Sionna: An Open-Source Library for Next-Generation Physical Layer Research

Jakob Hoydis, Sebastian Cammerer, Fayçal Aït Aoudia, Avinash Vem,
Nikolaus Binder, Guillermo Marcus, Alexander Keller

***Abstract*—Sionna™ is a GPU-accelerated open-source library for link-level simulations based on TensorFlow. It enables the rapid prototyping of complex communication system architectures and provides native support for the integration of neural networks. Sionna implements a wide breadth of carefully tested state-of-the-art algorithms that can be used for benchmarking and end-to-end performance evaluation. This allows researchers to focus on their research, making it more impactful and reproducible, while saving time implementing components outside their area of expertise. This white paper provides a brief introduction to Sionna, explains its design principles and features, as well as future extensions, such as integrated ray tracing and custom CUDA kernels. We believe that Sionna is a valuable tool for research on next-generation communication systems, such as 6G, and we welcome contributions from our community.**

I. Introduction

While 5G is deployed around the globe and the work areas for its future evolution called 5G Advanced (Release 18) have just been approved [1], researchers in academia and industry have already started to define visions and key technologies for 6G, e.g., [2]. One recurring vision in many papers is that of creating digital twin worlds to connect physical and biological entities and to enable new mixed-reality experiences such as the Metaverse [3].

Some of the most prominent 6G research topics are communication in the Terahertz band [4], cell-free, holographic, and very large aperture multiple-input multiple-output (MIMO) [5], unmanned aerial vehicle (UAV) and satellite communication [6], machine learning (ML)-based air interface design [7], semantic communication [8], reconfigurable intelligent surfaces (RIS) [9], sensing and localization [10], digital twins [11], computer vision for wireless [12], as well as federated learning over wireless [13]. Interestingly, research on most of the above topics requires one or more of the following:

- **Data from specific radio environments:** While the widely used stochastic channel models, e.g., [14], are well suited to capture the behavior of a certain class of environment, they cannot be used for any application that requires the simulation of a *specific* environment, e.g., the optimal configuration of a particular RIS. Radar-based sensing and localization as well as computer vision-aided applications only work if there is a spatially consistent correspondence between physical location and channel impulse response (or visual input). This can only be achieved through either ray tracing or extensive measurement campaigns.
- **Native integration of ML:** ML and, particularly, neural networks (NNs) are expected to play an increasingly important role for the implementation of transceiver algorithms and maybe even the 6G air interface design [1], [7], [15]. Research on such topics does not only benefit tremendously from a tight integration of ML, e.g., [16], [17], and link-level simulation tools, but also from automatic gradient computation through the entire system which allows for seamless integration of NNs.
- **Very high detail or scale:** Link-level simulations of 6G systems require unprecedented modeling accuracy and scale. While cell-free or very large aperture MIMO systems need the computation of a very large number of signal propagation paths, Terahertz communication systems have complex channel models accounting for nonlinear hardware impairments. Moreover, the full potential of ML-enhanced algorithms will only be unleashed with very realistic simulations that account for detail that has been ignored in the past due to prohibitive simulation complexity and for mathematical tractability.

Due to the reasons above, we firmly believe that the breakthroughs required for 6G can only be achieved if our community has access to new tools for physical layer research. That is why we have started to develop a new and open-source link-level simulator called Sionna.[1]

II. Motivation & Background

A. Why we decided to create Sionna

Sionna[2] is written by researchers for researchers in communications and aims at making our work more efficient:

- **Rapid prototyping:** Sionna provides a high-level Python application programming interface (API) to rapidly model complex communication systems from end-to-end while allowing one to easily adapt the parts a new research idea is about. Graphic processing unit (GPU) acceleration makes it super fast and enables interactive exploration, e.g., in Jupyter notebooks [18].
- **Benchmarking against the state of the art:** Sionna provides many carefully tested standard processing blocks and state-of-the-art algorithms that can be used for performance benchmarking. This reduces the time spent implementing auxiliary components that one's research is not about.
- **Realistic industry-grade evaluations:** Experts in one domain, say channel estimation, do not necessarily have

[1] https://github.com/NVlabs/sionna

[2] Sionna is a goddess in Irish mythology and the namesake of the River Shannon, see https://en.wikipedia.org/wiki/Shannon_(given_name).

the time, tools, or background to evaluate their algorithms for end-to-end performance, e.g, coded block error rate (BLER) of a 5G Polar code over a realistic 3GPP channel model. With Sionna, one can make such evaluations with a few additional lines of code. If needed, simulations can be scaled across large multi-GPU setups.

- **Native support of NNs:** Sionna enables seamless integration of NNs in the physical layer signal processing chain. As most building blocks are differentiable, gradients can be backpropagated through an entire system, which is the key enabler for end-to-end learning of new artificial intelligence (AI)-defined air interfaces. Sionna also eliminates the need for different tools for data generation, training, and evaluation.
- **Addressing 6G research needs:** Sionna will allow for the use of ray tracing, datasets, and generative models instead of stochastic channel models to enable research on many novel topics, such as joint communication and sensing, vision-aided wireless communications, intelligent reflecting surfaces, semantic communications, and digital twins. Also, a THz channel model is under development. Community contributions to components are welcome.
- **Reproducibility:** With Sionna, it is straight-forward to make research reproducible by others. We encourage all users to publish their Sionna-based code along with their research papers and contribute novel components so that they can be reused by others. This will also make it easier to compare algorithms under the same conditions.

B. Related open-source software (OSS)

Commercial software products, such as Matlab [19], play an important role for simulation-based research in our field. There is also a growing body of well-maintained OSS projects serving different purposes, such as channel simulation [20], [21], link-level simulations [22]–[24], system-level simulations [25], network simulations [26], [27], and hardware experimentation [28], [29]. There are also several open experimental platforms, such as the Platforms for Advanced Wireless Research (PAWR) in the US [30] and the OneLab Platforms [31] in Europe. These allow for reproducible large-scale experiments in controlled and/or realistic environments. Not all the mentioned OSS, e.g., [20], [21], [24], accept contributions from external parties though.

Sionna is a tool for link-level simulations with in-built channel simulator. It has hence some similarities with Quadriga [20] and NYUSIM [21] for the generation of channel impulse responses, as well as with the Vienna 5G Link-Level Simulator [22] and HermesPy [24] for physical layer processing. It differs, however, through its end-to-end differentiability, native support of NNs and GPUs, as well as ray tracing capabilities. Sionna can be scaled to large multi-GPU setups to simulate, e.g., realistic multi-cell Massive MIMO systems.

III. A Sionna Primer

Sionna is written in Python using TensorFlow [16] and the Keras API [17]. Exceptions are the custom CUDA kernels and ray tracing capabilities that will be described in Sec. IV.

A. Design principles

One of the key differences between Sionna and alternative link-level simulators is that all algorithms are implemented using high-dimensional Tensors, e.g., of shape

`[batch_size, num_tx,..., fft_size]`

where the first dimension is the *batch size*, representing different independent Monte Carlo trials that are executed in parallel. For-loops are generally avoided. This representation is borrowed from the field of deep learning and leads to "embarrassingly parallel" workloads that can be efficiently executed on GPUs. If no GPU is available, Sionna will also run on (multiple) central processing units (CPUs).

Another design principle of Sionna is that all components are implemented as independent Keras layers [17]. This has the advantages that (i) complex system models can be constructed by simply connecting the desired layers, (ii) components can be easily replaced by NNs, and (iii) gradients are automatically computed by TensorFlow [16] which is a key enabler for end-to-end learning of communication systems. Both Keras' sequential and functional APIs can be used.

Sionna's default data type is `tf.complex64`, i.e., 32bit-precision for each complex dimension. For certain applications, e.g., optical communications, the precision of all or a few selected layers can be increased to `tf.complex128`. Note that this also doubles their memory requirements.

Sionna supports both of TensorFlow's execution modes: eager and graph execution. The former is very useful for interactive development of new components, while the latter provides improved memory efficiency and reduced runtime. In addition, all of Sionna's layers (apart from a few exceptions) support just-in-time compilation using XLA (accelerated linear algebra) [32] for further speed-ups.

B. Features

The first public release of Sionna (v0.8.0) implements the following list of features:

Forward error correction (FEC):

- 5G LDPC codes including rate matching [33]
- 5G Polar codes including rate matching [33]
- Cyclic redundancy check (CRC) [33]
- Reed-Muller & Convolutional codes
- Interleaving & Scrambling
- Belief propagation (BP) decoder and variants
- SC, SCL, and SCL-CRC Polar decoders
- Viterbi decoder
- Demapper with prior
- EXIT chart simulations
- Import of partity-check matrices in *alist* format

Channel models:

- Additive white Gaussian noise (AWGN) channel
- Flat-fading channel models with antenna correlation
- 3GPP 38.901 TDL, CDL, UMa, UMi, RMa models [14]
- Import of channel impulse response from datasets
- Channel output computed in time or frequency domain

```
batch_size = 1024
n = 1000 # codeword length
k = 500  # information bits per codeword
m = 4 # number of bits per symbol
snr = 10
c = Constellation("qam", m)
b = BinarySource()([batch_size, k])
u = LDPC5GEncoder(k, n)(b)
x = Mapper(constellation=c)(u)
y = AWGN()([x, 1/snr])
llr = Demapper("app", c)([y, 1/snr])
b_hat = LDPC5GDecoder(LDPC5GEncoder(k, n))(llr)
compute_ber(b, b_hat)
```

Listing 1. Sionna "Hello, World!" example.

MIMO processing:

- Multiuser & multicell MIMO support
- 3GPP 38.901 & custom antenna arrays/patterns
- Zero forcing (ZF) precoding
- Minimum mean squared error (MMSE) equalization

Orthogonal frequency-division multiplexing (OFDM):

- OFDM modulation & demodulation
- Cyclic prefix insertion & removal
- Flexible 5G slot-like frame structure
- Arbitrary pilot patterns
- LS channel estimation & Nearest neighbor interpolation

Despite the fact that Sionna supports 5G-compliant channel codes and channel models, it does not try by any means to be a 5G-compliant link-level simulator. It rather allows researchers to test their algorithms under realistic conditions and to compare to state-of-the-art algorithms that are widely accepted in the industry, while having a maximum degree of flexibility and freedom. The current set of features should be seen as a good starting point that one can build on to rapidly prototype and evaluate new ideas.

C. Hello, World!

The Sionna webpage provides an extensive set of tutorial notebooks (https://nvlabs.github.io/sionna/tutorials.html) that cover in detail the various ways the library can be used. In order to get a first impression of how developing with Sionna feels like, we provide and discuss a few short code examples.

Listing 1 shows a Sionna "Hello, World!" example in which the transmission of `batch_size` LDPC codewords over an AWGN channel using 16QAM modulation is simulated. This example shows how Sionna layers are instantiated and then immediately applied to a previously defined tensor. For example, in line 8, the `LDPC5GEncoder` is instantiated with the parameters `n` and `k` and then applied to the tensor `b` which contains randomly generated information bits. This coding style follows the functional API of Keras.

In Listing 2, we have made the `Constellation` *trainable* and replaced the `Demapper` by a `NeuralDemapper` which is defined in Listing 3. What happens under the hood is that the tensor defining the constellation points has become a trainable TensorFlow Variable. It can now be tracked together with the weights of the NN of the `NeuralDemapper` by TensorFlow's automatic differentiation feature, see [34]. One could then define an adequate `loss` function (line 7), such as the total binary cross-entropy, and compute the gradient of this loss with respect to all trainable variables. This gradient may then be used to optimize jointly the constellation points and the `NeuralDemapper` through stochastic gradient descent. This concept is sometimes referred to as *end-to-end learning* [35] and can be applied to very complex system models.

```
c = Constellation("qam", m, trainable=True)
b = BinarySource()([batch_size, k])
u = LDPC5GEncoder(k, n)(b)
x = Mapper(constellation=c)(u)
y = AWGN()([x, 1/snr])
llr = NeuralDemapper()([y, 1/snr])
loss = BinaryCrossentropy(from_logits=True)(u, llr)
```

Listing 2. Components can be easily replaced by NNs or made trainable.

```
class NeuralDemapper(Layer):
    def build(self, input_shape):
        # Initialize the neural network layers
        self._dense1 = Dense(16, activation="relu")
        self._dense2 = Dense(m)

    def call(self, inputs):
        y, no = inputs

        # Stack noise variance, real and imaginary
        # parts of each symbol. The input to the
        # neural network is [Re(y_i), Im(y_i), no].
        no = no*tf.ones(tf.shape(y))
        llr = tf.stack([tf.math.real(y),
                        tf.math.imag(y),
                        no], axis=-1)

        # Compute neural network output
        llr = self._dense1(llr)
        llr = self._dense2(llr)

        # Reshape to [batch_size, n]
        llr = tf.reshape(llr, [batch_size, -1])
        return llr
```

Listing 3. Definition of a simple neural demapper in Keras.

Listing 3 shows the definition of a `NeuralDemapper` as a Keras layer which was used in Listing 2 to replace a traditional `Demapper`. This simple combination of NN components and traditional algorithms together with automatic gradient computation is one of the key advantages of Sionna for ML-based research.

D. Limitations

As Sionna parallelizes simulations over batches, the available GPU memory can quickly become the bottleneck. This can be circumvented by either reducing the batch size or distributing the model or batch across multiple GPUs.

Algorithms with complex conditional logic, e.g., where examples in a batch can be treated differently, do not lend themselves to be written efficiently using TensorFlow's Python API. In this case, one can resort to implementing custom TensorFlow operations in C++ and CUDA (see Sec. IV-A). At the cost of execution speed, it is also possible to write such algorithms in native Python code and wrap them as a TensorFlow operations.

```python
import SionnaRTX
rt = SionnaRTX.RayTracer()
rt.scene.load("../scenes/voyager.obj")
# Render an image of the loaded scene
rt.render(num_samples = 8)
# Render a coverage map on top of the scene
rt.render_lossmap()
# Compute impulse responses of all TX/RX pairs
a, tau = rt.compute_ir()
```

Listing 4. Sionna ray tracing example.

E. Contributing

Sionna is released under the Apache 2.0 license. We have chosen this license so that users do not need to worry about possible patent infringements and litigation. We accept and welcome contributions from external parties through *pull request* on the Sionna GitHub repository. The only requirement is that all contributors "sign-off" their commits with the Developer Certificate of Origin (DCO) [36].

IV. Future Extensions

A. Custom CUDA kernels

We have chosen Python and TensorFlow for writing most of Sionna due to their popularity and simplicity. However, not all signal processing algorithms and simulation routines can be easily expressed using tensors and compositions of existing TensorFlow operations. This is generally the case, whenever an algorithm might execute differently for each example in a batch, e.g., due to `if-else` statements, or requires complex indexing. Examples comprise the Polar SCL and min-sum BP decoder, components of the 3GPP channel models, as well as the convolution with a time-varying channel impulse response. Moreover, some algorithms expressed as the composition of TensorFlow operations might not lead to acceptable performance or memory requirements.

As an alternative, one can implement such algorithms without the constraints of the TensorFlow API as *custom* TensorFlow operations [37] using C++ and XLA [32] or CUDA [38] to benefit from GPU acceleration. We found the resulting code to be significantly more convenient, better readable, and less prone to errors. The source code of a custom operation is compiled into a binary *shared object* that can be loaded by TensorFlow from the Python code. The custom operation can then be called from the Python TensorFlow API in the same way as any other operation. Although tensors are used as inputs and outputs to custom operations, any type of data structure can be used in the internal C++ implementation with CUDA. Moreover, the CUDA-X libraries [39] can be leveraged to further benefit from efficient GPU-accelerated algorithms. If required, gradients can be specified to make a custom operation differentiable.

B. Ray tracing

As mentioned earlier, many 6G topics, such as RIS or integrated sensing and communications, require the simulation of a specific radio environment in a physically based manner which cannot be done with stochastic channel models. For some applications, also the visual representation of a scene is required, see, e.g., [40]. To address these needs, SionnaRTX will soon bring both of these capabilities to Sionna so that one can render a scene, get the desired channel impulse responses, and use them immediately for link-level simulations or other applications.

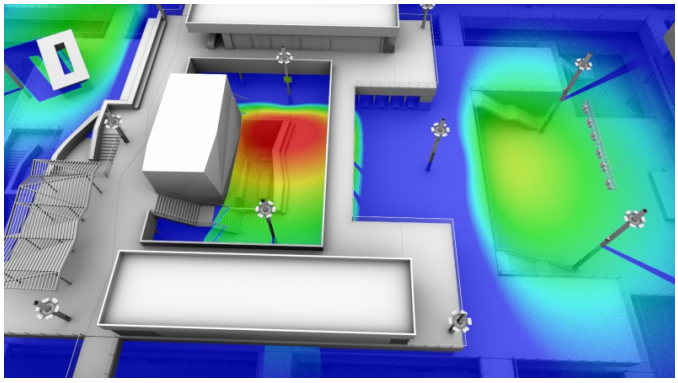

Fig. 1. Example of a loss map rendered on top of a scene geometry using radiation transport simulation by `SionnaRTX`.

As the name says, SionnaRTX embraces the RTX™ architecture[3] for hardware ray tracing and supports the material definition language (MDL)[4] [41], which is the industry standard for sharing definitions of physically based materials.

Listing 4 demonstrates the simplicity of invoking high performance ray tracing from within a Jupyter notebook. First, a scene is loaded from a scene file. A wide range of formats is supported.[5] Then, an image of the scene is rendered, and a coverage map is displayed on top of it (see Fig. 1). Finally, channel impulse responses between all defined transmitters and receivers are computed. These can be directly used for link-level simulations instead of a stochastic channel model. Beyond that simplicity, one can manipulate scene parameters from within Python and specify materials with MDL.

V. Conclusions

The next decade will be marked by groundbreaking research on disruptive technologies for 6G whose exploration poses various challenges for computer simulations. For this reason, we have started the development of Sionna, a differentiable open-source link-level simulator with native integration of NNs and full GPU acceleration. Soon, Sionna will have an integrated ray tracer for scene rendering and wave propagation. We are excited about the possibilities Sionna offers and hope for wide adoption and numerous contributions from our community.

References

[1] X. Lin, "An Overview of 5G Advanced Evolution in 3GPP Release 18," *arXiv preprint arXiv:2201.01358*, Jan. 2022.

[2] M. A. Uusitalo, M. Ericson, B. Richerzhagen, E. U. Soykan, P. Rugeland, G. Fettweis, D. Sabella, G. Wikström, M. Boldi, M.-H. Hamon *et al.*, "Hexa-X the European 6G flagship project," in *Proc. Joint Europ. Conf. Netw. Commun. & 6G Summit (EuCNC/6G Summit)*, Porto, Portugal, Jun. 2021, pp. 580–585.

[3] https://developer.nvidia.com/rtx

[4] https://www.nvidia.com/en-us/design-visualization/technologies/material-definition-language/

[5] http://assimp.sourceforge.net/main_features_formats.html

[3] H. Viswanathan and P. E. Mogensen, "Communications in the 6G Era," *IEEE Access*, vol. 8, pp. 57 063–57 074, Mar. 2020.

[4] I. F. Akyildiz, C. Han, Z. Hu, S. Nie, and J. M. Jornet, "TeraHertz Band Communication: An Old Problem Revisited and Research Directions for the Next Decade," *arXiv preprint arXiv:2112.13187*, Dec. 2021.

[5] E. Björnson, L. Sanguinetti, H. Wymeersch, J. Hoydis, and T. L. Marzetta, "Massive MIMO is a reality—What is next?: Five promising research directions for antenna arrays," *Digital Signal Processing*, vol. 94, pp. 3–20, Nov. 2019, special Issue on Source Localization in Massive MIMO. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200419300776

[6] G. Geraci, A. Garcia-Rodriguez, M. M. Azari, A. Lozano, M. Mezzavilla, S. Chatzinotas, Y. Chen, S. Rangan, and M. Di Renzo, "What Will the Future of UAV Cellular Communications Be? A Flight from 5G to 6G," *arXiv preprint arXiv:2105.04842*, May 2021.

[7] J. Hoydis, F. A. Aoudia, A. Valcarce, and H. Viswanathan, "Toward a 6G AI-Native Air Interface," *IEEE Commun. Mag.*, vol. 59, no. 5, pp. 76–81, May 2021.

[8] M. Kountouris and N. Pappas, "Semantics-Empowered Communication for Networked Intelligent Systems," *IEEE Commun. Mag.*, vol. 59, no. 6, pp. 96–102, Jun. 2021.

[9] M. D. Renzo, M. Debbah, D.-T. Phan-Huy, A. Zappone, M.-S. Alouini, C. Yuen, V. Sciancalepore, G. C. Alexandropoulos, J. Hoydis, H. Gacanin, J. d. Rosny, A. Bounceur, G. Lerosey, and M. Fink, "Smart radio environments empowered by reconfigurable AI meta-surfaces: An idea whose time has come," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, p. 129, May 2019.

[10] A. Bourdoux, A. N. Barreto, B. van Liempd, C. de Lima, D. Dardari, D. Belot, E.-S. Lohan, G. Seco-Granados, H. Sarieddeen, H. Wymeersch, J. Suutala, J. Saloranta, M. Guillaud, M. Isomursu, M. Valkama, M. R. K. Aziz, R. Berkvens, T. Sanguanpuak, T. Svensson, and Y. Miao, "6G White Paper on Localization and Sensing," *University of Oulu: 6G Research Visions*, Jun. 2020.

[11] P. Almasan, M. Ferriol-Galmés, J. Paillisse, J. Suárez-Varela, D. Perino, D. López, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong *et al.*, "Digital Twin Network: Opportunities and Challenges," *arXiv preprint arXiv:2201.01144*, Jan. 2022.

[12] T. Nishio, Y. Koda, J. Park, M. Bennis, and K. Doppler, "When wireless communications meet computer vision in beyond 5G," *IEEE Commun. Standards Mag.*, vol. 5, no. 2, pp. 76–83, Jun. 2021.

[13] Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, and D. Niyato, "Federated learning for 6G communications: Challenges, methods, and future directions," *China Communications*, vol. 17, no. 9, pp. 105–118, Sep. 2020.

[14] ETSI, "ETSI TR 138 901 V16.1.0: Study on channel model for frequencies from 0.5 to 100 GHz," ETSI, Tech. Rep., Nov. 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_tr/138900_138999/138901/16.01.00_60/tr_138901v160100p.pdf

[15] S. Ali, W. Saad, and D. Steinbach, Eds., *White Paper on Machine Learning in 6G Wireless Communication Networks*. University of Oulu, 2020, no. 7. [Online]. Available: http://urn.fi/urn:isbn:9789526226736

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, accessed Jan. 2022. [Online]. Available: https://www.tensorflow.org/

[17] F. Chollet *et al.*, "Keras," https://keras.io, 2015, accessed Jan. 2022.

[18] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter Notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87–90. [Online]. Available: https://jupyter.org

[19] T. M. Inc., "Matlab," https://www.mathworks.com/products/matlab.html, accessed Jan. 2022.

[20] S. Jaeckel, L. Raschkowski, K. Börner, and L. Thiele, "QuaDRiGa: A 3-D multi-cell channel model with time evolution for enabling

[21] S. Sun, G. R. MacCartney, and T. S. Rappaport, "A novel millimeter-wave channel simulator and applications for 5G wireless virtual field trials," *IEEE Trans. Antennas Propag.*, vol. 62, no. 6, pp. 3242–3256, 2014, accessed Jan. 2022. [Online]. Available: https://wireless.engineering.nyu.edu/nyusim/

communications," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Paris, France, May 2017, accessed Jan. 2022. [Online]. Available: https://quadriga-channel-model.de/

[22] S. Pratschner, B. Tahir, L. Marijanovic, M. Mussbah, K. Kirev, R. Nissel, S. Schwarz, and M. Rupp, "Versatile mobile communications simulation: the Vienna 5G Link Level Simulator," *EURASIP J. Wireless Commun. Netw.*, vol. 2018, no. 1, p. 226, Sep. 2018, accessed Jan. 2022. [Online]. Available: https://www.nt.tuwien.ac.at/research/mobile-communications/vccs/vienna-5g-simulators/

[23] A. Cassagne, O. Hartmann, M. Léonardon, K. He, C. Leroux, R. Tajan, O. Aumage, D. Barthou, T. Tonnellier, V. Pignoly, B. Le Gal, and C. Jégo, "AFF3CT: A Fast Forward Error Correction Toolbox!" *Elsevier SoftwareX*, vol. 10, p. 100345, Oct. 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352711019300457

[24] Barkhausen-Institut, "HermesPy," https://github.com/Barkhausen-Institut/hermespy, 2022, accessed Jan. 2022.

[25] M. K. Müller, F. Ademaj, T. Dittrich, A. Fastenbauer, B. R. Elbal, A. Nabavi, L. Nagel, S. Schwarz, and M. Rupp, "Flexible multi-node simulation of cellular mobile communications: the Vienna 5G System Level Simulator," *EURASIP J. Wireless Commun. Netw.*, vol. 2018, no. 1, p. 17, Sep. 2018, accessed Jan. 2022. [Online]. Available: https://www.nt.tuwien.ac.at/research/mobile-communications/vccs/vienna-5g-simulators/

[26] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–34, accessed Jan. 2022. [Online]. Available: https://www.nsnam.org

[27] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proc. SIMUTools*, Marseille, France, Mar. 2008, pp. 1–10, accessed Jan. 2022. [Online]. Available: https://omnetpp.org

[28] GNU Radio Project, "GNU Radio – the Free and Open Software Radio Ecosystem," https://www.gnuradio.org, accessed Jan. 2022.

[29] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A flexible platform for 5G research," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 33–38, Oct. 2014, accessed Jan. 2022. [Online]. Available: https://openairinterface.org

[30] "Platforms for Advanced Wireless Research (PAWR)," accessed Jan. 2022. [Online]. Available: https://advancedwireless.org

[31] "OneLab - Future Internet Testbeds," accessed Jan. 2022. [Online]. Available: http://www.onelab.eu

[32] "XLA: Optimizing Compiler for Machine Learning," accessed Jan. 2022. [Online]. Available: https://www.tensorflow.org/xla

[33] ETSI, "ETSI TS 138 212 V16.2.0: Multiplexing and channel coding," ETSI, Tech. Rep., Jul. 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138200_138299/138212/16.02.00_60/ts_138212v160200p.pdf

[34] "Introduction to gradients and automatic differentiation," accessed Jan. 2022. [Online]. Available: https://www.tensorflow.org/guide/autodiff

[35] T. O'shea and J. Hoydis, "An introduction to deep learning for the physical layer," *IEEE Trans. on Cogn. Commun. Netw.*, vol. 3, no. 4, pp. 563–575, Dec. 2017.

[36] "Developer Certificate of Origin Version 1.1," accessed Jan. 2022. [Online]. Available: https://developercertificate.org

[37] "Create an op," accessed Jan. 2022. [Online]. Available: https://www.tensorflow.org/guide/create_op

[38] "CUDA Zone," accessed Jan. 2022. [Online]. Available: https://developer.nvidia.com/cuda-zone

[39] "NVIDIA CUDA-X: GPU-Accelerated Libraries," accessed Jan. 2022. [Online]. Available: https://developer.nvidia.com/gpu-accelerated-libraries

[40] M. Alrabeiah, A. Hredzak, Z. Liu, and A. Alkhateeb, "ViWi: A Deep Learning Dataset Framework for Vision-Aided Wireless Communications," in *IEEE Proc. Vehicular Technology Conf. (VTC2020-Spring)*, May 2020, pp. 1–5.

[41] L. Kettner, M. Raab, D. Seibert, J. Jordan, and A. Keller, "The Material Definition Language," in *Workshop on Material Appearance Modeling*, R. Klein and H. Rushmeier, Eds. The Eurographics Association, 2015.