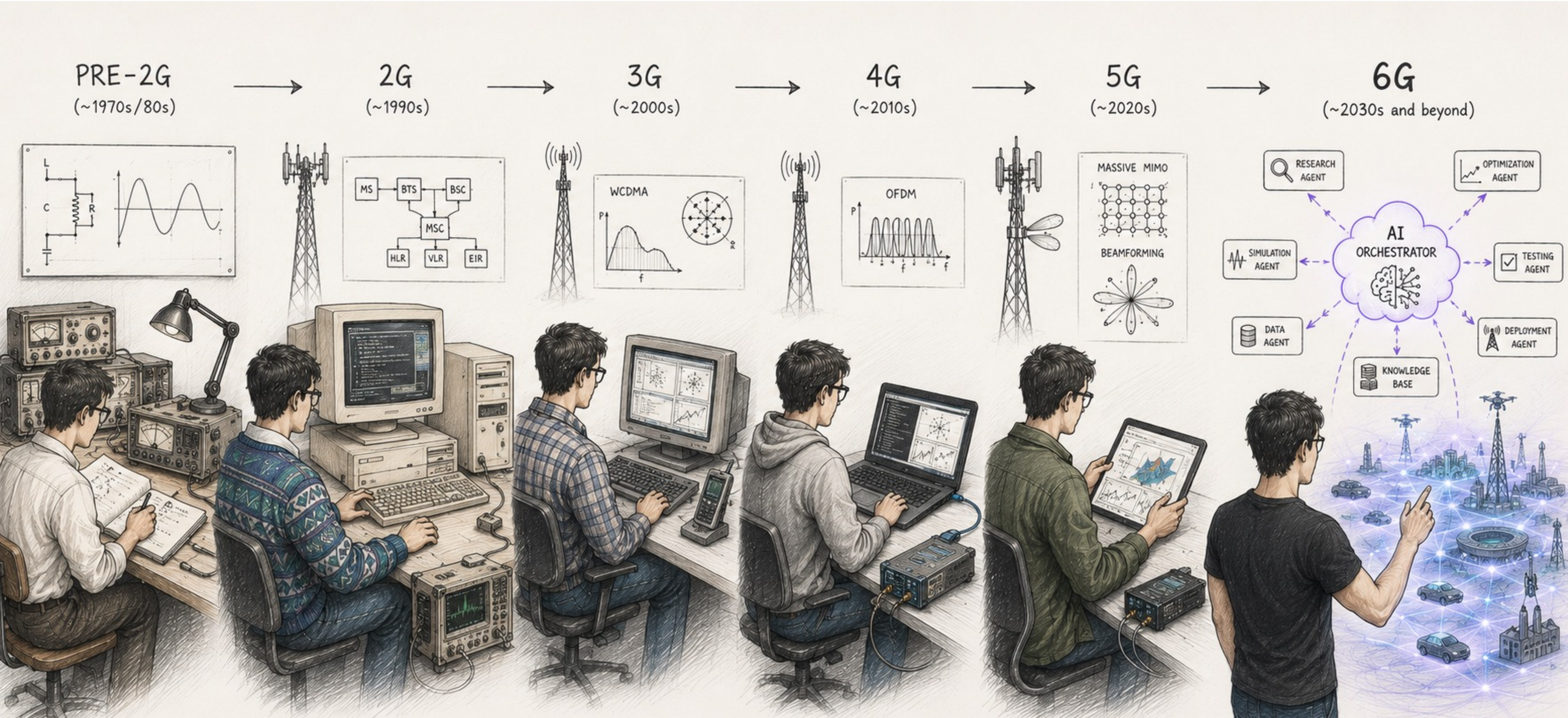
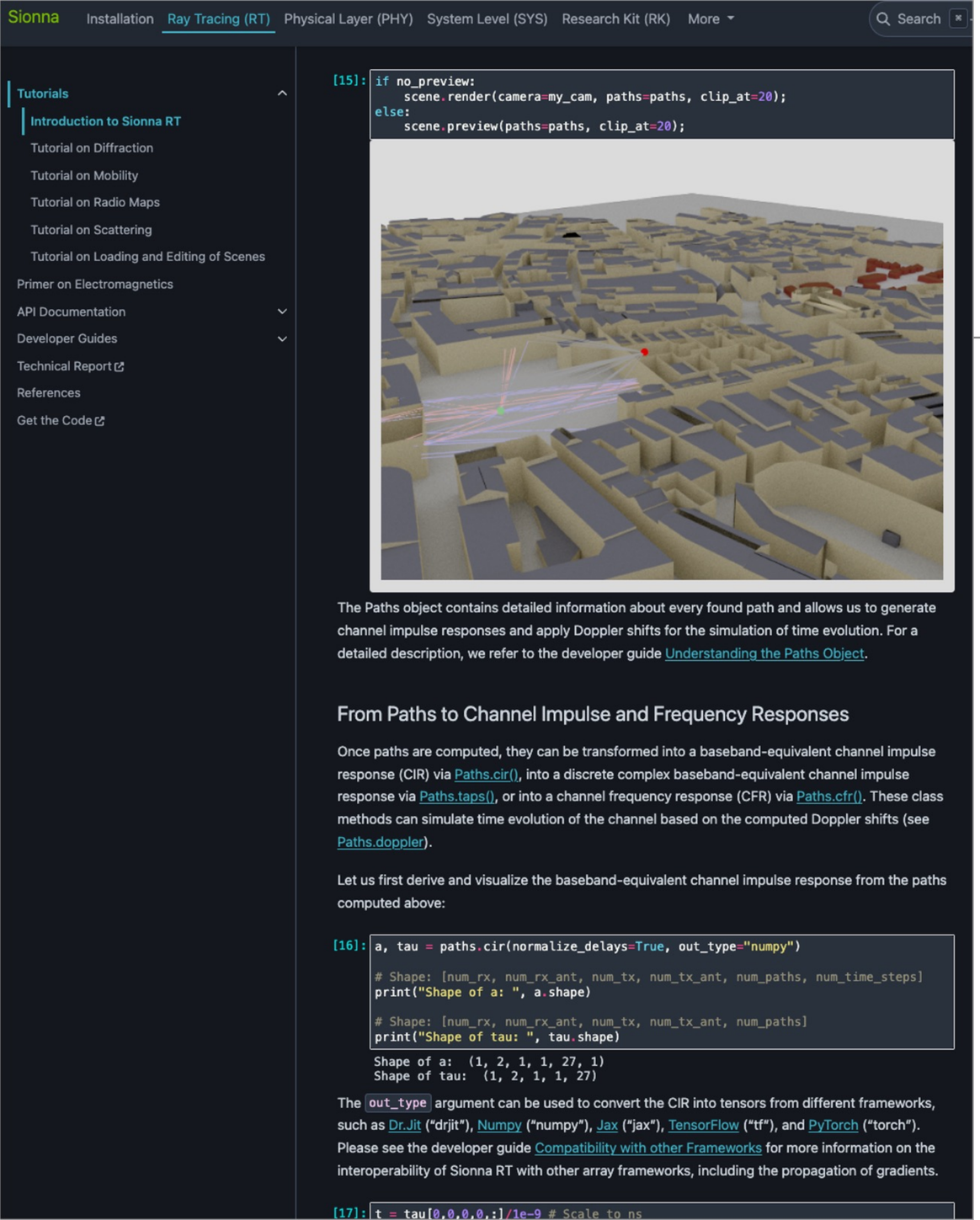
Agentic AI for Wireless Research

Sebastian Cammerer | Senior Research Scientist | June 22, 2026

Research has always evolved…

…so should we





Sionna 2.0

Migration to PyTorch

- Sionna 1.0 built on TensorFlow
  - 55,000 lines of code
  - 123 files, 84 classes, 1388 functions, 30 tutorials
  - Manual migration estimated at **18 person-months**
- Automated workflow via agentic coding tools
  - Cursor with Claude Opus 4.5 LLM
  - Module-by-module translation
  - Ensure 1300+ unit tests pass
- Migration in **one person-month**
  - Token cost ~2000$
- Now, could probably be done in less than a week!

https://nvlabs.github.io/sionna/

From 3GPP 38.901 Specification to CUDA

Automated implementation of calibrated channel models

Specifications

ETSI TR 125 996 V19.0.0 (2025-10)

ETSI TR 138 901 V14.0.0 (2017-05)

ETSI TR 138 901 V19.2.0 (2026-02)

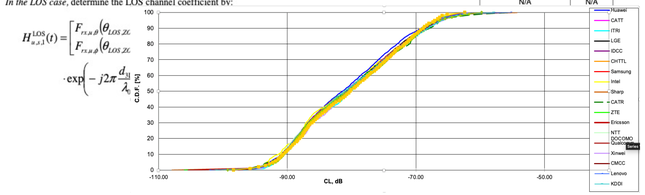
Table 7.5-6 Part-1: Channel model parameters for UMi-Street Canyon and UMa

Scenarios		UMi - Street Canyon			UMa		
		LOS	NLOS	O2I	LOS	NLOS	O2I
Delay spread (DS) lgDS=log$_{10}$(DS/1s)	μ_{lgDS}	-0.18 $\log_{10}(1+f_c)$ - 7.28, see note 8	-0.22 $\log_{10}(1+f_c)$ - 6.87, see note 8	-6.62	-7.067 - 0.0794 $\log_{10}(f_c)$, see note 8	-6.47 - 0.134 $\log_{10}(f_c)$, see note 8	-6.62
	σ_{lgDS}	0.39, see note 8	0.19 $\log_{10}(1+f_c)$ + 0.22, see note 8	0.32	0.57 + 0.026 $\log_{10}(f_c)$, see note 8	0.39 see note 9	0.32
AOD spread (ASD) lgASD=log$_{10}$(ASD/1°)	μ_{lgASD}	-0.05 $\log_{10}(1+f_c)$ + 1.21 see note 9	-0.24 $\log_{10}(1+f_c)$ + 1.54 see note 8	1.25	0.92, see note 8	1.09, see note 8	0.58, see note 8
	σ_{lgASD}	0.08 $\log_{10}(1+f_c)$ + 0.29 see note 8	0.10 $\log_{10}(1+f_c)$ + 0.33 see note 8	0.42	0.31, see note 8	0.44, see note 8	0.7 see note 8
AOA spread (ASA) lgASA=log$_{10}$(ASA/1°)	μ_{lgASA}	-0.07 $\log_{10}(1+f_c)$ + 1.66, see note 8	-0.07 $\log_{10}(1+f_c)$ + 1.76, see note 8	1.76	1.76, see note 8	2.04 - 0.25 $\log_{10}(f_c)$, see note 8	1.76
	σ_{lgASA}	0.021 $\log_{10}(1+f_c)$ + 0.26, see note 8	0.05 $\log_{10}(1+f_c)$ + 0.27, see note 8	0.16	0.19, see note 8	0.17 - 0.03 $\log_{10}(f_c)$, see note 8	0.16
ZOA spread (ZSA) lgZSA=log$_{10}$(ZSA/1°)	μ_{lgZSA}	-0.11 $\log_{10}(1+f_c)$ + 0.81 see note 8	-0.03 $\log_{10}(1+f_c)$ + 0.92 see note 8	1.01	0.96 see note 8	-0.2856 $\log_{10}(f_c)$ + 1.445, see note 8	1.01
	σ_{lgZSA}	-0.03 $\log_{10}(1+f_c)$ + 0.29 see note 8	-0.05 $\log_{10}(1+f_c)$ + 0.35 see note 8	0.43	0.15 see note 8	0.17, see note 8	0.43
Shadow fading (SF) [dB]	σ_{SF}	See Table 7.4.1-1	See Table 7.4.1-1	7	See Table 7.4.1-1	See Table 7.4.1-1	7
K-factor (K) [dB]	μ_K	9	N/A	N/A	9	N/A	N/A
	σ_K	5	N/A	N/A	3.5	N/A	N/A
	ASD vs DS	0.5	0	0.4	0.4	0.4	0.4
						0.6	0.4
						0	0
						-0.6	0.2
						-0.4	-0.5
						0.4	0
						N/A	N/A
						N/A	N/A
						N/A	N/A
						N/A	N/A
						0	0
						-0.4	0
						N/A	N/A

$$H_{u,s,j,n,m}^{\mathrm{NLOS}}(t)=\sqrt{\frac{P_n}{M}}\begin{bmatrix}F_{rx,\theta}\left(\theta_{n,m,ZOA},\phi_{n,m,AOA}\right)\\ F_{rx,\phi}\left(\theta_{n,m,ZOA},\phi_{n,m,AOA}\right)\end{bmatrix}^{T}\begin{bmatrix}\exp\left(j\Phi_{n,m}^{\theta\theta}\right) & \sqrt{\kappa_{n,m}^{-1}}\exp\left(j\Phi_{n,m}^{\theta\phi}\right)\\ \sqrt{\kappa_{n,m}^{-1}}\exp\left(j\Phi_{n,m}^{\phi\theta}\right) & \exp\left(j\Phi_{n,m}^{\phi\phi}\right)\end{bmatrix}\cdot\begin{bmatrix}F_{tx,\theta}\left(\theta_{n,m,ZOD},\phi_{n,m,AOD}\right)\\ F_{tx,\phi}\left(\theta_{n,m,ZOD},\phi_{n,m,AOD}\right)\end{bmatrix}\exp\left(j2\pi\frac{\hat{r}_{rx,n,m}^{T}\vec{d}_{rx,n}}{\lambda_0}\right)\exp\left(j2\pi\frac{\hat{r}_{tx,n,m}^{T}\vec{d}_{tx,n}}{\lambda_0}\right)\exp\left(j2\pi\frac{\hat{r}_{rx,n,m}^{T}\vec{v}}{\lambda_0}t\right) \quad (7.5\text{-}28)$$

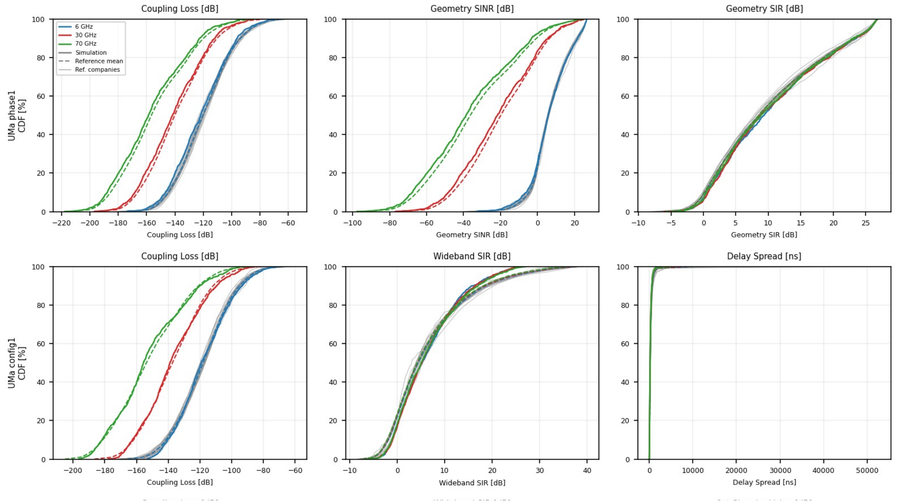
In the LOS case, determine the LOS channel coefficient by:

$$H_{u,s,j}^{\mathrm{LOS}}(t)=\begin{bmatrix}F_{rx,\theta}\left(\theta_{LOS,ZL}\right)\\ F_{rx,\phi}\left(\theta_{LOS,ZL}\right)\end{bmatrix}^{T}\cdot\exp\left(-j2\pi\frac{d_{3D,us}}{\lambda_0}\right)$$



Claude Code, Opus 4.6
Autorun (Sandbox)
48h/1500$

Calibrated channel simulator (Python/CUDA)



Where Claude struggled:

- Complex nested tables
- Description of calibration metrics in prose
- Hexagonal grid with 57 sectors, wrap-around, user drops

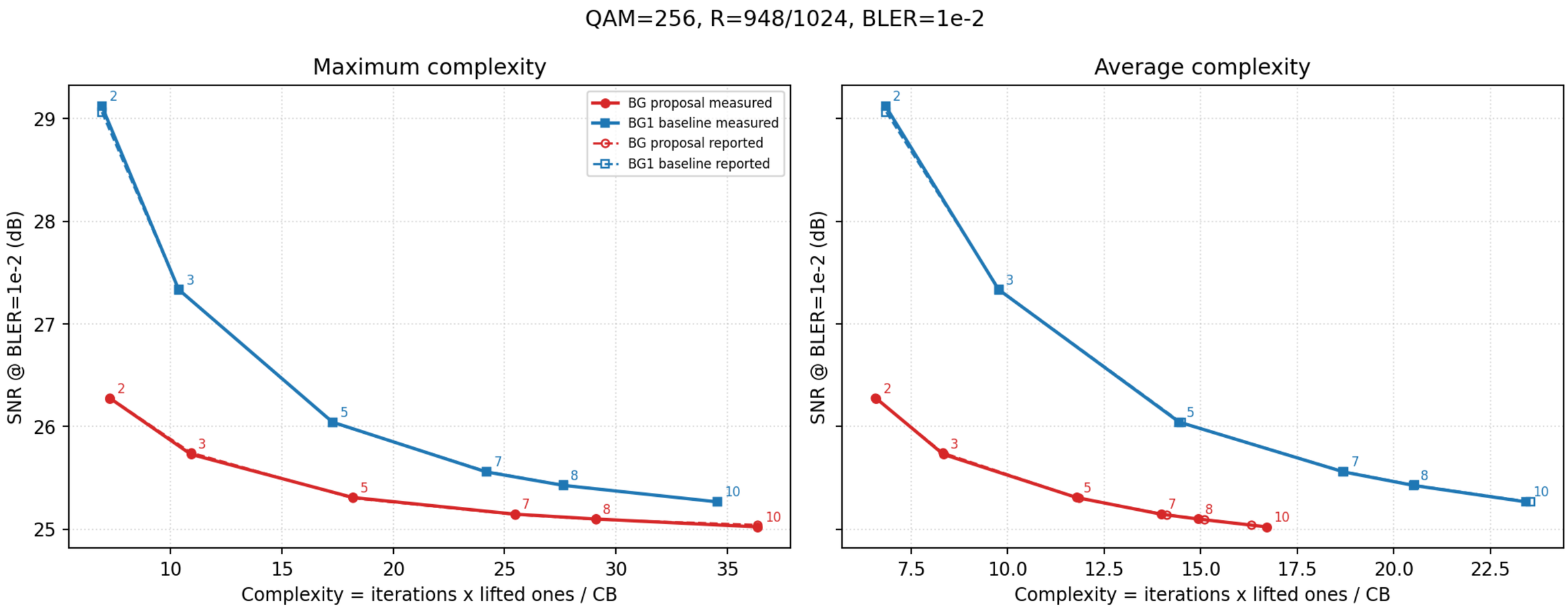
All due to lacking “machine-friendliness”

Can this be done for the full 5G/6G specs?

NVIDIA

The Agentic 3GPP Backoffice

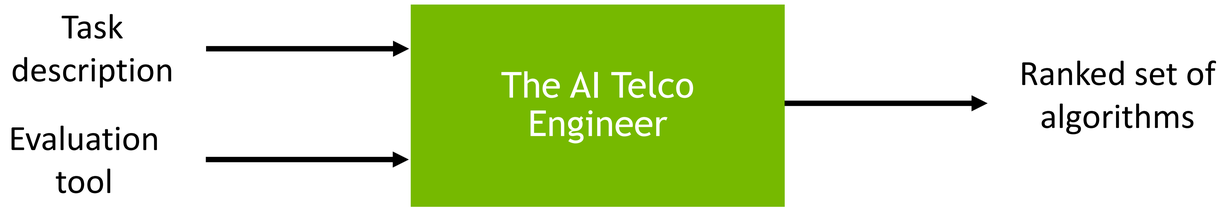
Automated evaluation of new LDPC BG3 candidates in CUDA



The AI Telco Engineer

Exploring Agentic AI for Wireless Research

Given a problem, generate best algorithm according to user-defined metric



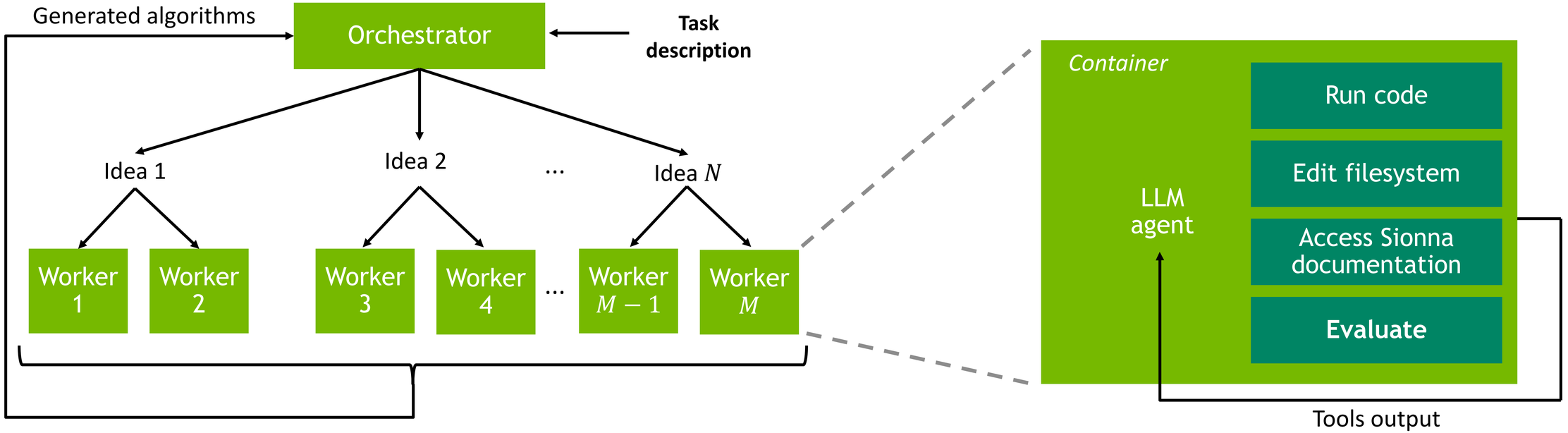
- Task description defines the wireless communication problem for the agentic framework
- Evaluation tool runs the generated algorithm and computes its performance metric

https://github.com/NVlabs/the-ai-telco-engineer

The AI Telco Engineer

Agentic AI framework

- Idea-driven framework: An orchestrator generates ideas that are distributed to parallel agents
- Use LLMs to explore algorithm designs for a user-provided wireless problem
- Workers run in a containerized sandbox and iteratively improve designs via trial and error
- User-defined evaluation is immutable (code + data) to prevent metric gaming



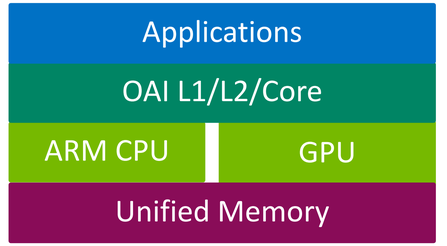
Toward Agents with Hardware-in-the-Loop

Sionna Research Kit

Realize ideas in an actual 5G network

Channel

Fronthaul



5G Modem | Cable/Antenna | USRP | USB 3.0 | NVIDIA DGX Spark

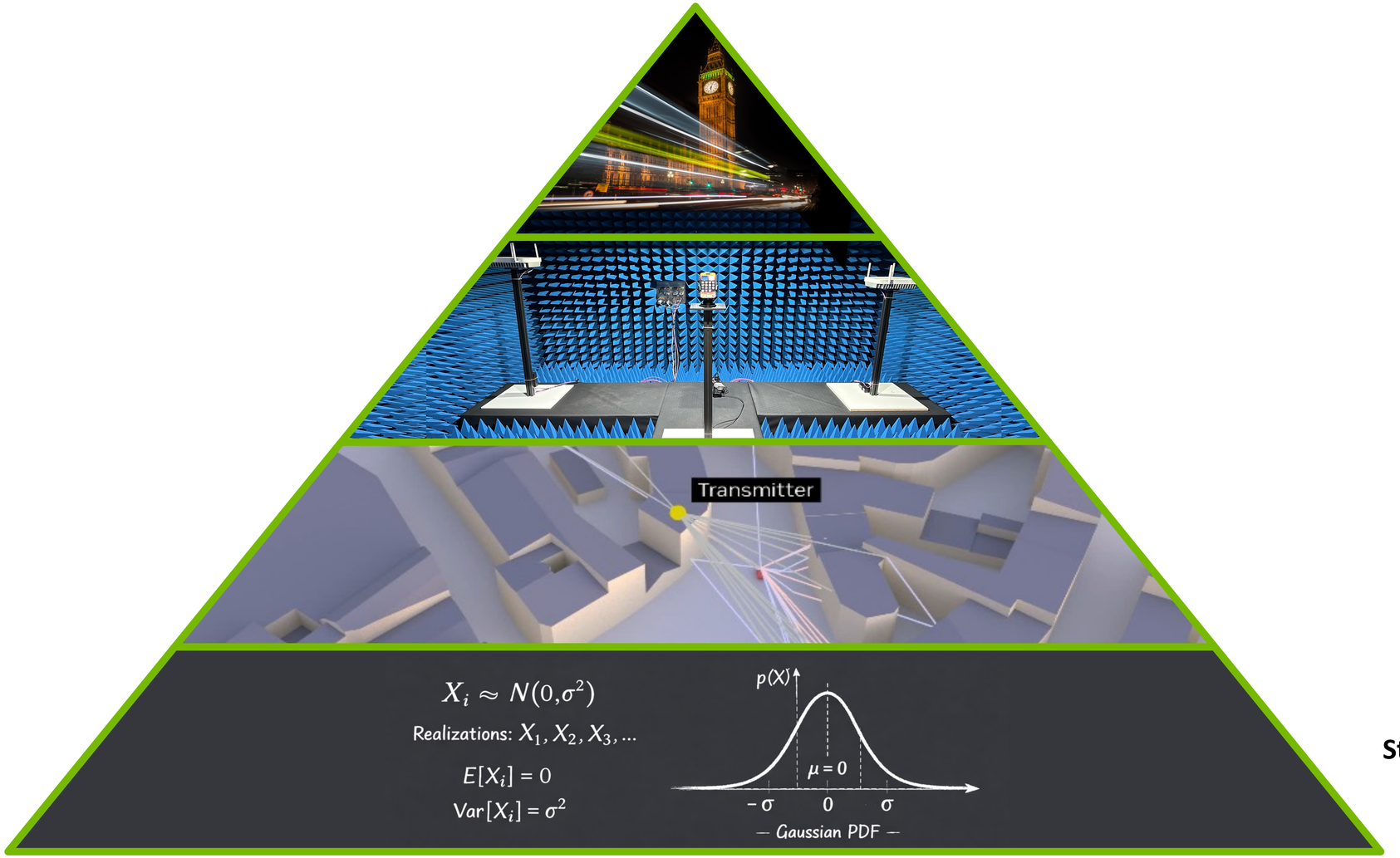
- Quick start in an afternoon
- Affordable (6-8k$)
- GPU-accelerated
- Tutorials
- Blueprints for own experiments
- Open source

The Economics of Training Data

Start with simulated data, and don't stop there

Cost

Diversity



Real world

Testbed

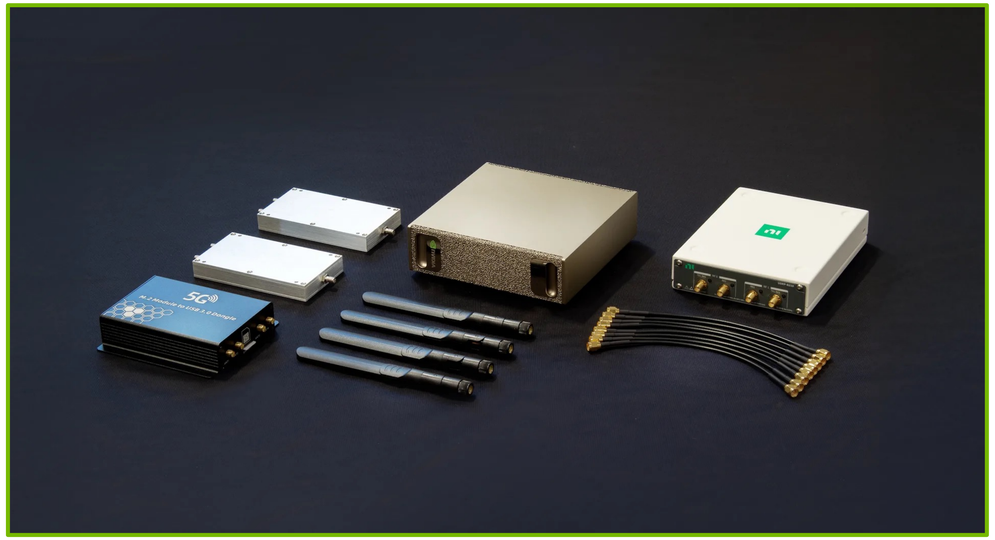
Ray tracing

Stochastic channel models

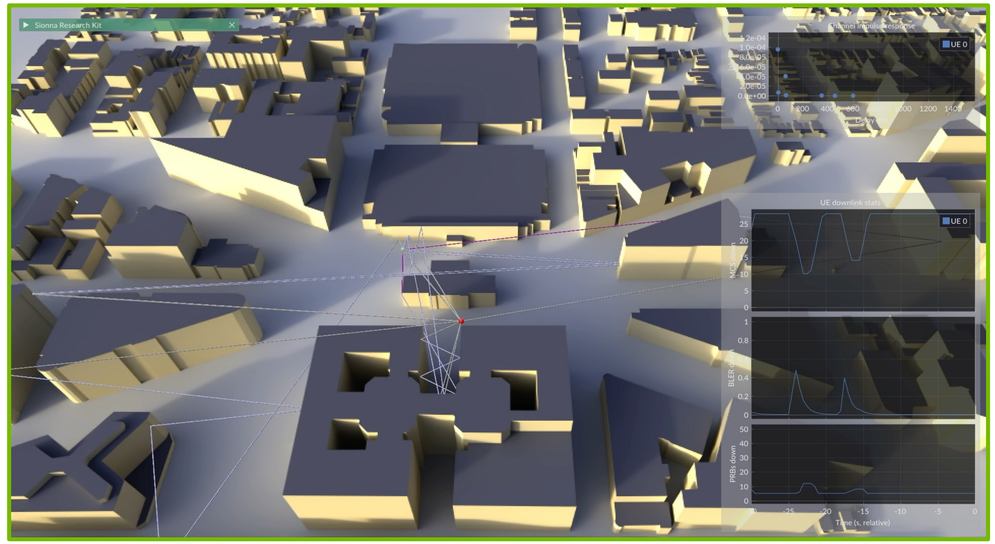
NVIDIA.

Sionna Research Kit

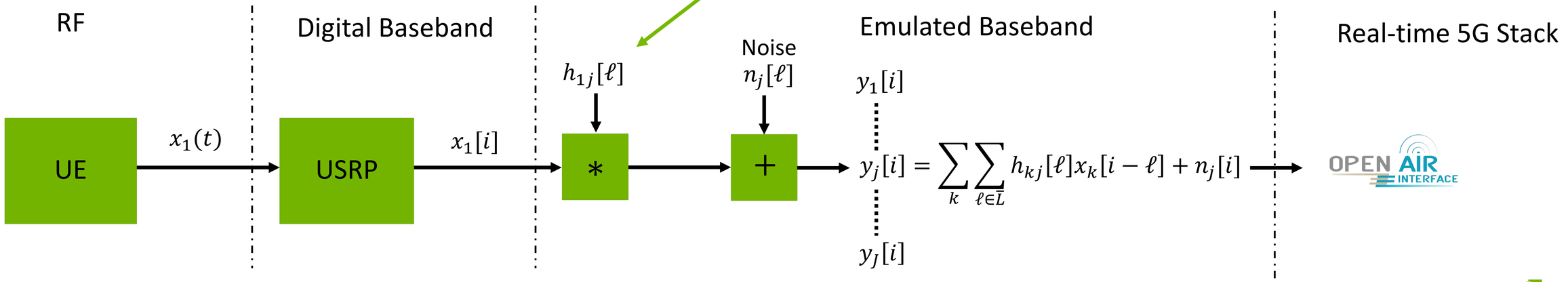
Real-time channel emulation integrated in the OpenAirInterface Radio Unit (RU)



Real-world RF impairments



Emulated multipath propagation

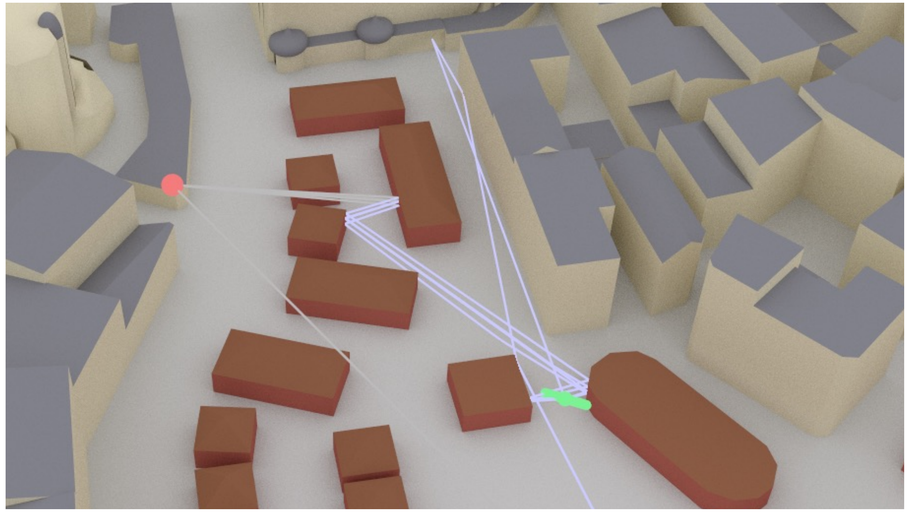


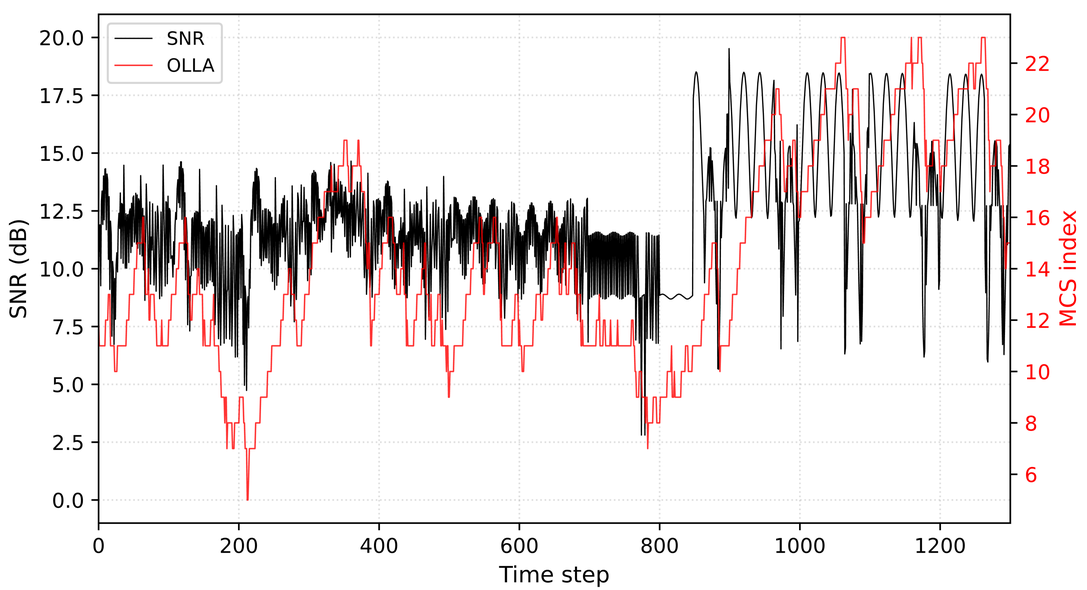

5G
M.2 Module to USB 3.0 Dongle
INFORMATION SOURCE
MESSAGE
UE
SIGNAL
CHANNEL
RECEIVED SIGNAL
gNB
DESTINATION
MESSAGE
NOISE SOURCE
Fig. 1—Schematic diagram of a general communication system.
NVIDIA

Task: Link Adaptation

State-of-the-art performance in 8 hours for 100$

```
Implement an MCS (Modulation and Coding Scheme) selection controller for link
adaptation that maximizes spectral efficiency while maintaining a target Block
Error Rate (BLER). Your goal is to maximize throughput (spectral efficiency in
bits/s/Hz) while keeping the long-term BLER below or equal to 10%. [...]
```



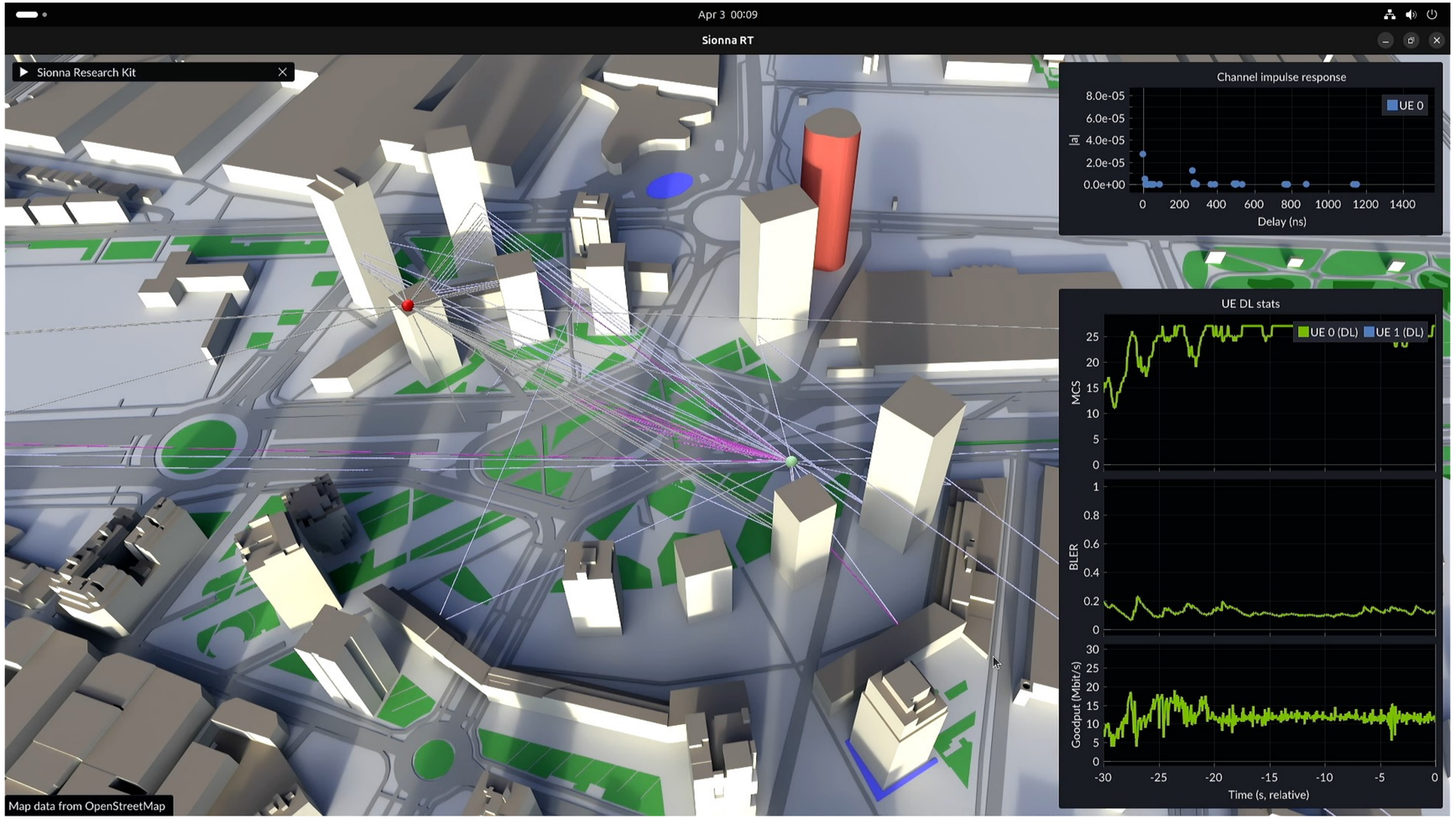


- Metric: Average spectral efficiency over 50 ray-traced channel sequences of 3000 time slots
- Algorithm has only access to ACK/NACK history

~3.4% higher spectral efficiency vs. OLLA

From Python to OAI in Minutes

Automated agentic conversion of the link-adaptation algorithm to OAI Plugin



Original algorithm was never exposed to long sequences of (N)ACKs during evolution

Agent fixes (N)ACK history to 64 observations

NVIDIA

Hierarchical Agents

Leveraging simulations, digital twins, and real networks as evaluation tools

C/CUDA Code (e.g., OAI Plugin)

Sim2Real Gap

hrs/days

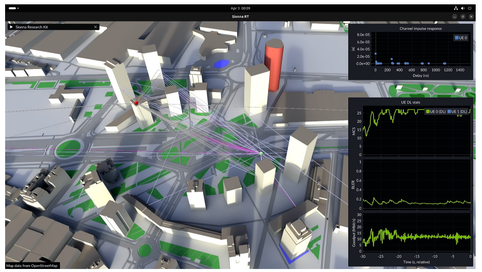
Real Network

Python Code (e.g., Sionna Block)

Full-stack Integration

min/hrs

Digital Twin

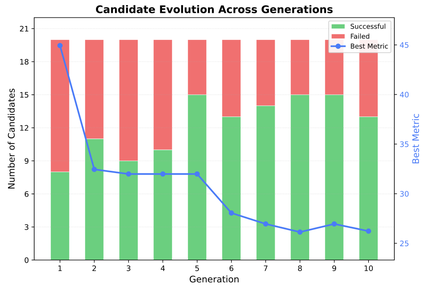


Data for better Digital Twin

Algorithm Evolution

sec/min

Link-level Simulation



Data for better Simulator

NVIDIA

What’s Possible with Unlimited Tokens?

Project “nanoRAN”

Agentic AI to write minimalistic 5G stack

Bring up a **minimalist 5G NR gNB (nanoRAN)** on a USRP B210 that lets a real Quectel RM520N-GL modem attach to an OAI 5GC over a cabled n78 link and run stable iperf. The gNB is greenfield — Karpathy-style single-purpose code, unified-memory IQ fast path, nanobind-only bindings, every CUDA kernel paired with a Python reference. **The MVP bar is one modem, one ping.**

Provide access to

- 3GPP TDocs 38.XXX
- OpenAirInterface & Sionna Research Kit
- OCUDU
- Aerial CUDA accelerated RAN (L1)

Implement. Debug. Repeat.

Can agents implement their own 5G stack?

```
Massive progress validated by OAI pcap comparison:

START today: UE not transmitting Msg3 (regression hidden in env defaults)

  → After 30 commits + OAI tcpdump diagnostic:

END: chain reaches PDUSessionResourceSetupResponse to AMF
    PRACH ✅ → Msg3 ✅ → Msg5 ✅ → Auth ✅ → SecMode ✅
    → RegistrationAccept ✅ → ICS Response ✅
    → AMF release ✅ (correct!)
    → UE 2nd RA with ServiceRequest ✅ (the OAI flow)
    → UE PDU Session Establishment Request decoded ✅
    → AMF: PDUSessionResourceSetupRequest ×16
    → gNB: RRCReconfig+DRB ×8 patched
    → gNB: PDUSessionResourceSetupResponse ×8 to AMF (speculative, OAI-style)
    → NGAP chain COMPLETE end-to-end
    → drb_active=0, tunnel_rx=0 ← UE doesn't apply our RRCReconfig

Key debugging-value lessons today

1. "OAI works" is a powerful debugging anchor - comparing tcpdump to ours definitively settled multiple speculations (nas_len=94 was wrong, our deferred-ICS-response was the bug, AT+CGACT is the
trigger).
2. The PUSCH-RX delay-derotation fix (3a60e3c) was the biggest algorithmic win - coh 0.45 → 0.999. Permanent improvement.
3. The remaining gap is RRC ASN.1 content, not RF and not gNB protocol logic. Need to ASN.1-diff our RRCReconfiguration vs OAI's. Both pcaps are preserved (/tmp/oai_attach_1778145373.pcap,
/tmp/nanoran_full_*.pcap).

Concrete next session tasks

1. tshark -r /tmp/oai_attach_1778145373.pcap -Y rrc -V - extract OAI's RRCReconfiguration container
2. Same for nanoran's pcap
3. Diff the two ASN.1 trees - focus on: SRB/DRB IDs, SDAP config, PDCP config, RLC config, MAC config, BWP config
4. Likely ~1-2 fields don't match. Fix in nanoran/rrc/setup.py or wherever DRB config is built.

The session ends with the NGAP-level chain matching OAI byte-for-byte (more or less), and a clear concrete debugging plan for closing the radio-side DRB activation. 832/832 tests pass throughout.
```

Auto-run for 4 weeks (with interrupts)

- Claude Code (Opus 4.7) and OpenAI Codex (GPT 5.5)
- Most time spend on debugging and running the system(!)
- Token cost <10k $

Observations

- Internal debugging with OAI as reference
- RFSim mode is used a lot (~digital twins)
- Efficient hardware usage (*“AT commands”)* & RF debugging
- Human guidance still needed(!)

Project ”nanoRAN”

First step towards a fully agentic 6G stack

Repository structure

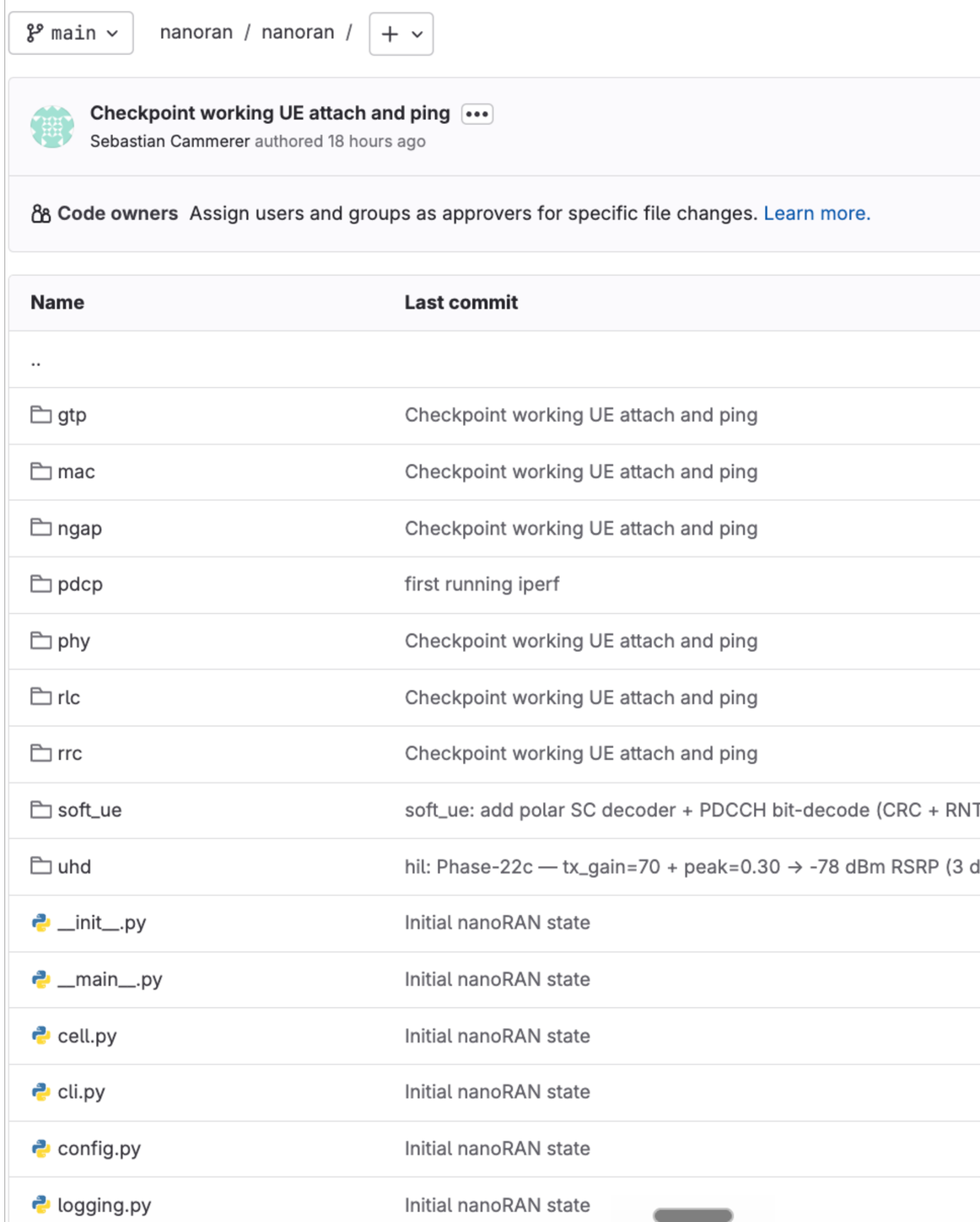
- 79% Python
- 1% CUDA
- 18% C/C++
- 2% Shell
- ASN1 used as external dependency

Connects to OpenAirInterface 5G Core

Current challenges

- Individual testability of blocks
- Digital twins & closed-loop testbeds
- Real-time & RF debugging

It’s not an autonomous stack only from TDocs yet!



Hello World?

World’s first ping over a fully agentic 5G stack

```
== Wait For UE Attach ==
waiting for UE attach/IP ... elapsed=15s
waiting for UE attach/IP ... elapsed=31s
waiting for UE attach/IP ... elapsed=47s
waiting for UE attach/IP ... elapsed=63s
waiting for UE attach/IP ... elapsed=79s
waiting for UE attach/IP ... elapsed=95s
UE attached  : yes
UE IP        : 12.1.1.40

== Network State ==
wwan0       : wwan0             UNKNOWN        12.1.1.40/28



== Downlink Ping ==
oai-ext-dn -> UE (12.1.1.40)
PING 12.1.1.40 (12.1.1.40) 56(84) bytes of data.
64 bytes from 12.1.1.40: icmp_seq=1 ttl=63 time=560 ms
64 bytes from 12.1.1.40: icmp_seq=2 ttl=63 time=747 ms
64 bytes from 12.1.1.40: icmp_seq=3 ttl=63 time=758 ms
64 bytes from 12.1.1.40: icmp_seq=4 ttl=63 time=1728 ms
64 bytes from 12.1.1.40: icmp_seq=5 ttl=63 time=1521 ms
64 bytes from 12.1.1.40: icmp_seq=6 ttl=63 time=1320 ms
64 bytes from 12.1.1.40: icmp_seq=7 ttl=63 time=1501 ms
64 bytes from 12.1.1.40: icmp_seq=8 ttl=63 time=1294 ms
64 bytes from 12.1.1.40: icmp_seq=9 ttl=63 time=1092 ms
64 bytes from 12.1.1.40: icmp_seq=10 ttl=63 time=1598 ms
64 bytes from 12.1.1.40: icmp_seq=11 ttl=63 time=1755 ms
64 bytes from 12.1.1.40: icmp_seq=12 ttl=63 time=1719 ms
64 bytes from 12.1.1.40: icmp_seq=13 ttl=63 time=1534 ms
64 bytes from 12.1.1.40: icmp_seq=14 ttl=63 time=1694 ms

--- 12.1.1.40 ping statistics ---
20 packets transmitted, 14 received, 30% packet loss, time 3878ms
rtt min/avg/max/mdev = 560.293/1344.387/1754.581/388.950 ms, pipe 9
```

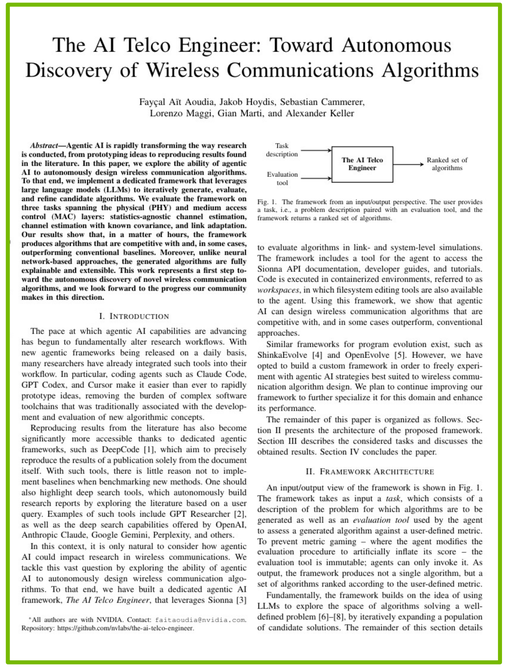
Research at the Speed of Inference

Agentic code generation

+

Open tools, platforms & twins

+



The AI Telco Engineer: Toward Autonomous Discovery of Wireless Communications Algorithms

Fayçal Ait Aoudia, Jakob Hoydis, Sebastian Cammerer, Lorenzo Maggi, Gian Marti, and Alexander Keller

Abstract—**Agentic AI is rapidly transforming the way research is conducted, from prototyping ideas to reproducing results found in the literature. In this paper, we explore the ability of agentic AI to autonomously design wireless communication algorithms. To that end, we implement a dedicated framework that leverages large language models (LLMs) to iteratively generate, evaluate, and refine candidate algorithms. We evaluate the framework on three tasks spanning the physical (PHY) and medium access control (MAC) layers: statistics-agnostic channel estimation, channel estimation with known covariance, and link adaptation. Our results show that, in a matter of hours, the framework produces algorithms that are competitive with and, in some cases, outperforming conventional baselines. Moreover, unlike neural network-based approaches, the generated algorithms are fully explainable and extensible. This work represents a first step toward the autonomous discovery of novel wireless communication algorithms, and we look forward to the progress our community makes in this direction.**

I. Introduction

The pace at which agentic AI capabilities are advancing has begun to fundamentally alter research workflows. With new agentic frameworks being released on a daily basis, many researchers have already integrated such tools into their workflow. In particular, coding agents such as Claude Code, GPT Codex, and Cursor make it easier than ever to rapidly prototype ideas, removing the burden of complex software toolchains that was traditionally associated with the development and evaluation of new algorithmic concepts.

Reproducing results from the literature has also become significantly more accessible thanks to dedicated agentic frameworks, such as DeepCode [1], which aim to precisely reproduce the results of a publication solely from the document itself. With such tools, there is little reason not to implement baselines when benchmarking new methods. One should also highlight deep search tools, which autonomously build research reports by exploring the literature based on a user query. Examples of such tools include GPT Researcher [2], as well as the deep search capabilities offered by OpenAI, Anthropic Claude, Google Gemini, Perplexity, and others.

In this context, it is only natural to consider how agentic AI could impact research in wireless communications. We tackle this vast question by exploring the ability of agentic AI to autonomously design wireless communication algorithms. To that end, we have built a dedicated agentic AI framework, *The AI Telco Engineer*, that leverages Sionna [3]

*All authors are with NVIDIA. Contact: faitaoudia@nvidia.com.
Repository: https://github.com/nvlabs/the-ai-telco-engineer.

Fig. 1. The framework from an input/output perspective. The user provides a task, i.e., a problem description paired with an evaluation tool, and the framework returns a ranked set of algorithms.

to evaluate algorithms in link- and system-level simulations. The framework includes a tool for the agent to access the Sionna API documentation, developer guides, and tutorials. Code is executed in containerized environments, referred to as *workspaces*, in which filesystem editing tools are also available to the agent. Using this framework, we show that agentic AI can design wireless communication algorithms that are competitive with, and in some cases outperform, conventional approaches.

Similar frameworks for program evolution exist, such as ShinkaEvolve [4] and OpenEvolve [5]. However, we have opted to build a custom framework in order to freely experiment with agentic AI strategies best suited to wireless communication algorithm design. We plan to continue improving our framework to further specialize it for this domain and enhance its performance.

The remainder of this paper is organized as follows. Section II presents the architecture of the proposed framework. Section III describes the considered tasks and discusses the obtained results. Section IV concludes the paper.

II. Framework Architecture

An input/output view of the framework is shown in Fig. 1. The framework takes as input a *task*, which consists of a description of the problem for which algorithms are to be generated as well as an *evaluation tool* used by the agent to assess a generated algorithm against a user-defined metric. To prevent metric gaming – where the agent modifies the evaluation procedure to artificially inflate its score – the evaluation tool is immutable; agents can only invoke it. As output, the framework produces not a single algorithm, but a set of algorithms ranked according to the user-defined metric.

Fundamentally, the framework builds on the idea of using LLMs to explore the space of algorithms solving a well-defined problem [6]–[8], by iteratively expanding a population of candidate solutions. The remainder of this section details

Open research problems

=

New era of wireless R&D

```
> pip install sionna
> make sionna-rk
```