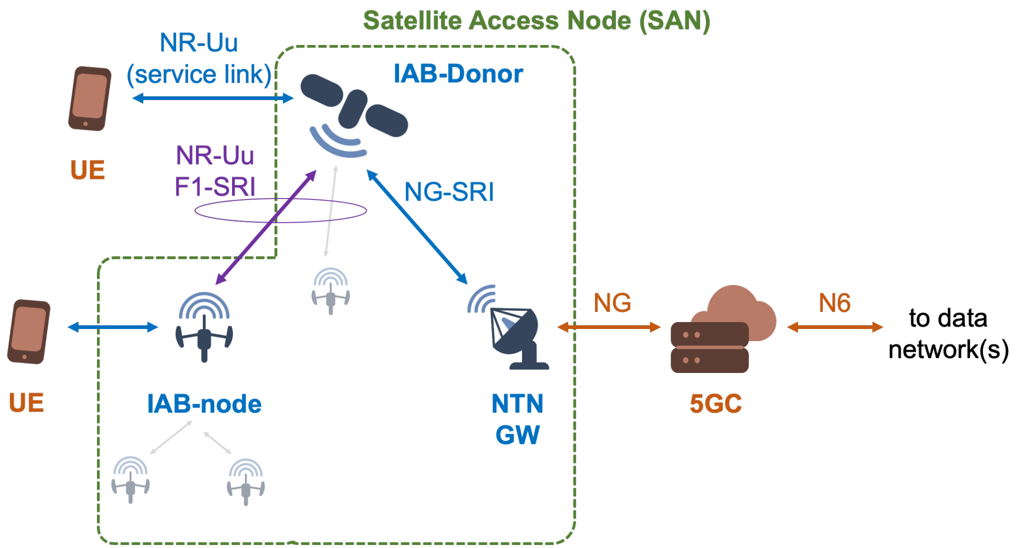
Indirect IAB Access

Source: EC HORIZON-JU-SNS-2022 Project 5G-STARDUST, D3.1 "System Requirements Analysis and Specifications," July 2023.

Regenerative payload reference architecture (no functional split)



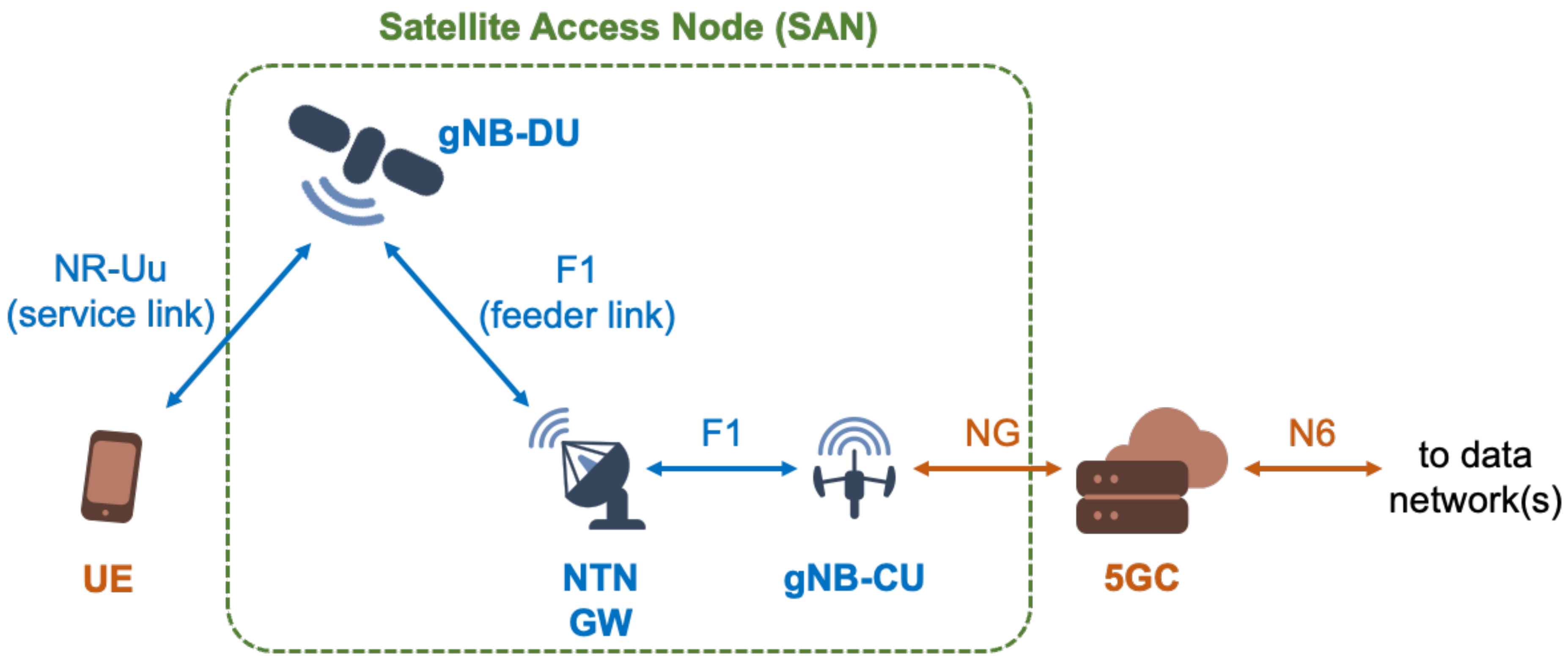
Direct Access



Indirect IAB Access

Source: EC HORIZON-JU-SNS-2022 Project 5G-STARDUST, D3.1 "System Requirements Analysis and Specifications," July 2023.

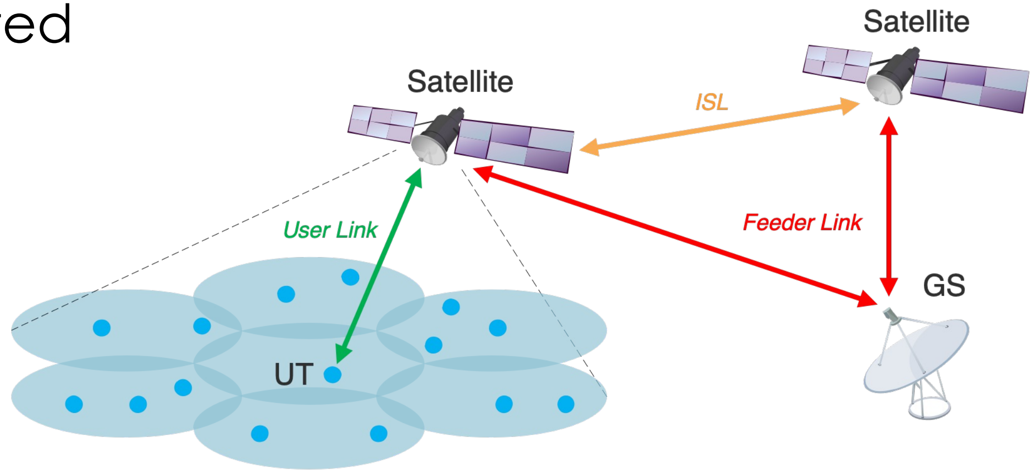
Regenerative payload reference architecture (with functional split)



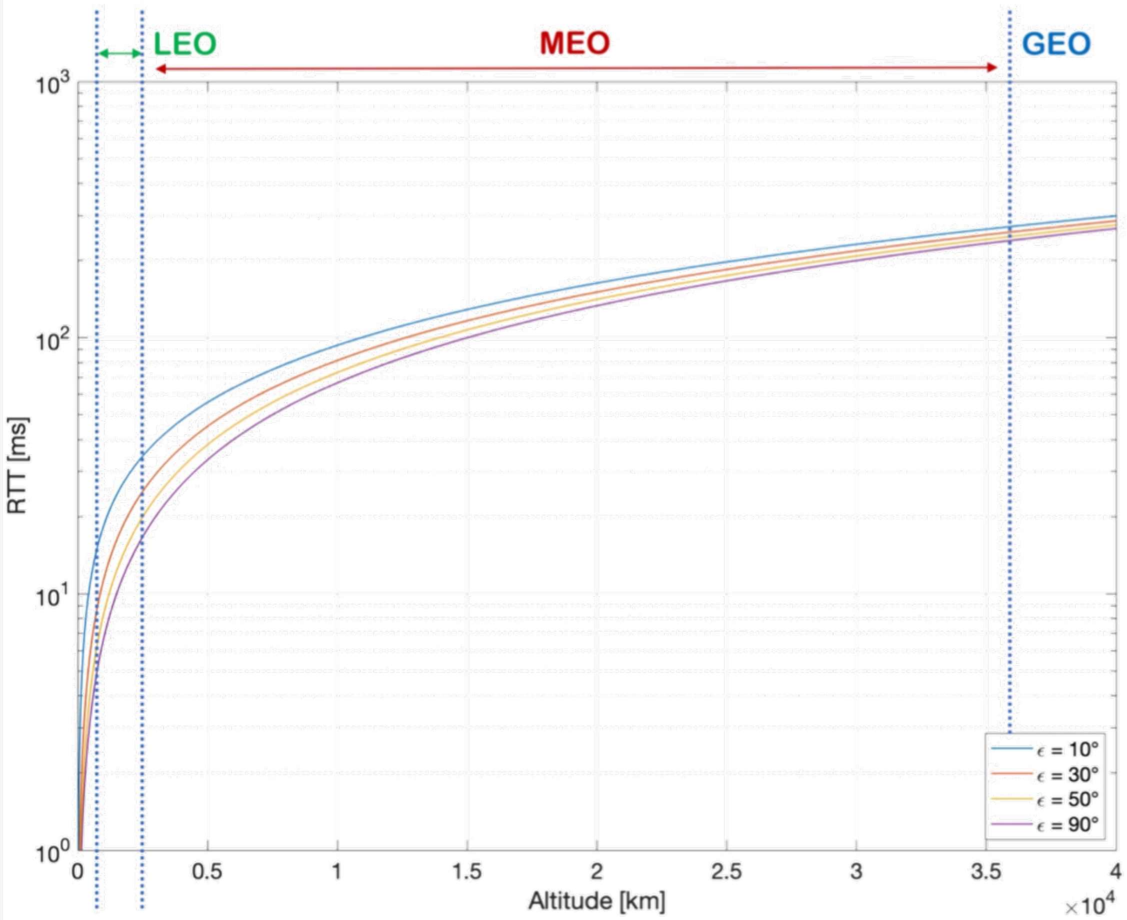
Source: EC HORIZON-JU-SNS-2022 Project 5G-STARDUST, D3.1 "System Requirements Analysis and Specifications," July 2023.

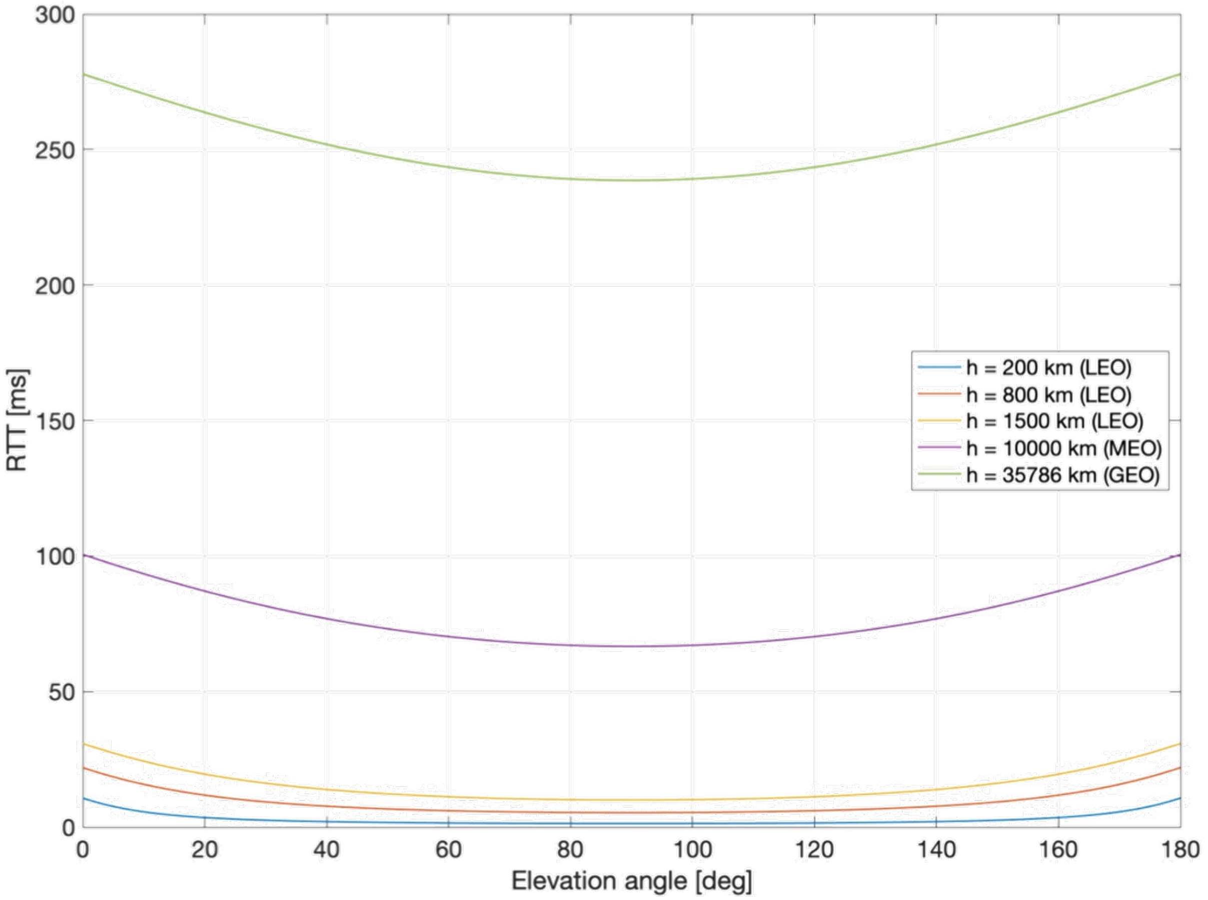
Typical Impairments in NTN: Delay

- Different types of delay are involved in SatCom:
  - the propagation delay along the user link
  - the propagation delay along the feeder link
  - the propagation delay along the ISL (if present)
- The propagation delay, directly related to the slant range, is the predominant one and its value is much larger than those of terrestrial networks.
- **Larger the footprint | Higher the orbit | Smaller the elevation angle ⇒ Larger the RTT**
- This could result in bottlenecks with harmful impacts on the protocols and procedures of the air interface implemented



Typical Impairments in NTN: Delay





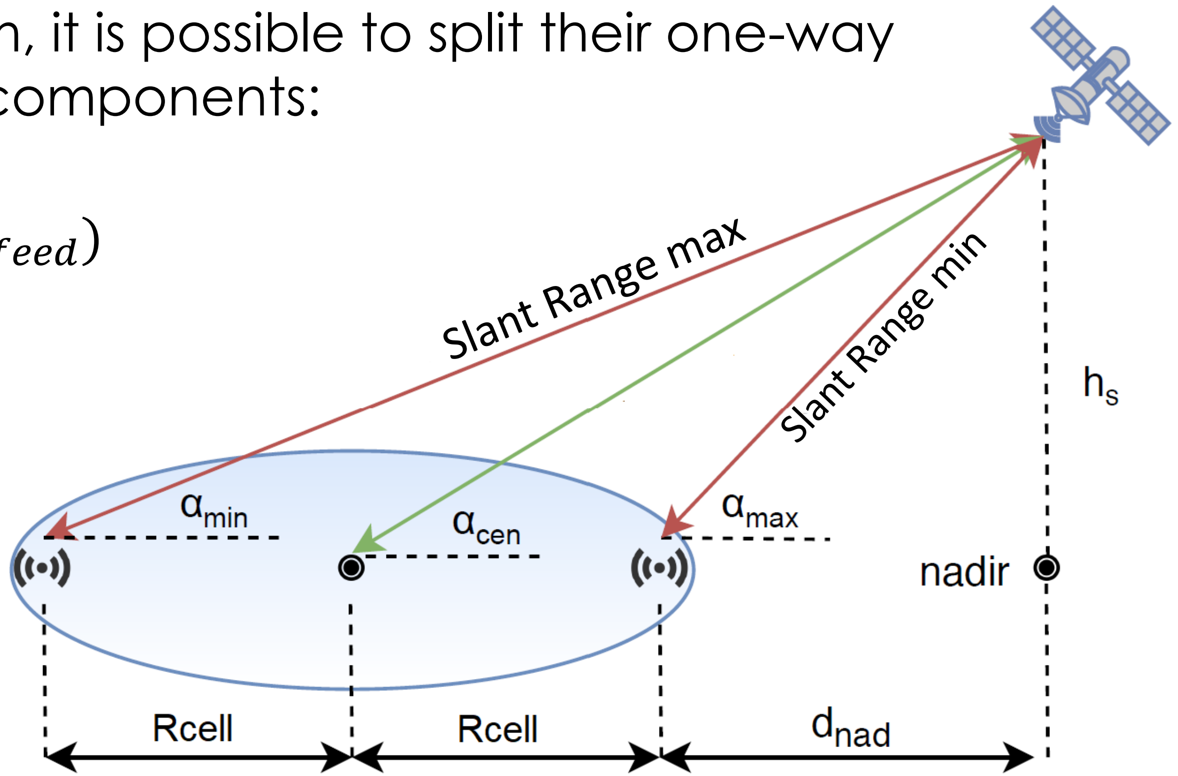
- Round Trip Time vs. Altitude
- Round Trip Time vs. Elevation

Typical Impairments in NTN: Differential Delay

- **Differential delay** is the difference among the propagation delay experienced by two different UTs in the access area of the same satellite.

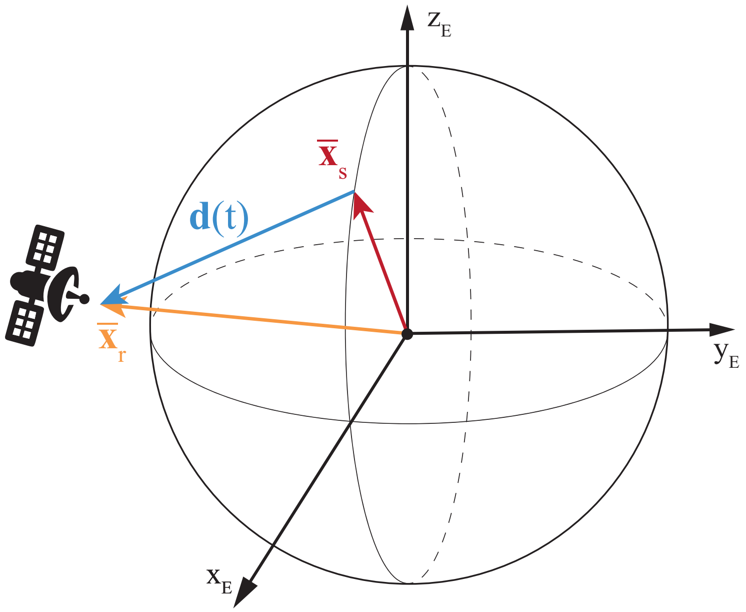
- For two or more UTs in the same beam, it is possible to split their one-way propagation delay, into two distinct components:

$$T_{OW} = \Delta\tau + T_{com} = \Delta\tau + (T_{user} + T_{isl} + T_{feed})$$



Typical Impairments in NTN: Doppler shift

- The **Doppler shift** consists in the change in the carrier frequency due to the relative motion between the satellite and the user terminal.
- When UTs mobility and LEO and VLEO satellite systems are considered, the Doppler shift can introduce significant frequency shifts with respect to those expected in terrestrial systems.



$$f_D(t) = -\left(\frac{\mathbf{d}(t)}{|\mathbf{d}(t)|} \cdot \frac{\partial \mathbf{x}_r}{\partial t} - \frac{\mathbf{d}(t)}{|\mathbf{d}(t)|} \cdot \frac{\partial \mathbf{x}_s}{\partial t}\right) \frac{f_0}{c}$$

$$= \frac{-(v_r(t) - v_s(t))}{c} f_0 = \frac{\Delta v}{c} f_0$$

approximated form as a function of the elevation angle

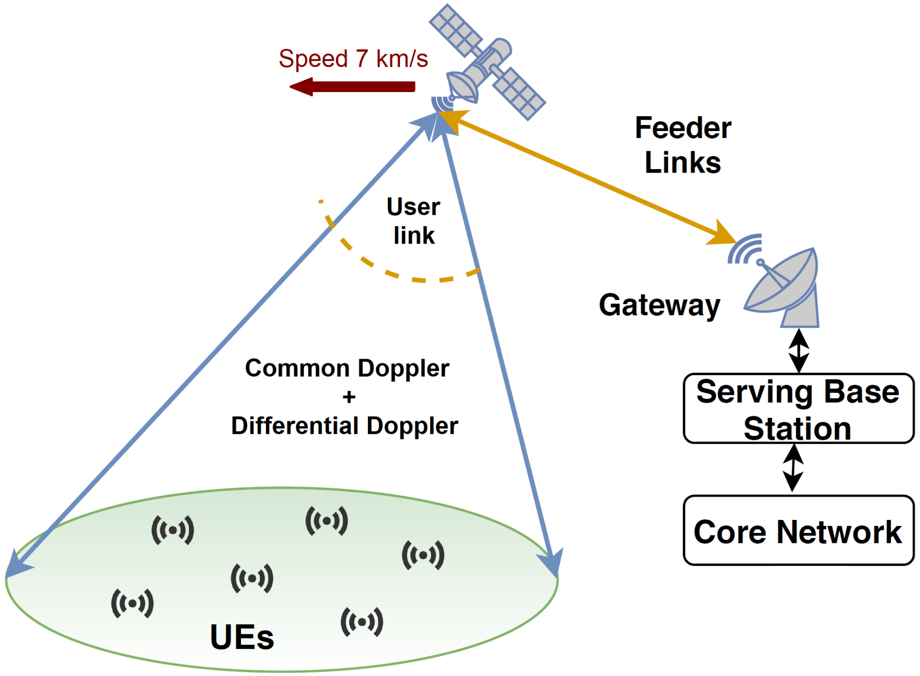
$$f_D(\varepsilon_i) = f_0 \frac{\omega_s R_E \cos(\varepsilon_i)}{c}$$

Typical Impairments in NTN: Differential Doppler shift

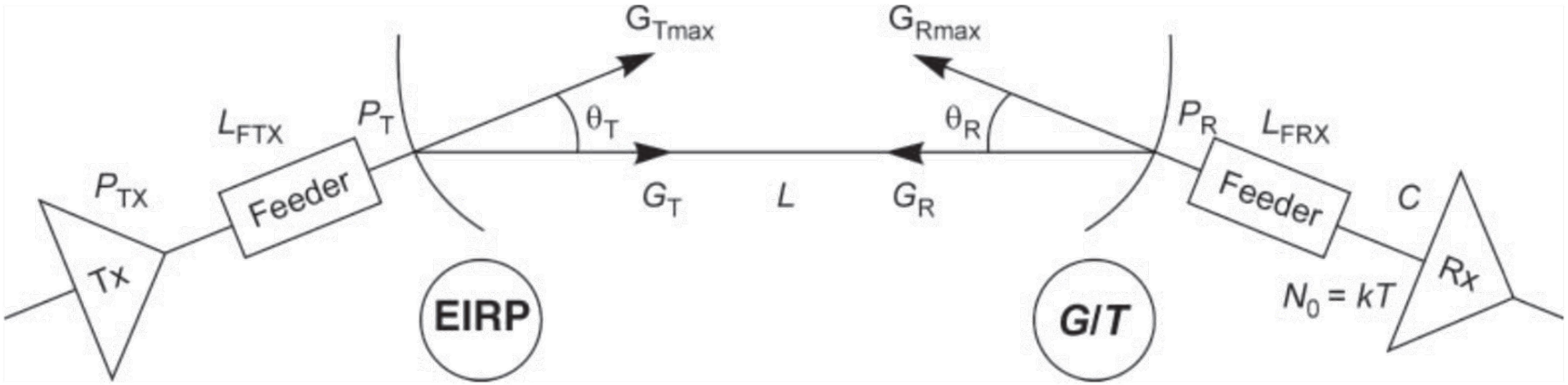
- The **differential Doppler shift** is the difference among the Doppler shift experienced by two different UTs in the access are of the same satellite.

- For two or more UTs in the same beam, it is possible to split their Doppler shift, $f_D(t)$, previously defined, into two distinct components:

$$f_D(t) = \Delta f_D(t) + f_D^{com}(t)$$



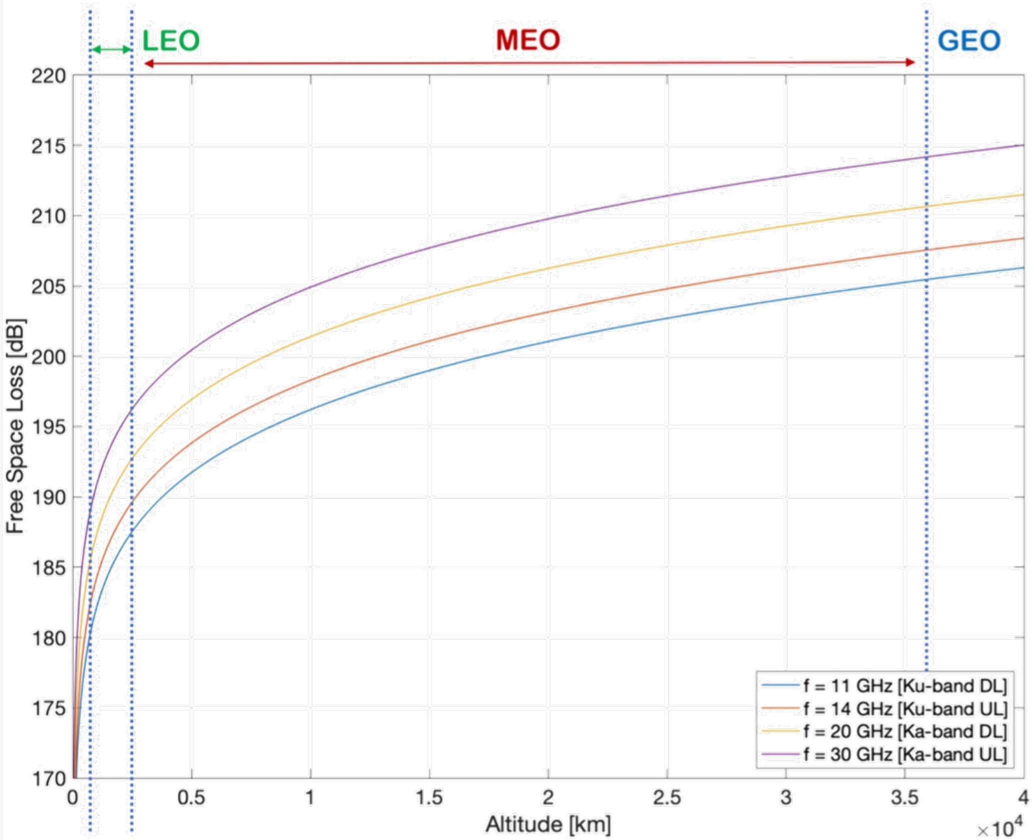
Typical Impairments in NTN: Link budget



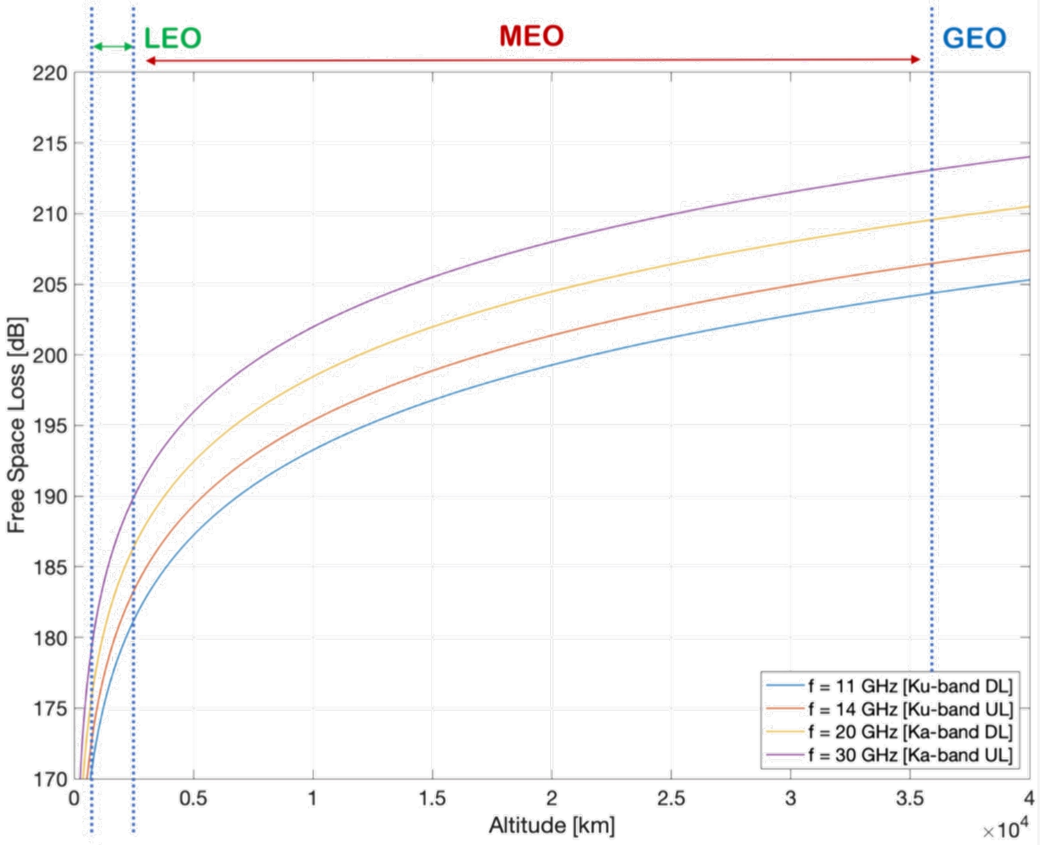
Link configuration

Source: G. Maral, M. Bousquet, “Satellite Communication Systems,” 5th ed., Wiley, 2009

Typical Impairments in NTN: Link budget



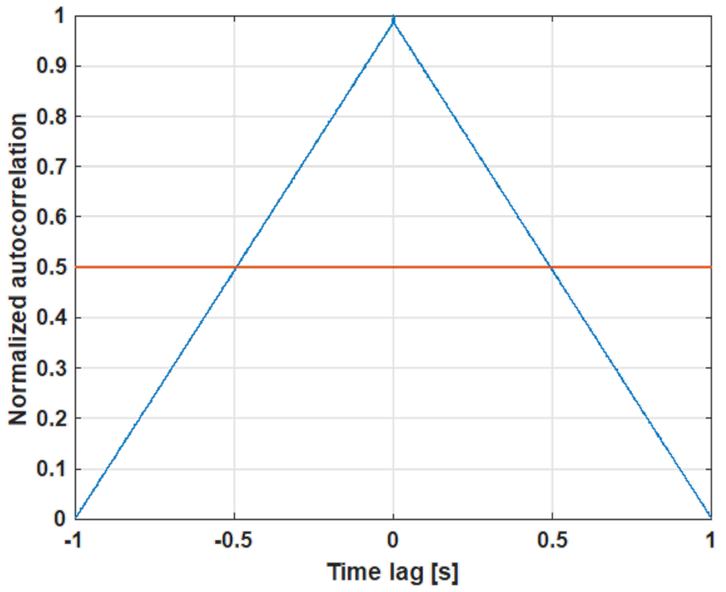
- FSL @ 10° Elevation



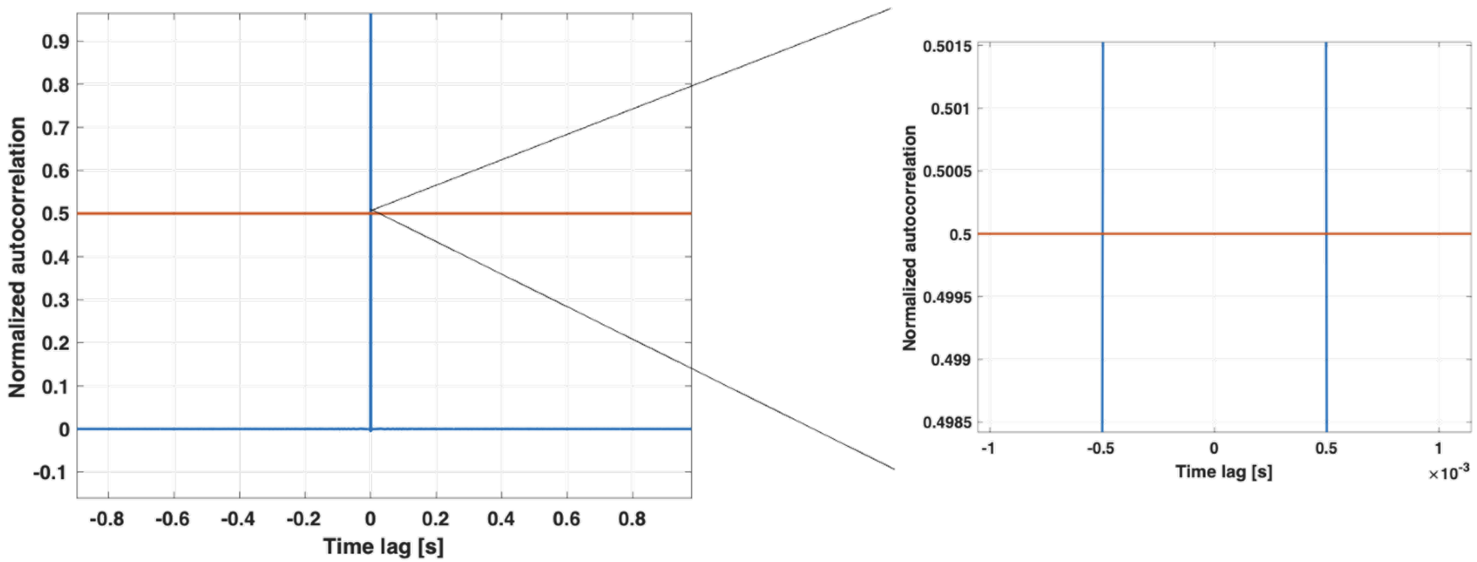
- FSL @ 90° Elevation

Typical Impairments in NTN: Fast-varying Channel

- Autocorrelation of the channel coefficients



Amplitude



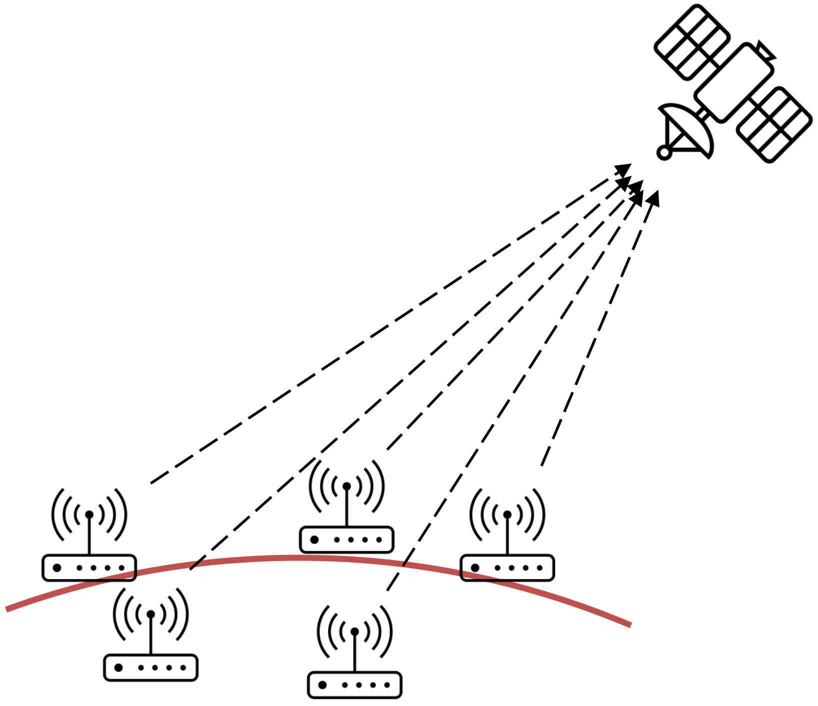
Phase

Case Study 1
AI-based Demodulator for Sparse Code Multiple Access

Overview and objective

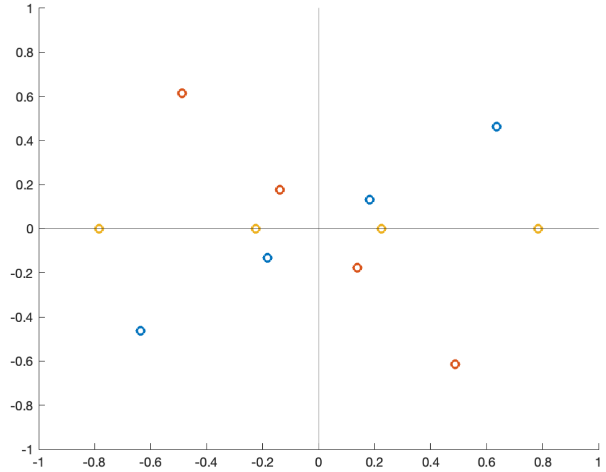
- **Massive radio access** expected from **IoT** devices
- NTNs: large coverage areas, short visibility window
- **Non-Orthogonal Multiple Access** techniques should be investigated!

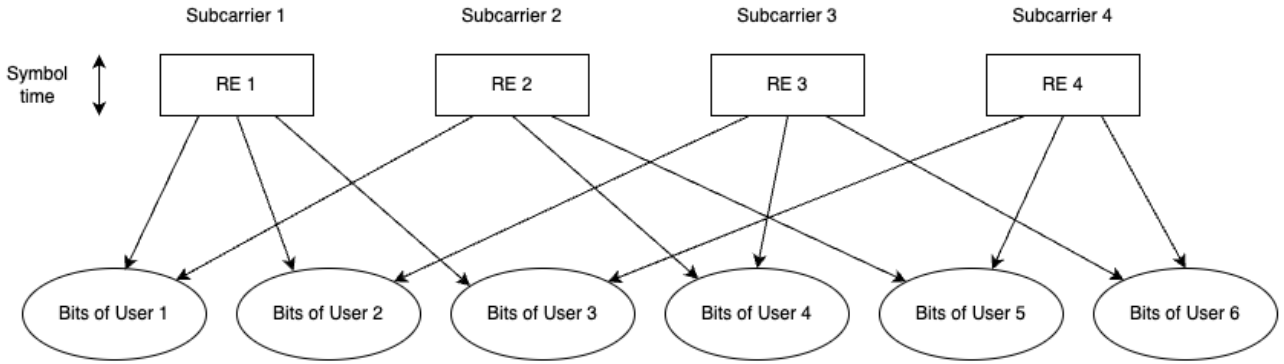
- IoT User Equipments (UEs) are required to be **low-power**
- **Objective**: Introduce a NN at the receiver to improve demodulation, achieving satisfactory performance at a lower SNR



Sparse Code Multiple Access

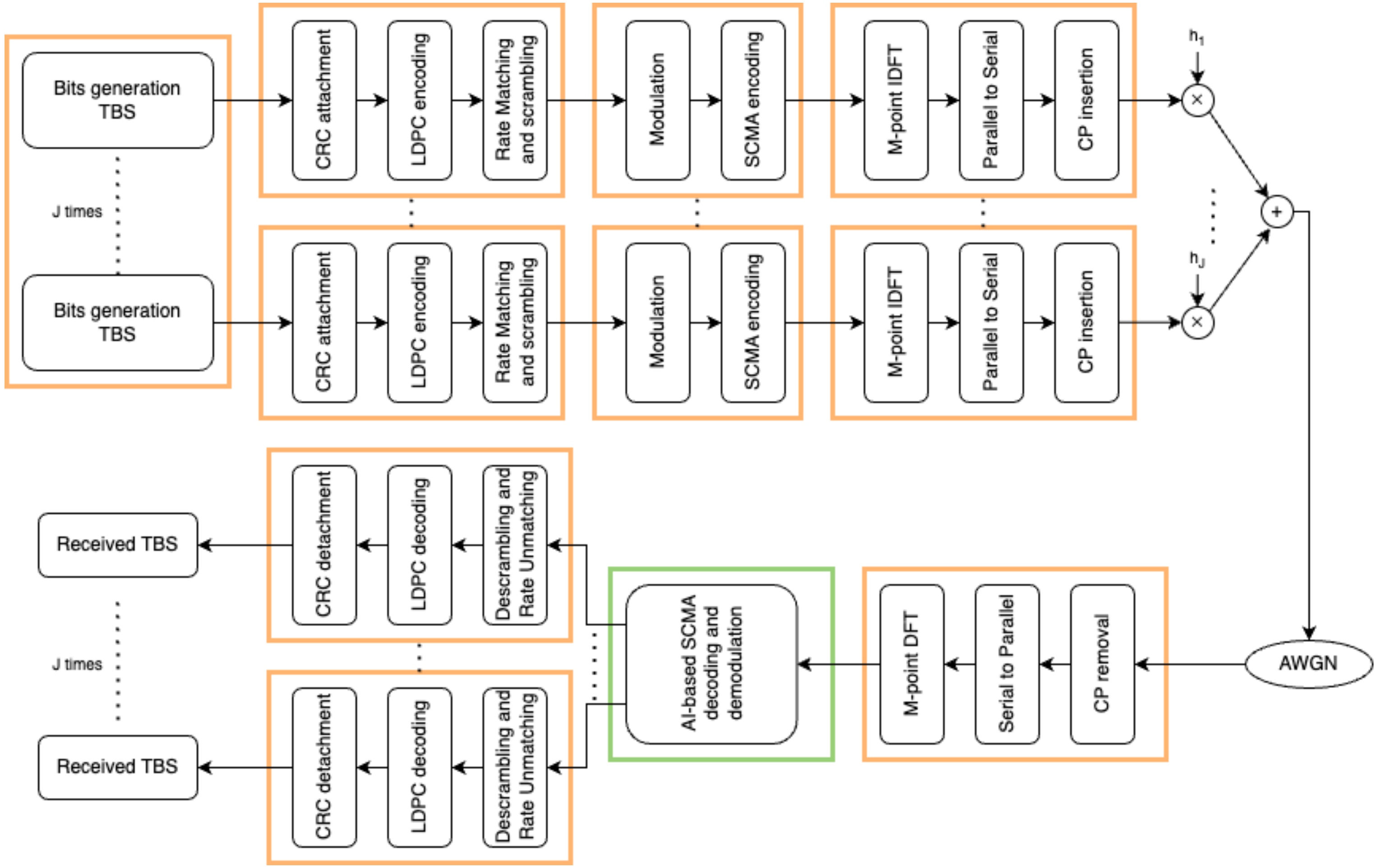
- **SCMA**: allow **limited and controlled overlapping** in frequency
  - Mapping between Resource Elements (REs) and UEs
  - Mapping between user bits and SCMA codewords (codebook)
- Each UE encodes 2 bits in a SCMA codeword
  - Two phase-shifted 4-ASK symbols, one on each RE
  - Phase shift is codebook-dependent
- Demodulator: Message-Passing Algorithm (MPA)
  - Iterative procedure (Log-MPA)
  - Exploit redundancy and phase shifts to separate non-orthogonal transmissions



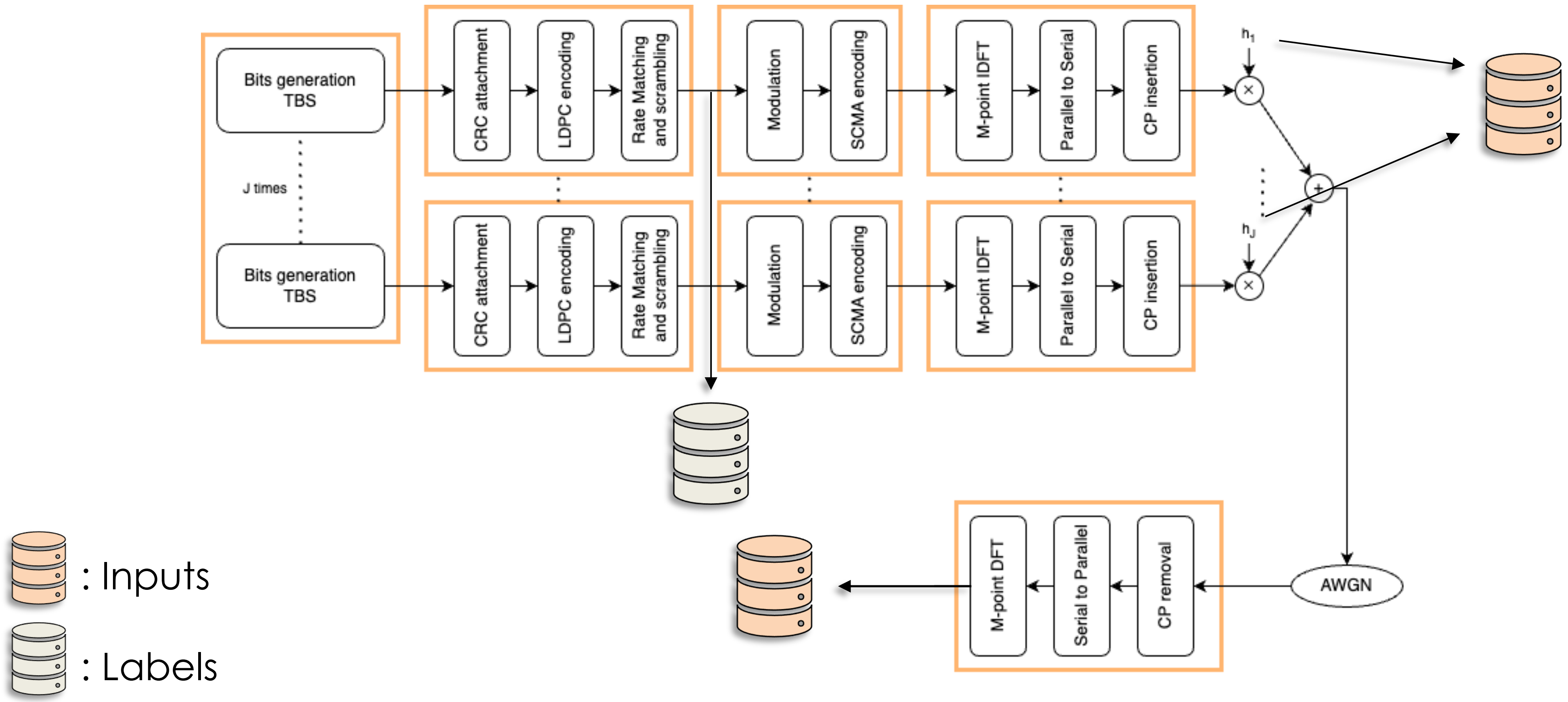


150% overload

Simulator overview



Dataset generation

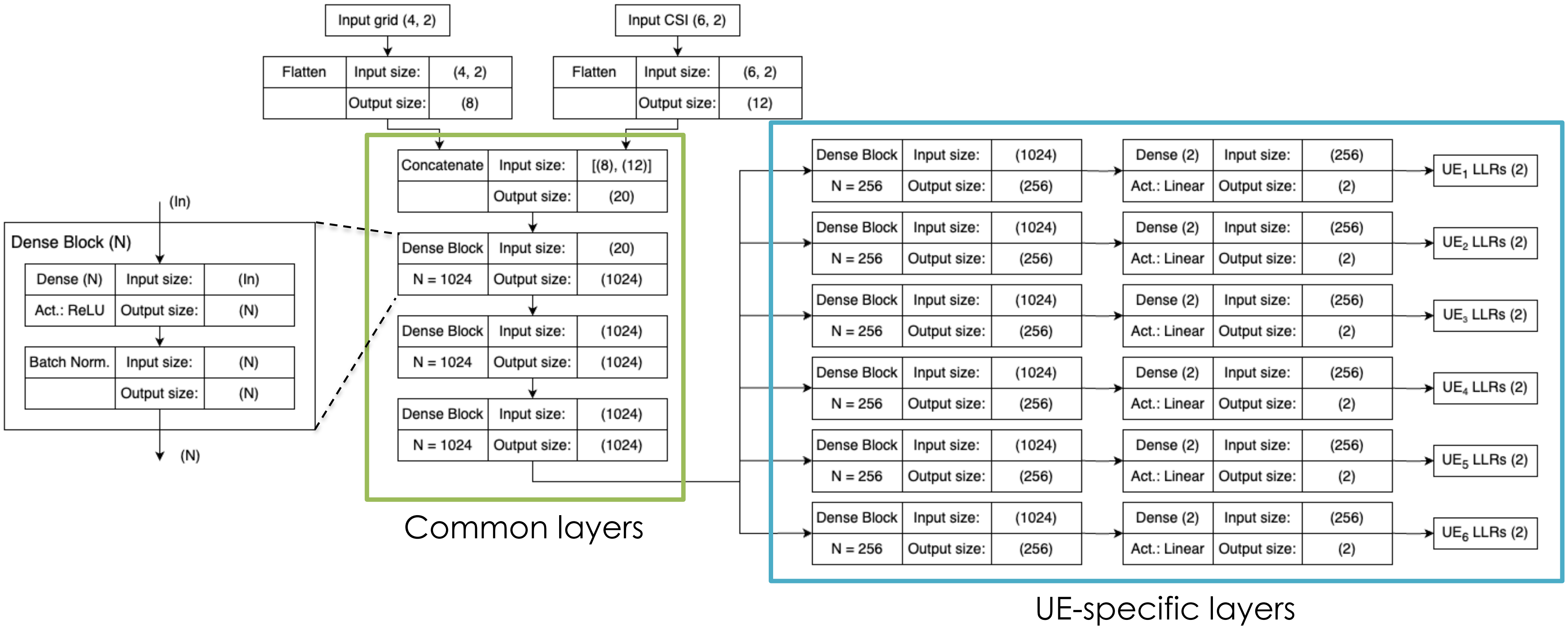


Dataset characteristics

- Entire dataset split into multiple files
  - **Handle large datasets** that would not fit in RAM
- Each row contains a transmission example: inputs (OFDM grid, channel coefficients) and labels
  - **Complex values are split** into real and imaginary parts
  - All values are **normalized** to improve the training process
  - **Different SNR levels** are considered
- Labels are bits, <u>not LLRs</u>!
  - Avoid learning to reproduce the traditional algorithm's output
  - The loss function will process the NN output to produce hard bits before comparing them with labels

	A	B	C	D	E	F	G	H	I	J	K	L
1	0.313294388800875,0.561228192604967,0.37149195463739,0.723610374066029,0.260398711302782,0.664131804596151,0.392271039521104,0.39931947061											
2	0.395535304946151,0.512153088617155,0.727891324404506,0.451982687129719,0.576602274651427,0.652395864396669,0.662284186946092,0.1590047712											
3	0.382576755398758,0.569711087910668,0.821827057737818,0.64491542918412,0.254696123343871,0.652291275036545,0.706361201589195,0.50869033602											
4	0.489601080379089,0.504829410601469,0.205710225752587,0.534931803913934,0.402675437238072,0.542816635481156,0.357874526409331,0.4102124163											
5	0.337262947743616,0.608954219845521,0.364721371453824,0.34697414222262,0.521775241096942,0.668580228368118,0.357424409593546,0.34933250290											
6	0.799365752128129,0.600954861671579,0.664131489162745,0.527041477651475,0.383432802304233,0.87333782192023,0.610261928147212,0.23779466520											
7	0.665150317699529,0.540916291496407,0.244178520414198,0.695242913294178,0.320759285528298,0.763898073987128,0.392872704325686,0.6527153519											
8	0.742534763127852,0.506390775383224,0.788031754592538,0.610021001206162,0.493930715146509,0.248296849405399,0.638069040856891,0.3198601203											
9	0.261951737082795,0.476082591094214,0.209745933032607,0.629189138213123,0.513941630109638,0.646460261369451,0.363990986116869,0.9210326962											

Neural Network overview



Dataset import (1/5)

```
def tf_data_generator(file_list, batch_size = 5):
    i = 0
    while True:
        if i*batch_size >= len(file_list):
            i = 0
            np.random.shuffle(file_list)
        else:
            file_chunk = file_list[i*batch_size:(i+1)*batch_size]
            data = []
            labels = []
            for file in file_chunk:
                temp = np.asarray(pd.read_csv(open(file,'r')))
                data.append(temp[:, :input_length])
                labels.append(temp[:, label_length:])

            data = np.asarray(data).reshape((-1, input_length))
            labels = np.asarray(labels).reshape((-1, label_length))

            yield data, labels
            i = i + 1
```

- Dataset is often too large to fit in RAM
- Generators can be used to yield data batch by batch

Dataset import (2/5)

```
def tf_data_generator(file_list, batch_size = 5):
    i = 0
    while True:
        if i*batch_size >= len(file_list):
            i = 0
            np.random.shuffle(file_list)
        else:
            file_chunk = file_list[i*batch_size:(i+1)*batch_size]
            data = []
            labels = []
            for file in file_chunk:
                temp = np.asarray(pd.read_csv(open(file,'r')))
                data.append(temp[:, :input_length])
                labels.append(temp[:, label_length:])

            data = np.asarray(data).reshape((-1, input_length))
            labels = np.asarray(labels).reshape((-1, label_length))

            yield data, labels
            i = i + 1
```

- Shuffle the files list every time the entire dataset has been yielded

Dataset import (3/5)

```
def tf_data_generator(file_list, batch_size = 5):
    i = 0
    while True:
        if i*batch_size >= len(file_list):
            i = 0
            np.random.shuffle(file_list)
        else:
            file_chunk = file_list[i*batch_size:(i+1)*batch_size]
            data = []
            labels = []
            for file in file_chunk:
                temp = np.asarray(pd.read_csv(open(file,'r')))
                data.append(temp[:, :input_length])
                labels.append(temp[:, label_length:])

            data = np.asarray(data).reshape((-1, input_length))
            labels = np.asarray(labels).reshape((-1, label_length))

            yield data, labels
            i = i + 1
```

- Select a batch of files
- Read each file
- Append the input data and the labels contained in each row to the corresponding lists
- Return the batch using *yield* to continue with the next batch

Dataset import (4/5)

```
# Create list of available training files
files_list = []
main_dir = "Dataset";
for path, subdirs, files in os.walk(main_dir):
    for name in files:
        files_list.append(os.path.join(path, name))
```

```
# Split files list into training and validation datasets
files_list_train, files_list_val = train_test_split(files_list, test_size = 0.2, random_state = 5)

# Training dataset generator
train_dataset = tf.data.Dataset.from_generator(tf_data_generator,
                                               args = [files_list_train, batch_size],
                                               output_types = (tf.float32, tf.float32),
                                               output_shapes = ((None, input_length), (None, output_length)))

# Validation dataset generator
val_dataset = tf.data.Dataset.from_generator(tf_data_generator,
                                             args = [files_list_val, batch_size],
                                             output_types = (tf.float32, tf.float32),
                                             output_shapes = ((None, input_length), (None, output_length)))
```

Dataset import (5/5)

```python
# Create list of available training files
files_list = []
main_dir = "Dataset";
for path, subdirs, files in os.walk(main_dir):
    for name in files:
        files_list.append(os.path.join(path, name))
```

```python
# Split files list into training and validation datasets
files_list_train, files_list_val = train_test_split(files_list, test_size = 0.2, random_state = 5)

# Training dataset generator
train_dataset = tf.data.Dataset.from_generator(tf_data_generator,
                                               args = [files_list_train, batch_size],
                                               output_types = (tf.float32, tf.float32),
                                               output_shapes = ((None, input_length), (None, output_length)))

# Validation dataset generator
val_dataset = tf.data.Dataset.from_generator(tf_data_generator,
                                             args = [files_list_val, batch_size],
                                             output_types = (tf.float32, tf.float32),
                                             output_shapes = ((None, input_length), (None, output_length)))
```

Fully-Connected Model (1/3)

```
input_shape = (input_length,)

# Input layers
fc_input = Input(shape = input_shape, name = 'Input_concat')

# Fully-Connected Layers (common grid processing)
x = Dense(size_common_fc, activation = 'relu', name = 'Common_Dense_1')(fc_input)
x = BatchNormalization(name = 'Common_BN_1')(x)
x = Dense(size_common_fc, activation = 'relu', name = 'Common_Dense_2')(x)
x = BatchNormalization(name = 'Common_BN_2')(x)
x = Dense(size_common_fc, activation = 'relu', name = 'Common_Dense_3')(x)
x = BatchNormalization(name = 'Common_BN_3')(x)
```

- *Input* layer collects input data from the batch
- *Dense* layer is a FC layer
- *BatchNormalization* layer standardizes the batch to zero mean and unit variance
  - *Helps the learning process*
- Each layer is connected to the previous, jointly processing the grid

Fully-Connected Model (2/3)

```
# Fully-Connected Layers (UE 1)
out_1 = Dense(size_UE_fc, activation = 'relu', name = 'UE1_Dense_1')(x)
out_1 = BatchNormalization(name = 'UE1_BN_1')(out_1)
out_1_final = Dense(2, activation = 'linear', name = 'UE1_Dense_2')(out_1)

# Fully-Connected Layers (UE 2)
out_2 = Dense(size_UE_fc, activation = 'relu', name = 'UE2_Dense_1')(x)
out_2 = BatchNormalization(name = 'UE2_BN_1')(out_2)
out_2_final = Dense(2, activation = 'linear', name = 'UE2_Dense_2')(out_2)

# Fully-Connected Layers (UE 3)
out_3 = Dense(size_UE_fc, activation = 'relu', name = 'UE3_Dense_1')(x)
out_3 = BatchNormalization(name = 'UE3_BN_1')(out_3)
out_3_final = Dense(2, activation = 'linear', name = 'UE3_Dense_2')(out_3)
```

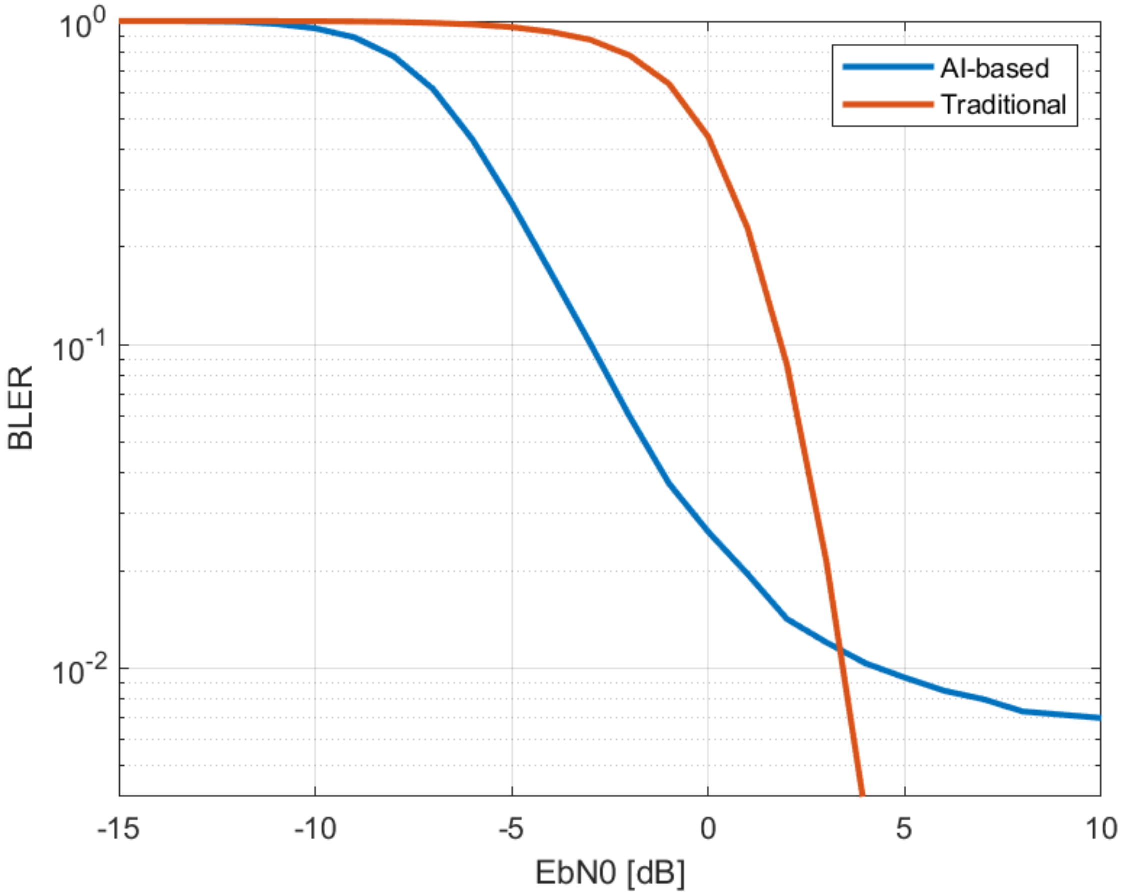
- Separate processing for each UE on the processed grid
  - Adapt each branch to the corresponding UE's codebook
- Smaller size (*size_UE_fc = 256*) to move closer to the output size (2 bits per UE)
- Linear activation to output LLRs
  - To be fed to a decoder
- NN output is the concatenation of the output LLRs

```
# Output concatenation
out_concat = Concatenate(axis = 1, name = 'Concatenate_outputs')([out_1_final, out_2_final, out_3_final, out_4_final, out_5_final, out_6_final])

# Model generation
model = Model(fc_input, out_concat)
model.summary()
```

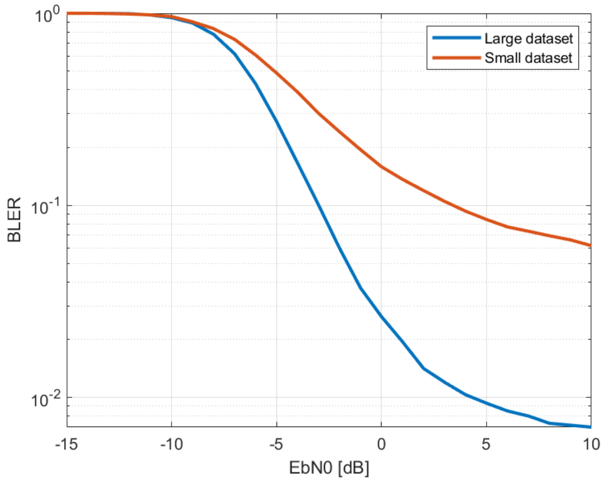
Fully-Connected Model (3/3)

```
def custom_loss(y_true, y_pred):
    # Compute hard bits from output LLRs, then compute Binary Crossentropy with labels
    y_pred_prob = K.activations.sigmoid(y_pred)
    eps = 1e-10     # Arbitrarily small value to avoid computing log(0) in some cases
    final_loss = -tf.reduce_mean(float(y_true) * tf.math.log(y_pred_prob + eps) +
                                 float(1 - y_true) * (1 - tf.math.log(y_pred_prob + eps)))
    return final_loss

# Compile model
model.compile(optimizer = tf.keras.optimizers.Adam(),
              loss = custom_loss);
```

Results – Block Error Rate

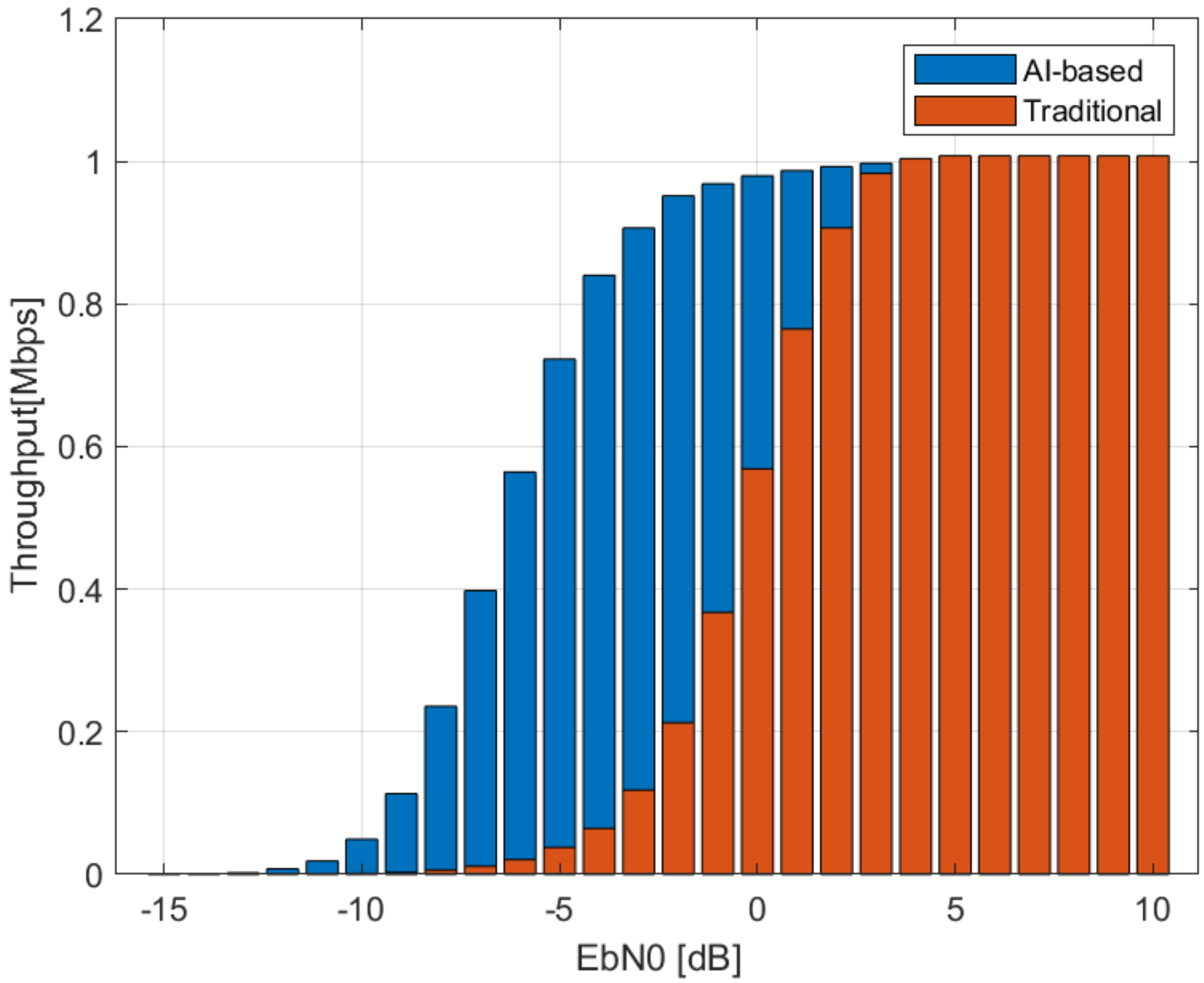


- **Better noise handling** with AI
  - 10% BLER at Eb/N0 = -3dB (AI) vs 2 dB (traditional)
  - 1% BLER at Eb/N0 = 4.5 dB (AI) vs 4 dB (traditional)
- **Error floor** over 1E-3
  - Suggests that non-orthogonality has not entirely been mitigated



40% smaller dataset

Results – Theoretical Throughput



- **Higher throughput** at low Eb/N0 with AI
  - 900kbps at Eb/N0 = -3dB (AI) vs 2 dB (traditional)
  - 1Mbps at Eb/N0 = 4 dB (AI and traditional)
- The BLER error floor is low enough to reach the 1Mbps peak throughput
- AI-based demodulator may reach the peak throughput at a lower Eb/N0 with further training

Complexity Analysis

Operation	AI-based	Traditional
Addition	3'702'796	34'840
Multiplication	3'702'796	9'456
Exponential	0	7'680
Maximum	4608	8'640

- Exponential function's complexity cannot be easily assessed
- AI-based mainly performs optimized matricial operations
- What is the **impact on demodulation time**?

With an off-the-shelves 12 cores CPU:

- AI-based: 3.05 ms
- Traditional: 0.89 ms

0.10 – 0.15 ms with GPU acceleration

Case Study 2
AI-based Channel Prediction

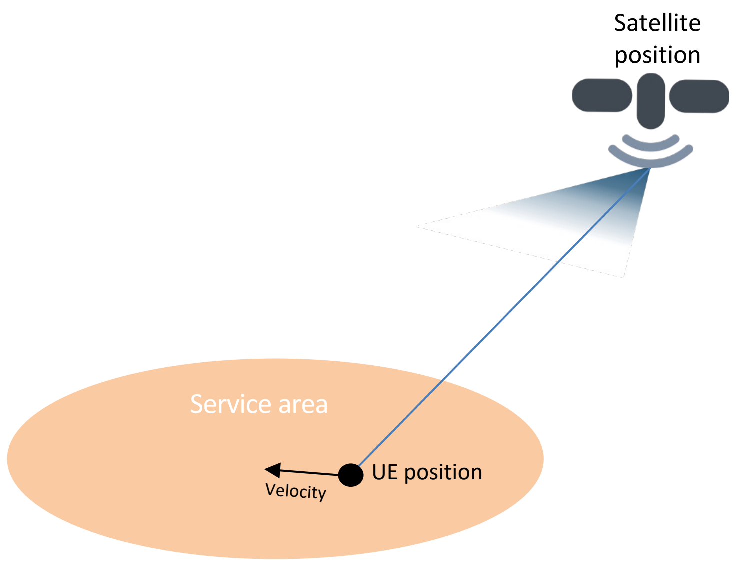
Overview and objective

Objective:

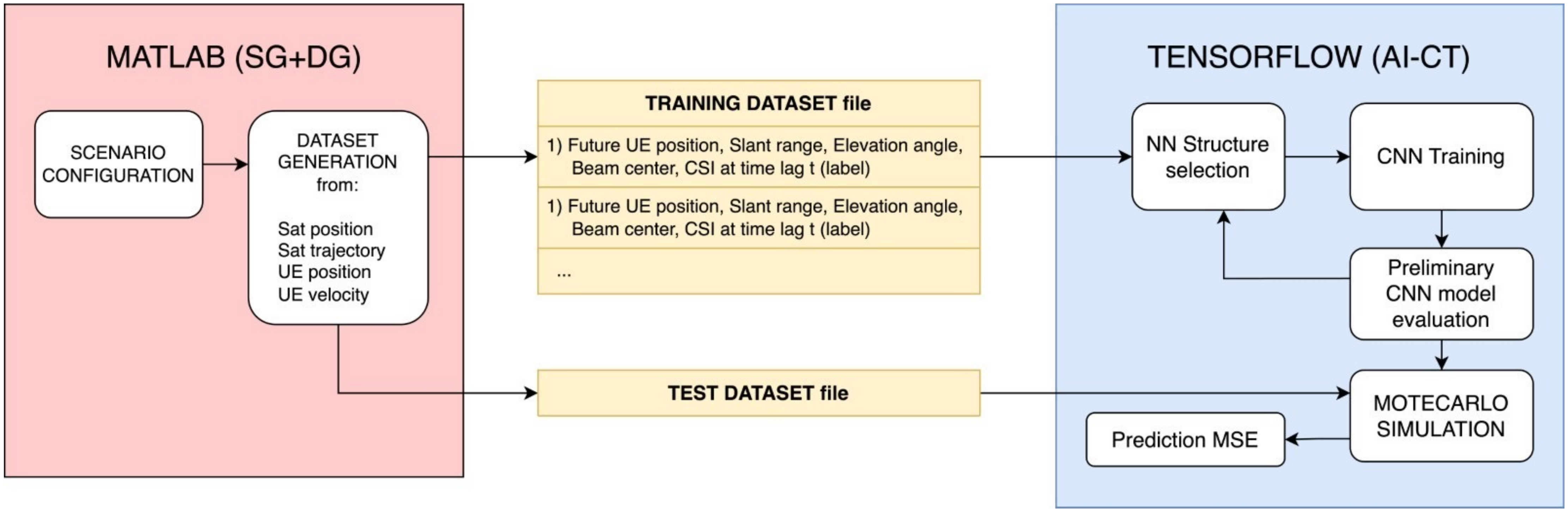
- Predict the future CSI of a specific UE in the coverage area
- Exploiting historical data about UEs CSI (training)
- Based on relative position and speed of UE and satellite (input)

Assumptions:

- NN implemented in the BS on-board
- UEs are moving (pedestrian, veicular, train)
- UEs provide their position to the BS
- NN trained with synthetic dataset



Simulator flowchart



Dataset generation

The dataset is generated in MATLAB following these steps:

- The UEs are deployed randomly in the coverage area.
- A random movement direction and velocity class is assigned to each UE.
- The trajectory of the satellite is computed.
- At each loop iteration:
  - The CSI of every UE is computed according to the desired channel model.
  - Each CSI value is saved in different row of the dataset file together with satellite position and UE position and velocity.
  - The position of each UE is updated according to its velocity.
  - The position of the satellite is updated according to its trajectory.

UE lat	UE lon	UE vel x	UE vel y	Sat ephemeris	CSI Real	CSI Imaginary

Dataset generation

Datasets generated with the following assumptions:

- **LEO at 600 km** (Set 2) with a single **fixed beam**
- **UL** transmission in **S band**
- **Sub-urban** environment in **LOS**
- One CSI example every **1ms**

Training dataset considering:

- UE density set to 1 UE/Km2

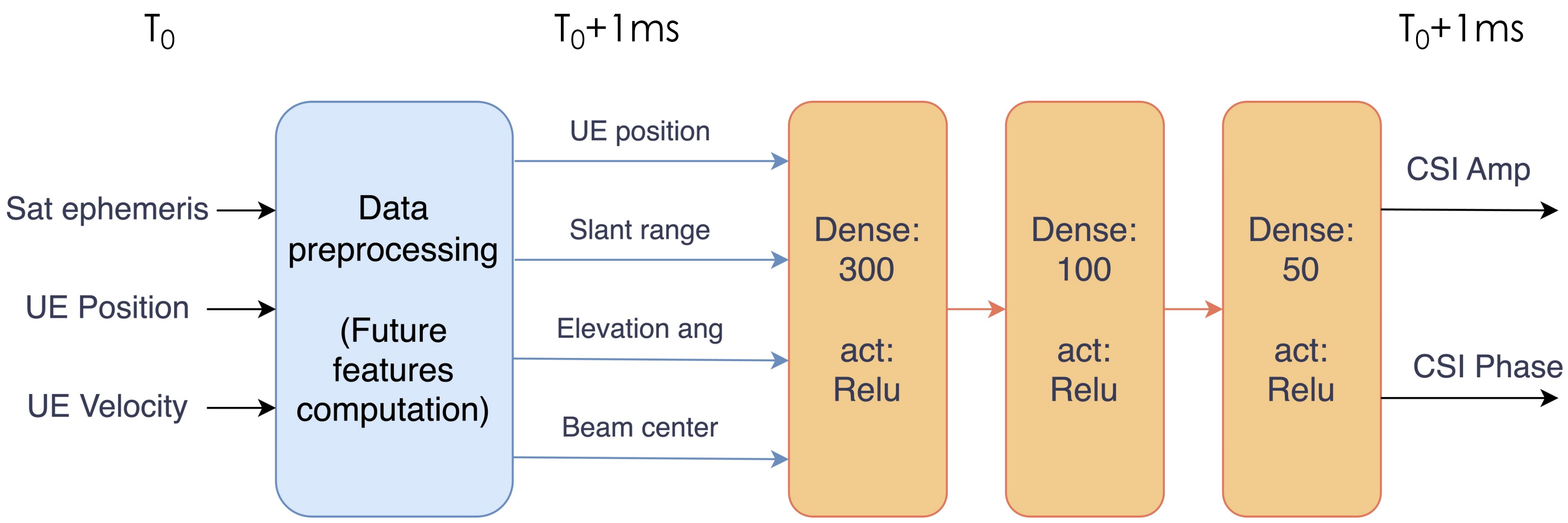
Generating 400M examples

Test dataset considering:

- UE density set to 0.1 UE/Km2

Generating 20M examples

NN Architecture



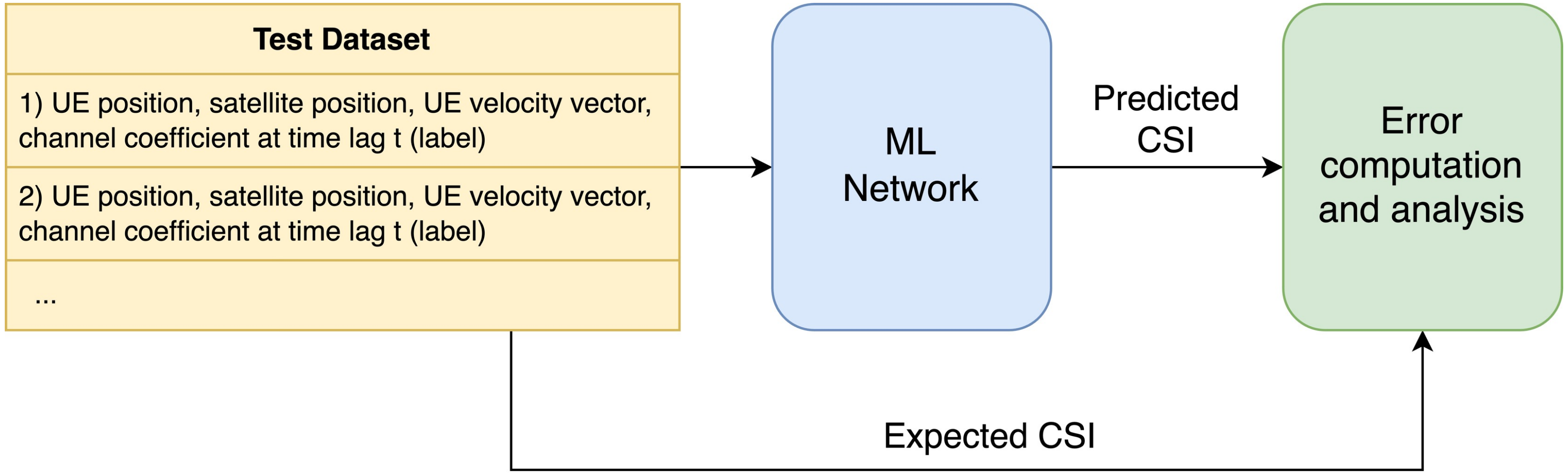
Parameter	Values
valid_split	0.3
learning_rate	0.001
N_epochs	10
batch_size	100
shuffle_seed	43
loss	mean_squared_error
output_metric	MSE

Loss function: Mean Square Error between future CSI (label) and predicted CSI

ML network testing phase (1/3)

To assess the performances of the trained ML network:

- A new dataset is fed to the network
- The network output is compared to the expected output
- A statistical evaluation of the network errors is performed.



ML network testing phase (2/3)

```
" Select the path to the test dataset "
urltest = './CSI_dataset_E0600_set1_S_0tier_fixed_5ms/LEO600_set2_S_0tier_moving_4.170000e-03ms.csv'

" Import of the dataset used to test the network "
npDataTest = csvr.input_csv_filter_t(urltest)

" Generate the training and validation dataset from the input numpy data "
test_dataset, _, _, _, _, _, _, value_labels = creData.input_data_to_dataset_t(npDataTest)
test_dataset = test_dataset.batch(batch_size)

" Import the previously saved AI model "
model = keras.models.load_model('CSI_AI_model')

" Input the test dataset to the AI model and obtain the output prediction "
labels_predict = model.predict(test_dataset, 1)
```

Data import

Model import

Model use

ML network testing phase (3/3)

Example of error computation

```
" Otput metrics computation and print "
if output_metric == 'MSE':
    CSI_prediction_MSE = mean_squared_error(labels_predict[:, 0, 0], value_labels[:, 0])
    print("MSE:%.4f" % CSI_prediction_MSE)
elif output_metric == 'MAE':
    CSI_prediction_MAE = mean_absolute_error(labels_predict[:, 0, 0], value_labels[:, 0])
    print("MAE:%.4f" % CSI_prediction_MAE)
elif output_metric == 'MAPE':
    CSI_prediction_MAPE = mean_absolute_percentage_error(labels_predict[:, 0, 0], value_labels[:, 0])
    print("MAPE:%.4f" % CSI_prediction_MAPE)
```

Results: MSE

- MSE of the amplitude of the complex CSIs
- MSE of the phase of the complex CSIs
- Aggregated MSE of amplitude and phase

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(CSI_i - \widehat{CSI_i}\right)^2$$

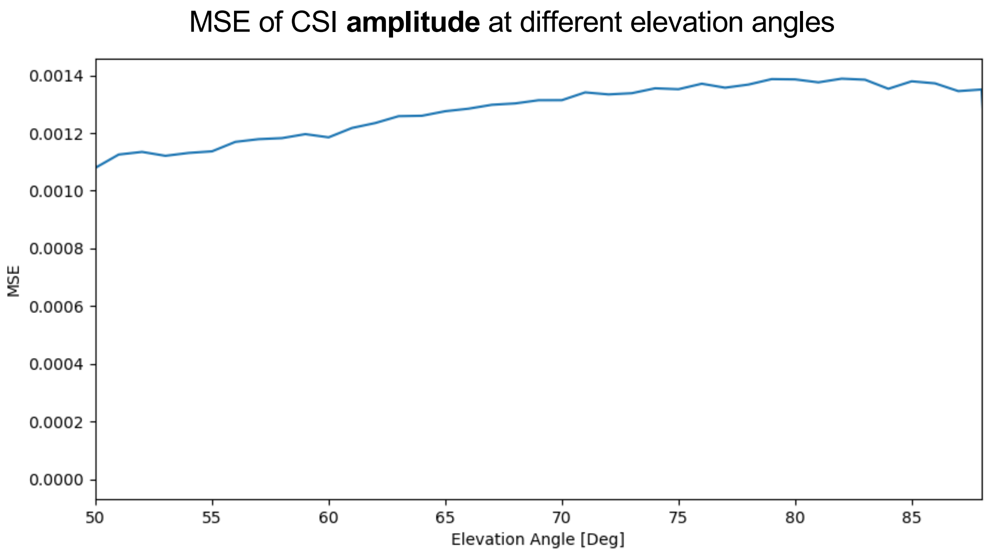
Metric	Value
Amplitude MSE	0,0012
Phase MSE	0,0843
Aggregate MSE	0,0836

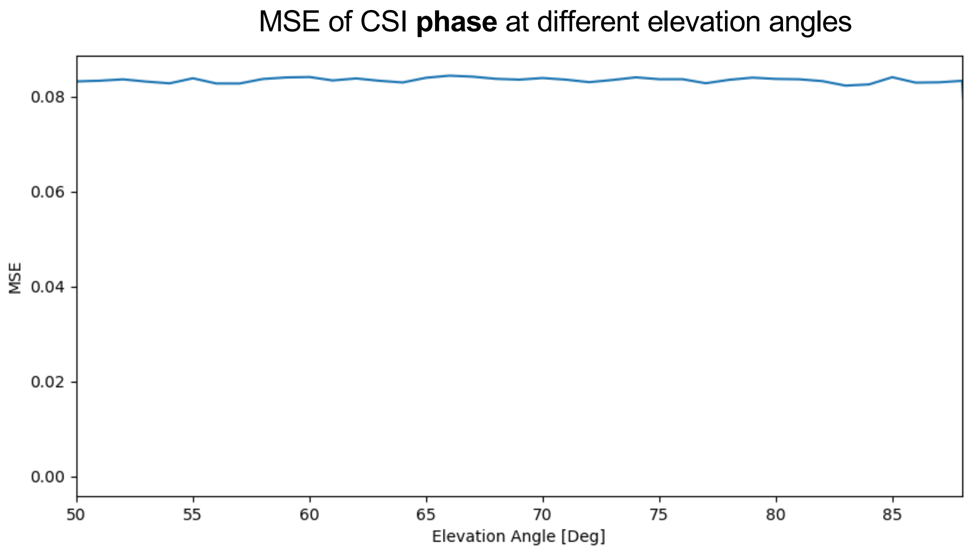
Better performances in the amplitude prediction compared to phase prediction:

- High correlation between amplitude of consecutive examples
- Low correlation between phase of consecutive examples

Results: MSE vs Elevation Angle

Analysis of CSI prediction performance variation along the orbit and inside the beam





- Quite constant CSI prediction performance at different elevation angles
- The NN can be proficiently exploited in extended coverage areas without losing prediction precision

Enabling NOMA NTN through CSI Prediction

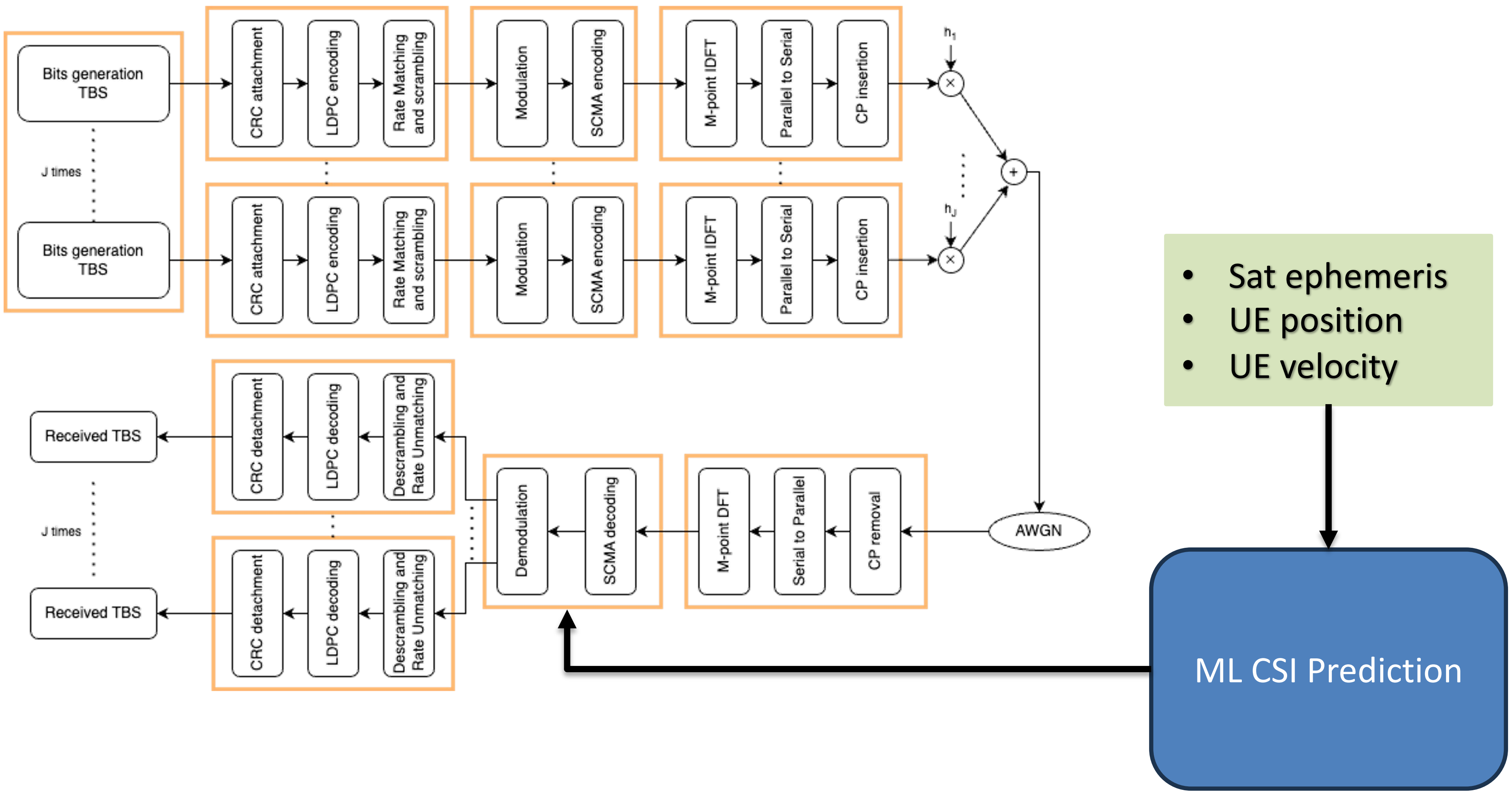
NOMA thechnique: The gNB requires the **knowledge of the CSI at each symbol time**.

Performance of NOMA technique knowing only the CSI of the first symbol of each user:

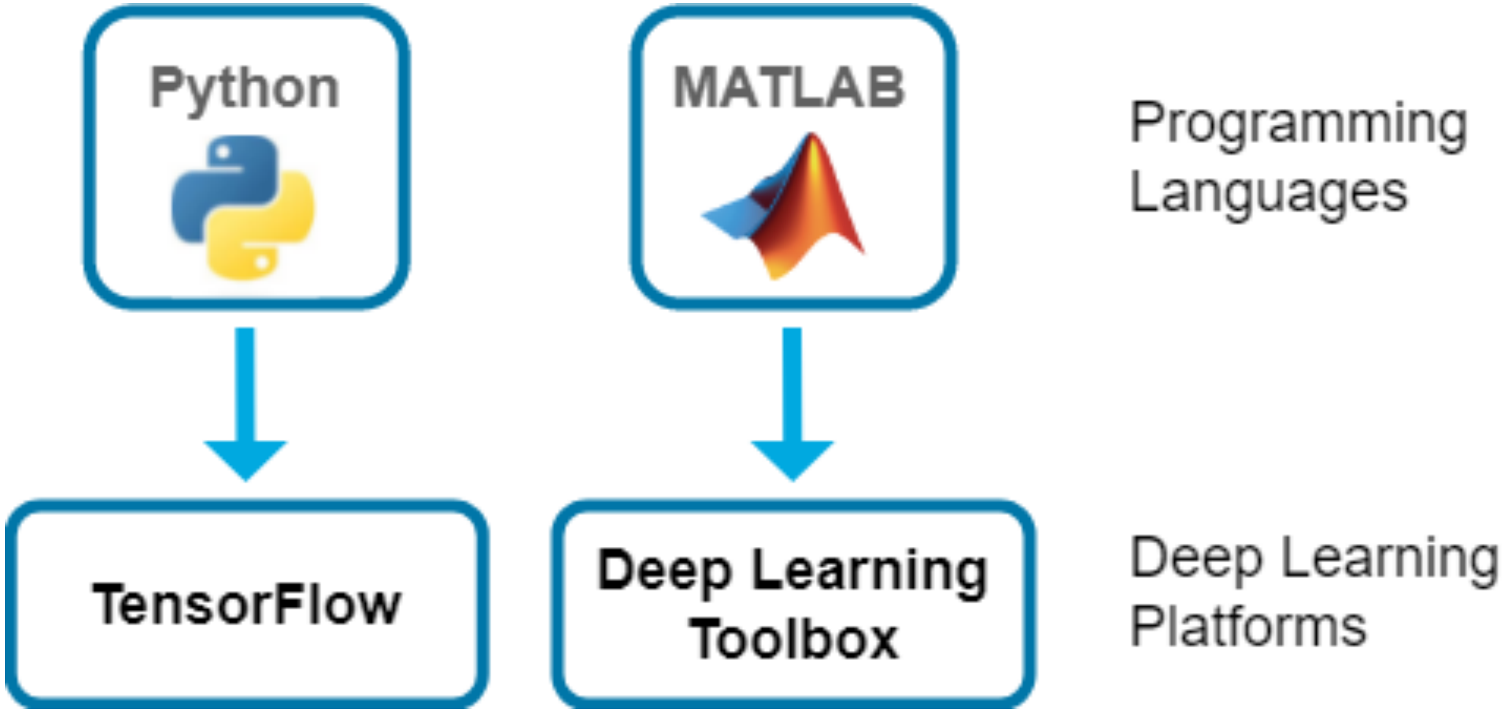
EbN0 [dB]	BLER (SCS = 15 kHz)	BER(SCS = 15 kHz)	BLER (SCS= 240 kHz)	BER(SCS = 240 kHz)
0	0.99	0.4358	0.99	0.2740
2	0.99	0.4256	0.99	0.2649
4	0.99	0.4232	0.99	0.2624
6	0.99	0.4217	0.99	0.2618

CSI prediction technique can enable the correct decoding of NOMA packets.

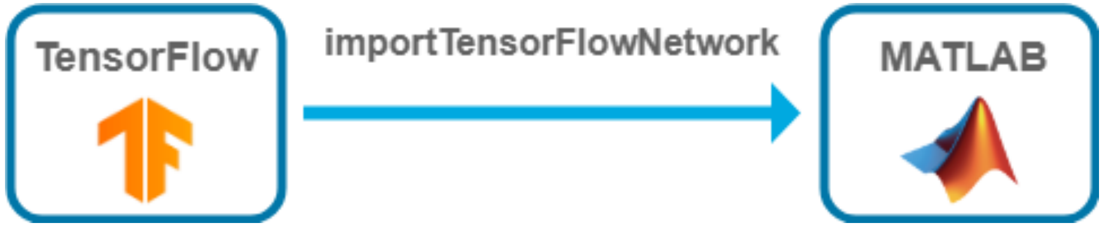
CSI prediction in NOMA SCMA demodulator



TF model exploitation in MATLAB (1/2)



TF model exploitation in MATLAB (2/2)



1) Save the tensorFlow model in the SavedModel format:

```
import tensorflow as tf
  tf.saved_model.save(model.modelFolder)
```

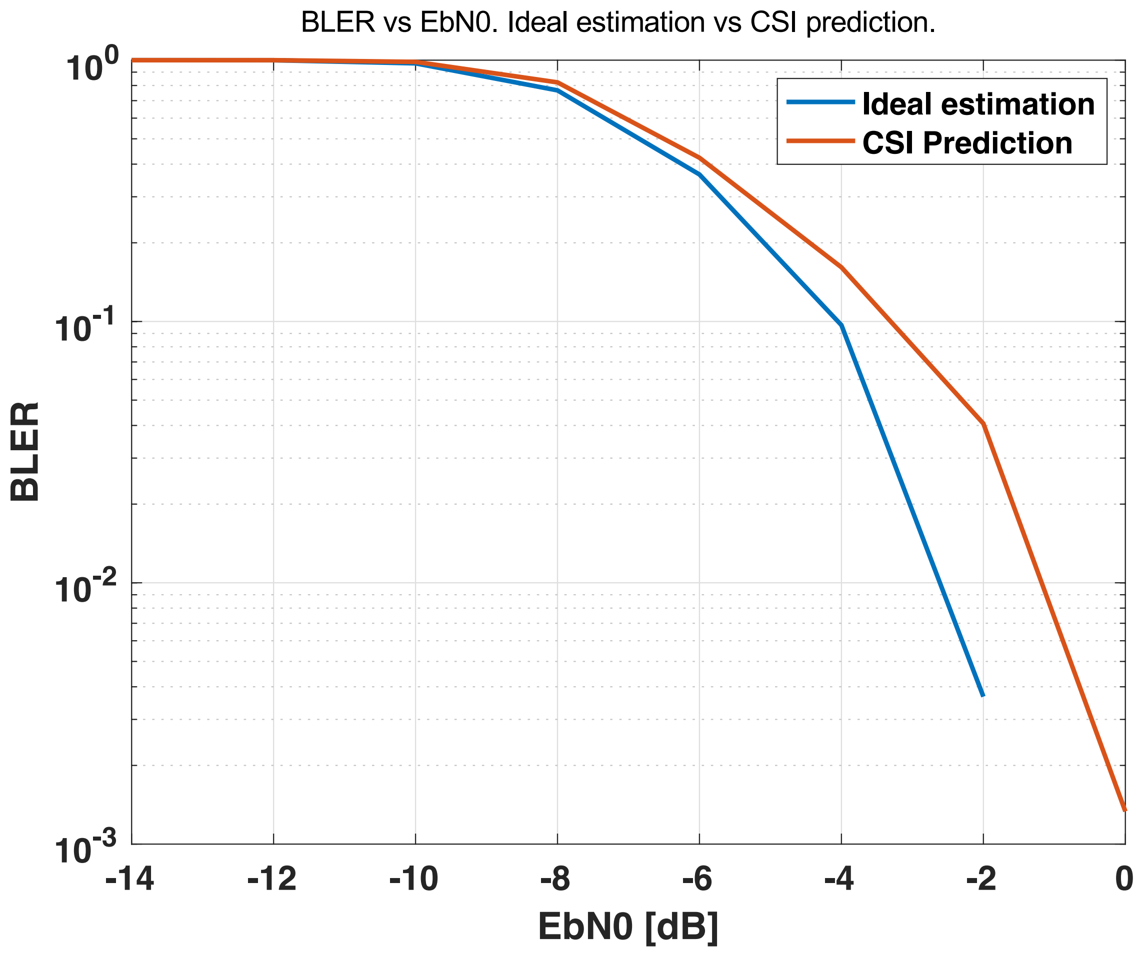
2) Import the TensorFlow model into MATLAB by using the MATLAB function:

```
modelFolder = "CSI_Prediction";
net = importTensorFlowNetwork(modelFolder,OutputLayerType="regression");
```

CSI prediction in NOMA SCMA demodulator

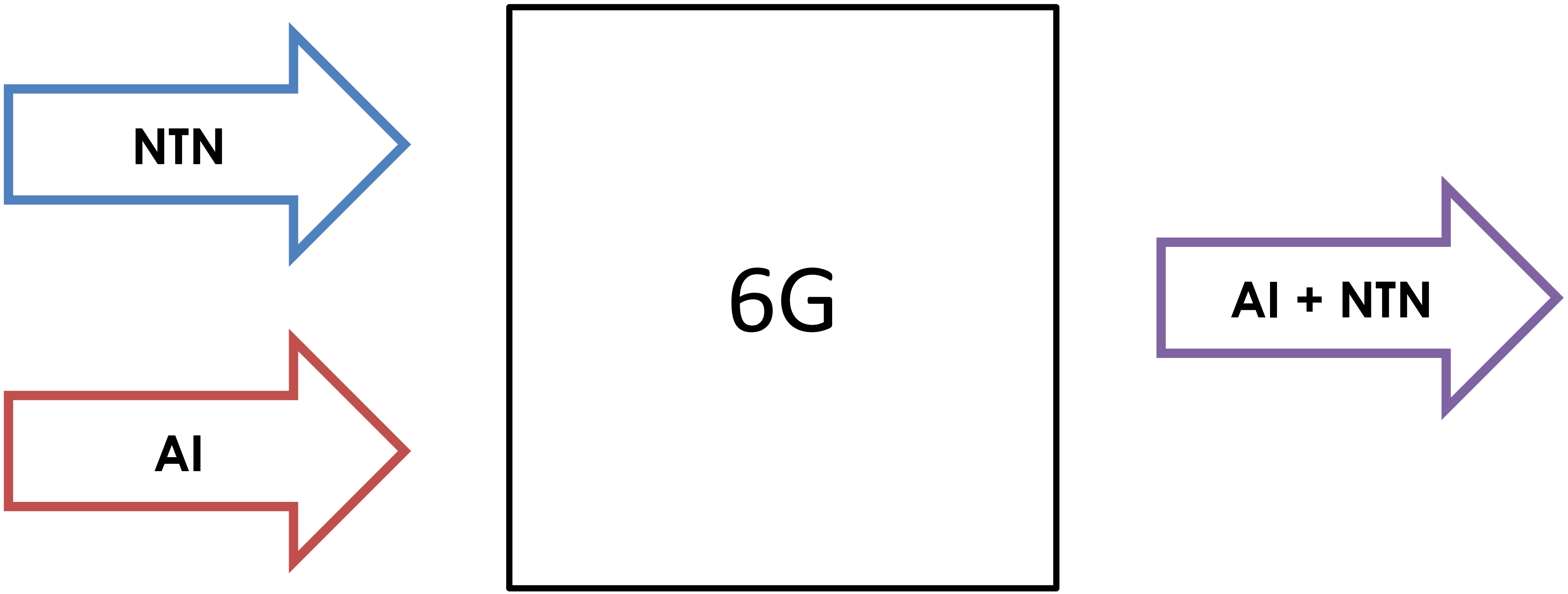
The CSI prediction obtained with the Neural Network allows to reach a **BLER very close** to the one in **ideal conditions**.

This is motivated by the amplitude of the phase error not exceeding the robustness of the SCMA technique



The Way Forward for AI in NTN

The Way Forward for AI in NTN



Challenges for AI in NTN

- **Data availability**
  - Are synthetic data enough?
- **Computational complexity**
  - Power, latency
- **Model adaptability**
  - Multiple specialized models vs one larger model
- **Real-world testing**
  - Deploy a NN in a network

Current funded projects on NTN

6GNTN

https://www.6g-ntn.eu/

https://www.linkedin.com/company/6g-ntn/

https://twitter.com/6Gntn

EAGER

https://www.eagerproject.eu

https://www.linkedin.com/company/eager-project/

https://twitter.com/eagersatcom

5G stardust

https://www.5g-stardust.eu

https://www.linkedin.com/company/5g-stardust/

Q&A