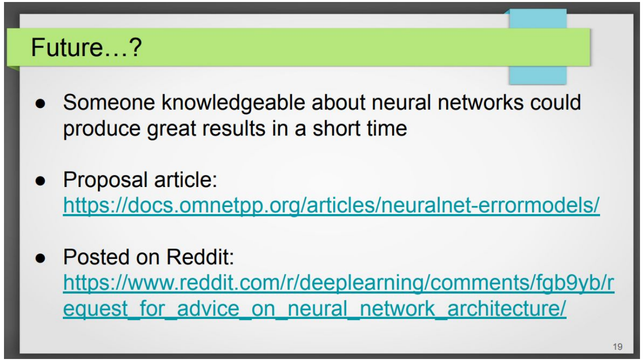
Inference of packet-error rates using SNIR patterns via neural networks

Chimera Solutions
in partnership with OMNeT++ Team

The beginnings

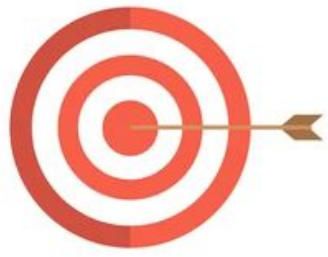


Problem statement

Main goal:

Improve speed of current wifi error models

while maintaining the baseline accuracy.

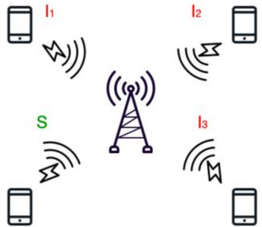


SNIRs, Error model, PER

SNIR (input): Commonly used in wireless communication as a way to measure the quality of wireless connections

Error model (what we enhancing): Describes how the SNIR affects the amount of errors at the receiver.

Packet Error Rate (output): is used to test the performance of a receiver. PER is the ratio, in percent, of the number of Test Packets not successfully received by the receiver.



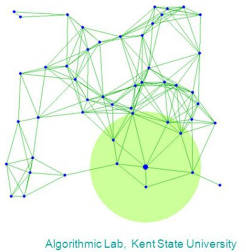
Problem statement

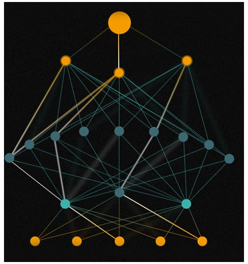
Models we going to mention:

- **Scalar Radio Model**
    - Analytical formulas
    - Based on single SNIR values
    - fast but still inaccurate

- **Layered Radio Model (baseline)**
    - Based on the whole SNIR pattern
    - Slow but can be trusted as a baseline measurement in this study (lack of empirical data)

- **Neural network approach**
    - Can be quite accurate.
    - Can achieve very fast inference speeds.
    - Can generalize for different wifi modes.
    - Can take into account the whole SNIR pattern by design



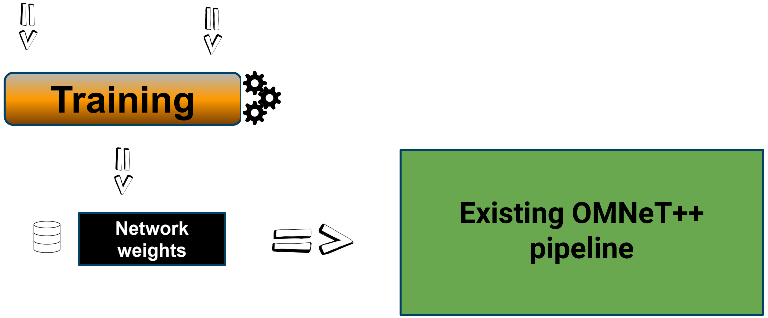


Problem statement

Trade-off:

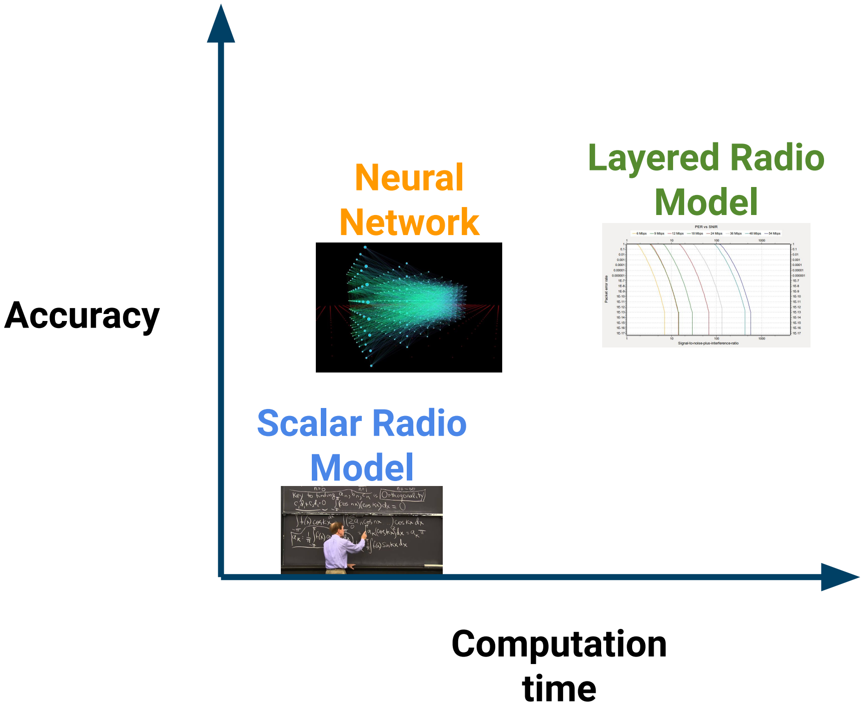
- Much faster computation compared to layered model

- Cannot be more accurate then layered model (only converge to it)



Based on OMNET++ presentation:
https://summit.omnetpp.org/archive/2020/assets/pdf/neural-errormodel-prez.pdf

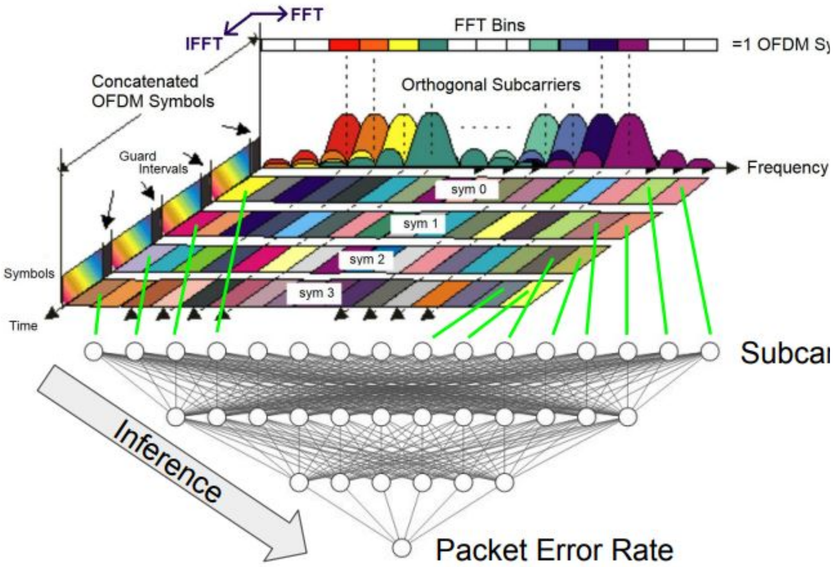
Motivation: trade-off



Exploring the data

	packetError	timeDivision	frequencyDivision	SNIR_0	SNIR_1	SNIR_2	SNIR_3	SNIR_4	SNIR_5	SNIR_6	...
0	0.42	8	52	45.39570	35.49330	40.84400	39.25960	37.55480	29.89270	41.27730	...
1	0.00	8	52	92.51780	92.20220	49.82480	50.40600	42.85030	40.73410	55.60380	...
2	0.01	8	52	76.55170	66.73260	37.70400	46.79620	82.01770	67.10550	50.40940	...
3	1.00	8	52	5.20857	4.90663	5.44037	6.72079	5.17013	5.50500	6.15690	...
4	0.00	8	52	58.82130	59.34190	96.12950	82.14270	48.66300	53.52890	45.76900	...
...	...	...	...	...	...	...	...	...	...	...	...
9995	1.00	8	52	3.69903	3.27762	2.76811	3.39221	6.55110	6.06512	7.43496	...
9996	0.17	8	52	46.08240	52.59990	59.25980	61.08370	71.71310	68.08100	44.40600	...
9997	0.49	8	52	42.20150	50.46780	54.03000	47.05220	43.58100	42.52460	53.61060	...
9998	0.71	8	52	66.20530	90.01320	76.54620	72.24160	61.60140	78.36080	145.99600	...
9999	0.76	8	52	11.31480	6.43079	6.38891	9.46025	8.52136	7.24323	6.70294	...

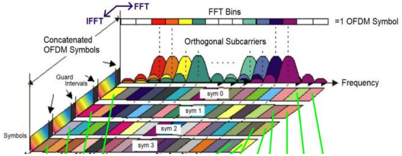
10000 rows × 419 columns

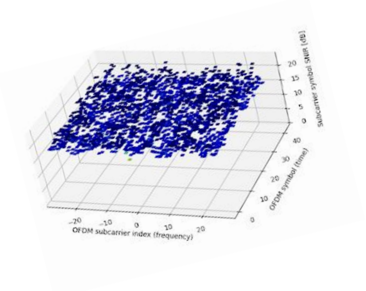


Raw data: SNIR values corresponding to a packetError value.

Borrowed from OMNET++ presentation:
https://summit.omnetpp.org/archive/2020/assets/pdf/neural-errormodel-prez.pdf

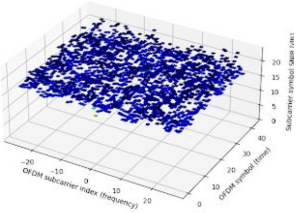
Exploring the data



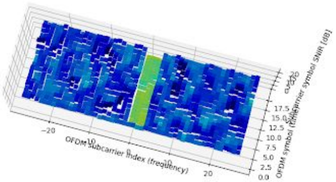


Conjecture: if the humans can recognize relationships between packet error values and the SNIR patterns, there is a great chance that neural network can do also and even doing better.

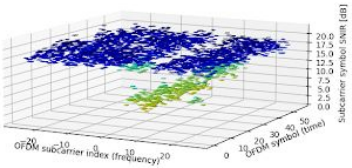
PER: 0



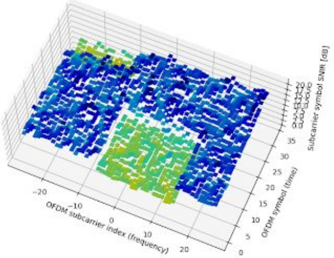
PER: 0.01



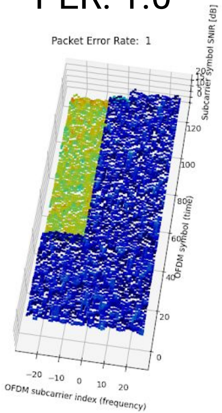
PER: 0.1



PER: 0.33



PER: 1.0



Increasing packet error values

Exploring the data

Conjecture: if humans can recognize relationships between packet error values and the SNIR patterns, there is a great chance that neural network can do also and even doing better.

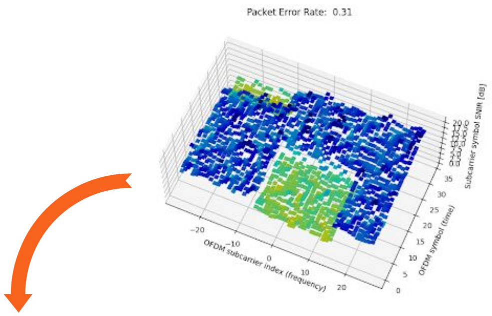
Try yourself!

colab-notebook demo

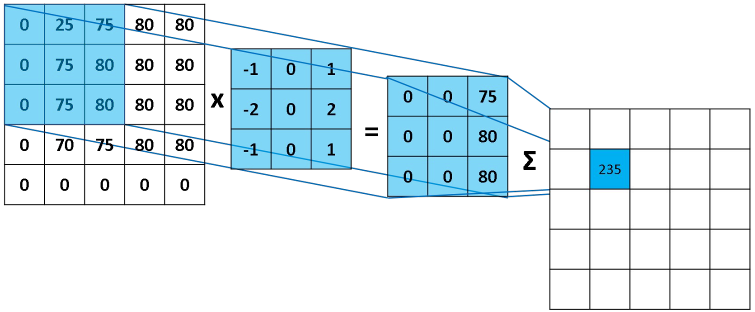
Interactive 3D clustering demo:

https://skfb.ly/ozB6B

Methods we tried



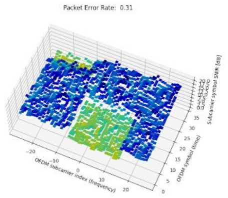
Convolutional Neural Networks



expectations:
- compatible with different input sizes
- excellent in image like pattern recognition
- **limitation: limited long range interactions in the data**

reality:
- in this particular case it worked but it was very hard to achieve good accuracy
- it was not as robust as we thought for architectural changes
- it was hard to tune and train it with this data
- it generalized poorly

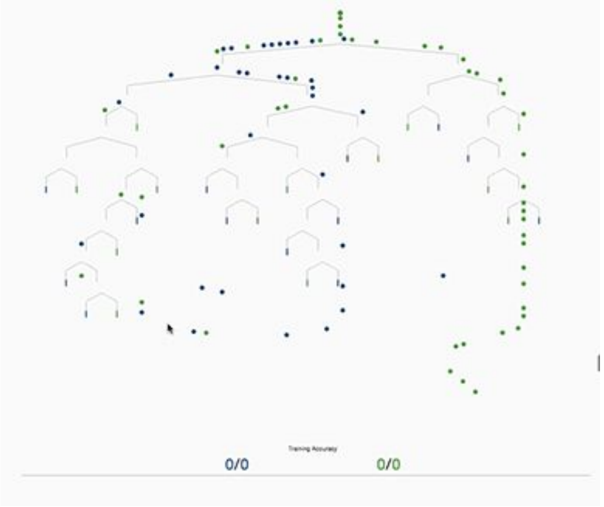
Methods we tried



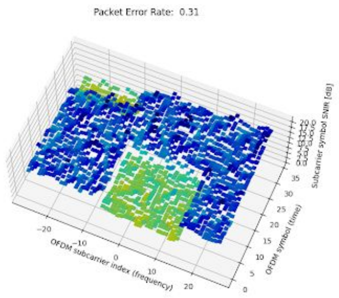
XGBOOST

expectations:
- SOTA in tabular data inference
- fast and generalizes exceptionally
- **limitation: input can be fixed size data only**

reality:
-for fixed size SNIR matrices it achieved very good accuracy and generalized well, but we needed it to work for variable SNIR sizes, so we just could not use it.



Methods we tried



LSTM

expectations:
- recognizing long range interaction in data
- excellent for time dependent data
- excellent for variable data size
- limitation: longer training times due to more sequential structure

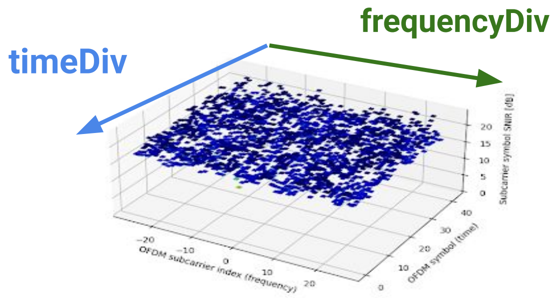
reality: it satisfied all of our expectations

LSTM network and settings which worked

Model configuration and topology:

Lightweight, and fast

Input:

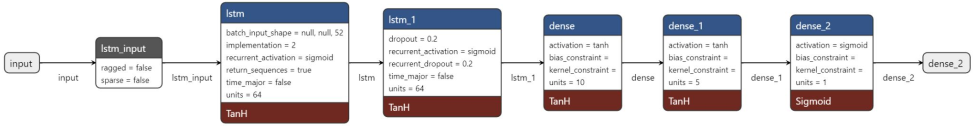


input size:

timeDivision X frequencyDivision (fixed)
(it was fixed in the data as well…)

batched input -> batch size as hyperparameter

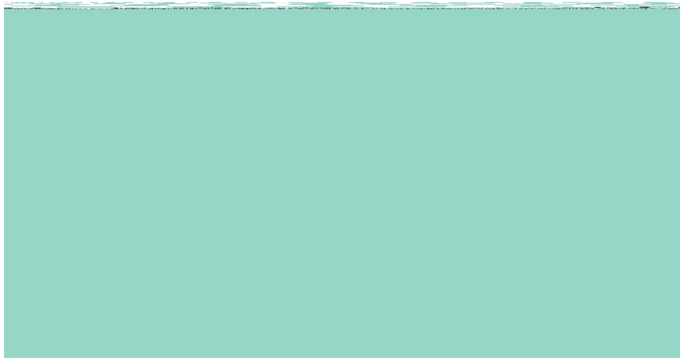
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None,None, 64)           29952

 lstm_1 (LSTM)               (None, 64)                33024

 dense (Dense)               (None, 10)                650

 dense_1 (Dense)             (None, 5)                 55

 dense_2 (Dense)             (None, 1)                 6

=================================================================
Total params: 63,687
Trainable params: 63,687
Non-trainable params: 0
_________________________________________________________________
```



LSTM network and settings which worked

Conjecture:

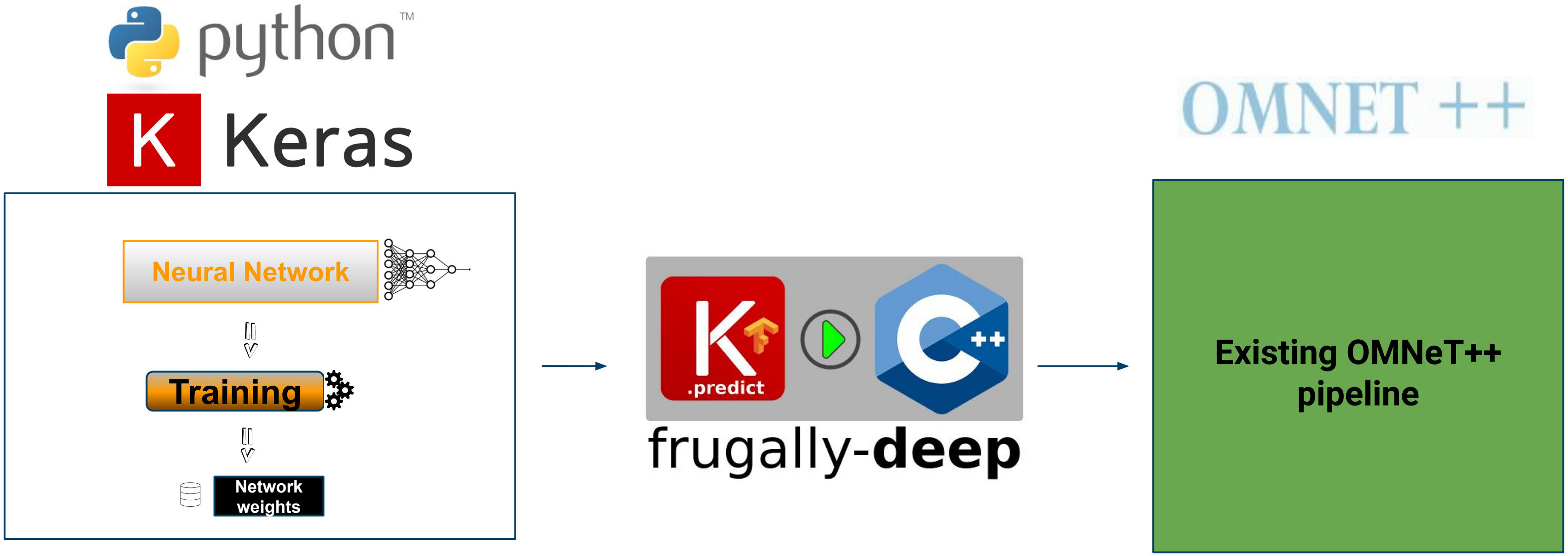
LSTM-s are designed for recognizing patterns which are "far from each other", e.g. time series data points distant from each other with respect to time, SNIR values can be distant in frequency…



Example for our conjecture in the study of long-range amino acid interaction inference:
https://academic.oup.com/bioinformatics/article/33/18/2842/3738544

CNN locality by design:
https://abenezer-g.medium.com/part-1-convolutional-neural-network-in-a-nutshell-89f81a329ec3 **-> "*The reason why CNN is best at image classification"* section**

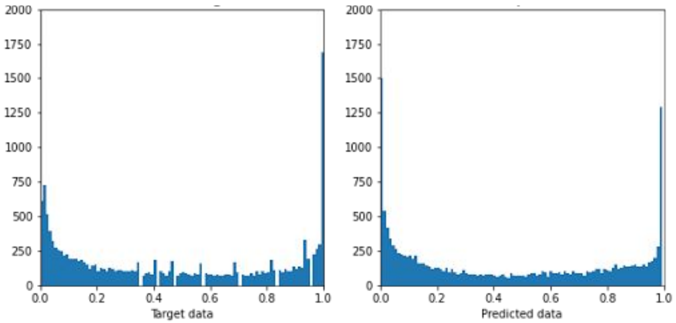
Neural Network integration to existing pipeline



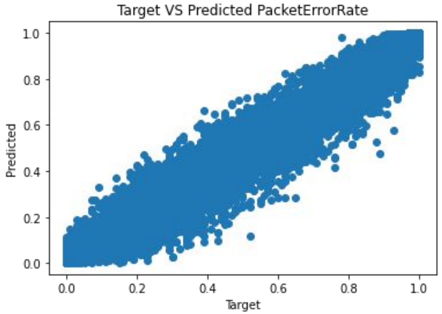
Results

Training on 160.000 lines,

- Split the data. 80% training, 20% testing.
- We place the data into batches with identical timeDivision.
- We transform the data to have a list with shapes of **batch_size x timeDivision x frequencyDivision**.



Target and predicted value distributions are very similar.



Predicted values plotted in the function of target values.

Predicted vs Target values:

RMSE: 0.059539
Corr coeff: 0.9893

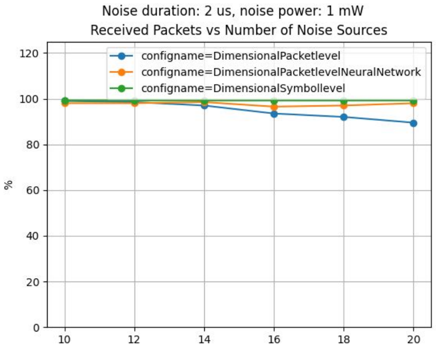
Results

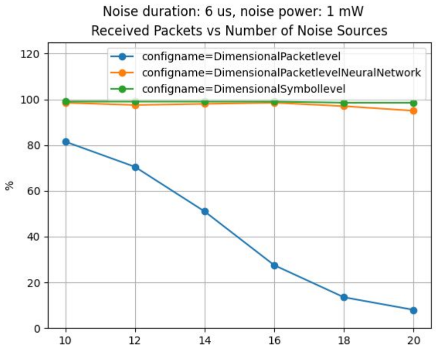
The closer the points to the green ones, “the better”, due to the lack of experimental data….

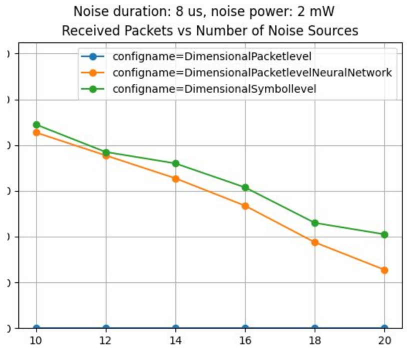
Scalar Radio model

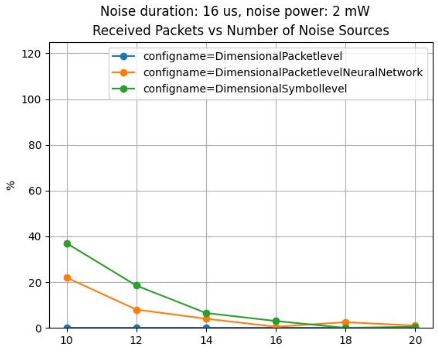
Neural Network

Layered Radio Model









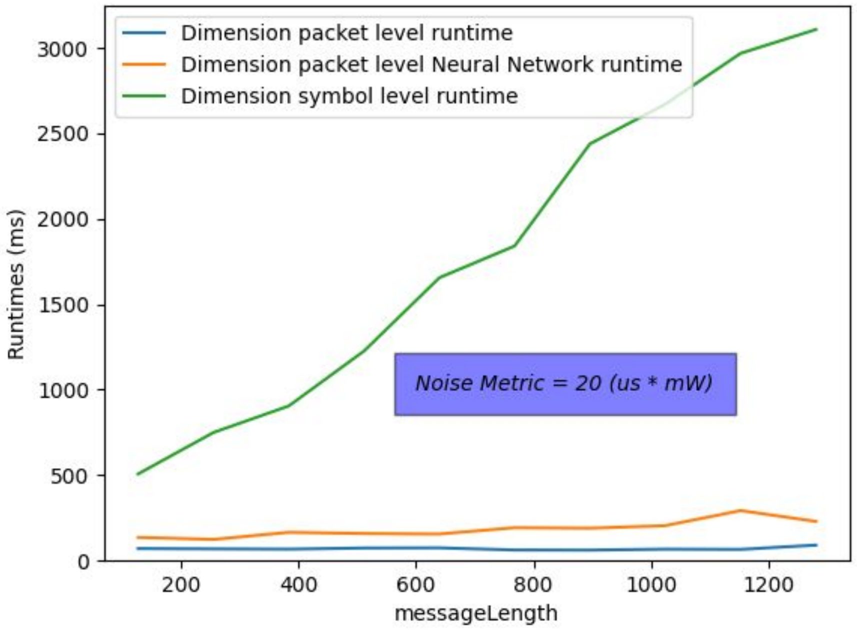
Increasing noise (Noise duration x Noise Power [us x mW])

Results

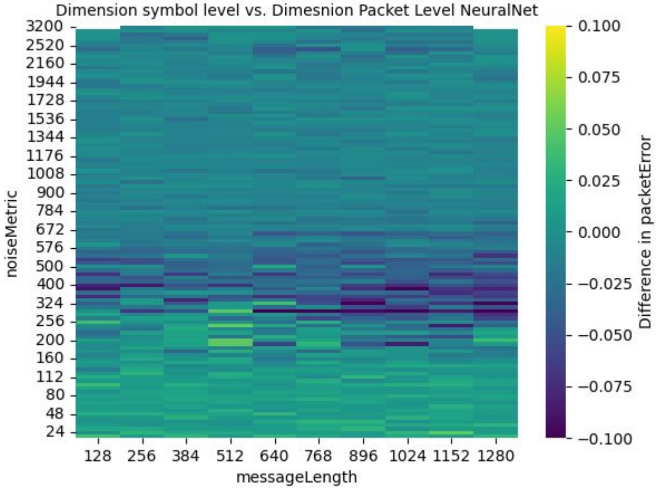
Comparing the

Scalar Radio Model (blue) and
Neural Network approach (orange) to
the Layered Radio Model (green).

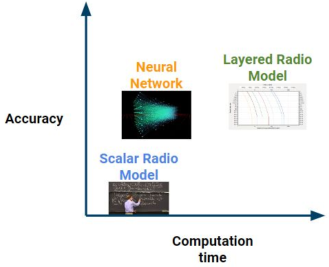
Comparison with respect of computational time.

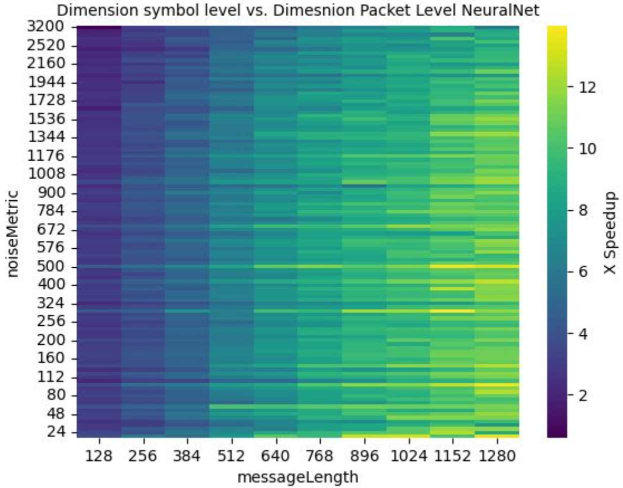


Results: trade-off is fulfilled



Neural network approach is in agreement with the simulated baseline





Neural Network approach can be orders of magnitude faster.

Conclusion

The novel neural network approach in agreement with the Layered Radio model baseline, while it has a significant speed up in computation time, especially when the packet length is large.

With the frugally-deep environment, our Keras-python implementation is compatible to the existing OMNET++ ecosystem, making this project potentially valuable for future studies and relevant use cases.

In the future we are planning to continue the collaboration and expand our model to generalize to multiple WIFI modes as for now it is only compatible with a single specific mode.

Tools used

python

Keras

Creating the architecture, train, inference

frugally-**deep**

Omnet++ team integrated our model with it to their pipeline

pandas

matplotlib

jupyter

Data visualization, showcasing, data management.

Thank you

Check out our other projects: www.chimeramultimedia.com

Follow us on Twitter, Instagram, LinkedIn, YouTube, join our Discord:

www.chimeramultimedia.com/contact

Chimera
Multimedia & Solutions

OMNET ++