

# Gigantic MU-MIMO: Toward Channel Statistics Independence in ML Receivers

## 1. Introduction

- 1.1. In this project, we explored, extended, and implemented Viterbi-model-based approaches and DNN architectures for learning the underlying statistics of wireless fading channel communication which obeys a Markovian stochastic input-output relationship.

Based on the main paper “**ViterbiNet: A Deep Learning Based Viterbi Algorithm for Symbol Detection**” by Nir Shlezinger, Nariman Farsad, Yonina C. Eldar, and Andrea J. Goldsmith

## 2. Files Structure and Uses

- 2.1. **Code** folder contains all code subdirectories and files implemented in python
  - 2.1.1. **channel** folder contains data generation code
    - 2.1.1.1. **channel.py** – contains ISI AWG transmit function
    - 2.1.1.2. **channel\_dataset.py** – contains the channel data generation class
    - 2.1.1.3. **channel\_estimation.py** – contains the channel method and costs
    - 2.1.1.4. **modulator.py** – contains the BPSK modulation function
  - 2.1.2. **ecc** folder contains the error correction, encoding, decoding files which based on Reed-Solomon algorithm  
[https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon\\_codes\\_for\\_coders](https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders)
  - 2.1.3. **dir\_definitions.py** – includes all project directories and sub-directories
  - 2.1.4. **detector.py** – contains the Detector class which responsible for the DNN/Statistical models and methods (ModelBased / EndToEnd / Statistical) including the Viterbi algorithm
  - 2.1.5. **models.py** – contains all the project models ADNN / Sionna / SionnaPlus / Transformer / LSTM / ViterbiNet / ClassicViterbi additional to experimental models, implemented using PyTorch.
  - 2.1.6. **trainer.py** – contains the Trainer class which responsible for all training and evaluation flow for a given model including save/load model, configurate loss and optimizer, initialize channel parameters and data, run train loop and backprop, training evaluation, and online evaluation.
  - 2.1.7. **plotter.py** – contains the plot function which creates graphs based on the models results such as “ser by block index” or “ser by snr”
  - 2.1.8. **configuration.yaml** – includes configurable project parameters such as channel parameters / train and validation hyper-parameters / loss type and available optimizers.
  - 2.1.9. **main.py** – responsible for running the project, by looping over SNR list from 7 to 15 cross all models and controls all phases of training, evaluation, and graphs using the “execute\_and\_plot” function. In addition, it contains “HYPERPARAMS\_DICT” with configurable parameters and main flags.

2.1.9.1. **Important “HYPERPARAMS\_DICT” configurable keys:**

2.1.9.1.1. **HYPERPARAMS\_DICT [‘val\_frames’]**

**HYPERPARAMS\_DICT [‘subframes\_in\_frame’]**

Their multiplication result determines the Minibatch size during training.

2.1.9.1.2. **HYPERPARAMS\_DICT [‘self\_supervised\_iterations’]**

determine the number of self (online) training iterations on the correctly detected block during online evaluation

2.1.9.1.3. **HYPERPARAMS\_DICT [‘train\_minibatch\_num’]**

determines the number of Minibatches during training

2.1.9.2. **Important main flags:**

2.1.9.2.1. **run\_over** – value = 0 load plots from previous runs, else value = 1 load trained weights and start online evaluation, else value = 2 clear all and start training from scratch.

2.1.9.2.2. **plot\_by\_block** – once set to **True** the project will generate “ser by bock index” plot else set to **False** the project will generate “ser by snr” plot.

2.1.9.2.3. **block\_lenght** – determine the transmission length of each block i.e. the number of bits.

2.1.9.2.4. **channel\_coefficients** - 'time\_decay' / 'cost2100'  
determine the channel cost type.

2.1.9.2.5. **snr\_start, snr\_end** – determine the range of SNRs

2.1.9.2.6. **models\_list** – contains list of DNN models which will be part of the detector

2.1.9.2.7. **detector\_method** – determine the detector methodology  
'ModelBased' for Viterbi based and llr learning /  
'EndToEnd' for bit to bit learning without Viterbi /  
Statistical used only for the 'ClassicViterbi' model which is the statistical Viterbi algorithm with perfect CSI.

2.1.9.2.8. **self\_supervised** - True/False for online evaluation enable

**\*\* Note -- every parameter configured in the main.py  
\*\* is the last that counts and overwrites configuration.yaml  
\*\* file values**

2.2. **Resources** folder contains the channel coefficients vectors for cost2100 (4 taps, each with 300 blocks).

2.3. **Results** folder

2.3.1. **figures** folder – contains all the saved graphs images

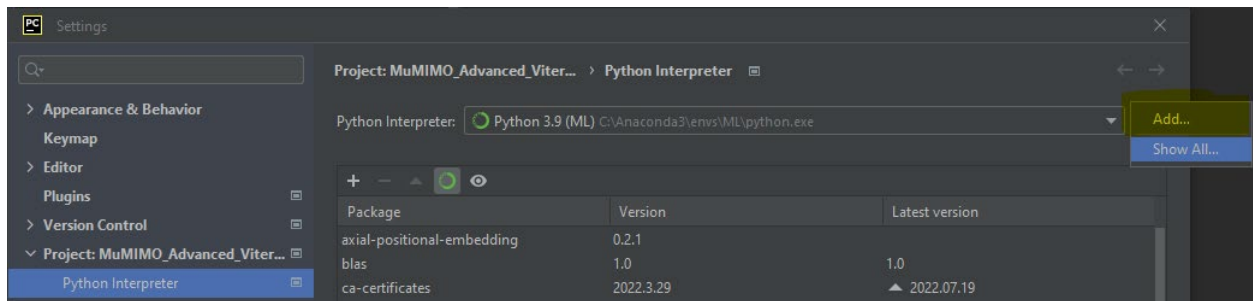
2.3.2. **plots** folder – contains all the saved plots data

2.3.3. **weights** folder – contains all the trained models' weights per channel cost and each by SNR and gamma.

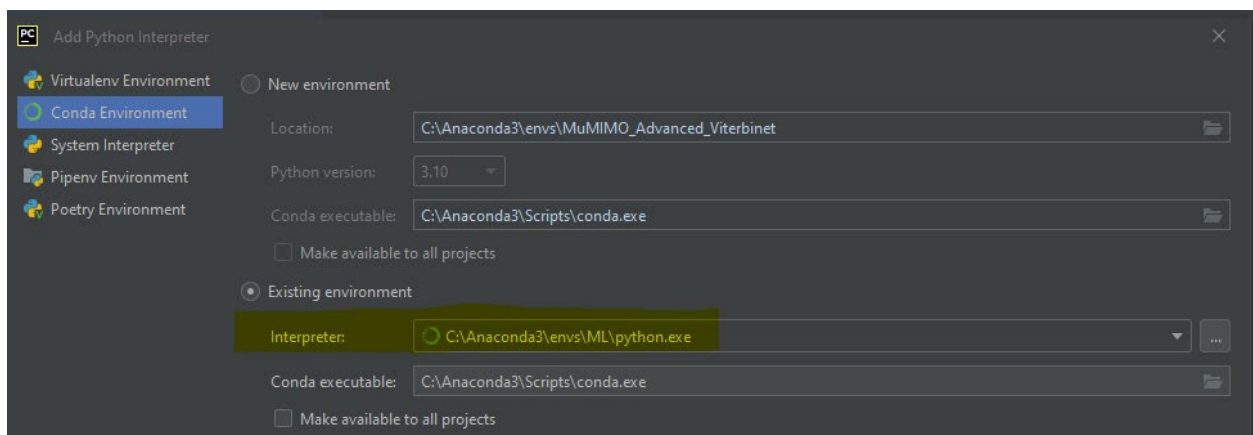
- 2.4. **project\_env.yml** – conda environment with all related packages and modules  
run the command "conda env create -f project\_env.yml " to create the project env

### 3. Execution

- 3.1. In order to execute the project first make sure you have Anaconda and PyCharm (IDE) installed, then install the project\_env.yml  
# --- console command "conda env create -f project\_env.yml " --- #  
Next follow instructions:
- 3.1.1. Open **PyCharm** in the project root directory
- 3.1.2. Go to, File → Settings → Python Interpreter → Add



- 3.1.3. Select the **Conda Environment** that created from the project\_env.yml file



- 3.1.4. For windows the conda env usually found at  
C:\Anaconda3\envs\<env\_name>\python.exe
- 3.2. Now you can run the **Code\main.py** file to execute the project.

## 4. Run and Modify Project

4.1. As described above at the “Files Structure and Uses” section, the **Code/main.py** file runs and controls all the project aspects, therefore, most of the running, plotting, training, evaluation and data configuration in it. So please look on the **main.py** sub-section, including all the relevant flags and parameters.

**\*\* All the most important running configuration described there \*\***

4.2. Use the “run\_over” flag to:

4.2.1. **run\_over = 0** → **load all plots** from previous results

4.2.2. **run\_over = 1** → **load trained weight** and start **online evaluation**

4.2.3. **run\_over = 2** → **clear all** results and **train** from scratch

```
20 HYPERPARAMS_DICT = {
21     'noisy_est_var': 0,
22     'fading_taps_type': 1, # 1 / 2 for time decay only
23     'fading_in_channel': True,
24     'fading_in_decoder': True,
25     'gamma': 0.2,
26     'channel_type': 'ISI_AWGN',
27     'val_frames': 12, # up to 12 for cost2100
28     'subframes_in_frame': 25, # up to 25 for cost2100
29     'self_supervised_iterations': 200,
30     'ser_thresh': 0.02, # ser threshold for online training
31     'train_minibatch_num': 25, # 25
32 }
33
34
35 if __name__ == '__main__':
36     # main flags
37     run_over = 0 # 0 - load plots from previous runs / 1 - load trained weights and start online evaluation / 2 - clear all and start training from scratch
38     plot_by_block = False # False / True either plot by SNR or by block index
39     block_length = 120 # determine the transmission length
40     channel_coefficients = 'cost2100' # 'time_decay' / 'cost2100'
41     n_symbol = 2
42     snr_start, snr_end = 7, 15
43
44     # deep learning models list 'ADNN', 'Sionna', 'SionnaPlus', 'Transformer', 'LSTM', 'ViterbiNet'
45     models_list = ['ADNN', 'Sionna', 'SionnaPlus', 'Transformer', 'LSTM', 'ViterbiNet']
46     detector_method = 'ModelBased' # ModelBased / EndToEnd / Statistical
47     self_supervised = True # True / False for online evaluation enablement
```

Enjoy!