Tiny-Twin: A CPU-Native Full-stack Digital Twin for NextG Cellular Networks

Ali Mamaghani[1], Ushasi Ghosh[1], Ish Kumar Jain[1,2], Srinivas Shakkottai[3] and Dinesh Bharadia[1]
[1]University of California San Diego, CA, [2]Rensselaer Polytechnic Institute, NY, [3]Texas A&M University, TX
{amamaghani, ughosh, ikjain, dineshb}@ucsd.edu {sshakkot}@tamu.edu

***Abstract*—Modern wireless applications demand testing environments that capture the full complexity of next-generation (NextG) cellular networks. While digital twins promise realistic emulation, existing solutions often compromise on physical-layer fidelity and scalability or depend on specialized hardware. We present Tiny-Twin, a CPU-Native, full-stack digital twin framework that enables realistic, repeatable 5G experimentation on commodity CPUs. Tiny-Twin integrates time-varying multi-tap convolution with a complete 5G protocol stack, supporting plug-and-play replay of diverse channel traces. Through a redesigned software architecture and system-level optimizations Tiny-Twin supports fine-grained convolution entirely in software. With built-in real-time RIC integration and per User Equipment(UE) channel isolation, it facilitates rigorous testing of network algorithms and protocol designs. Our evaluation shows that Tiny-Twin scales to multiple concurrent UEs while preserving protocol timing and end-to-end behavior, delivering a practical middle ground between low-fidelity simulators and high-cost hardware emulators. We release Tiny-Twin as an open-source platform to enable accessible, high-fidelity experimentation for NextG cellular research.**
Code available at: **https://github.com/ucsdwcsng/Tiny_Twin**

***Index Terms*—Digital Twin, Cellular Networks, Full Stack 5G;**

I. Introduction

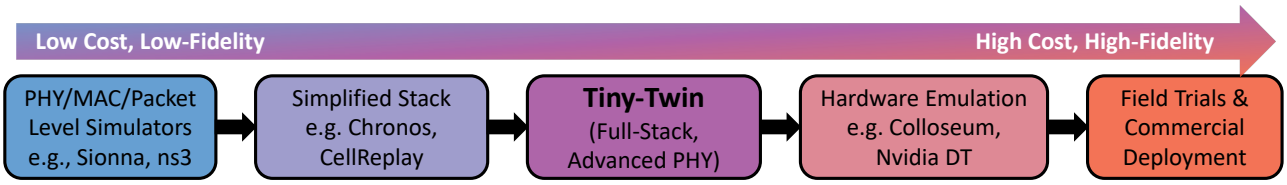
Next-generation cellular networks are evolving to support demanding applications like autonomous systems, extended reality (XR), and advanced IoT, each of them requires diverse application requirements beyond the traditional low-latency, reliability, and high-throughput wireless access. In response, the telecom industry is shifting toward highly programmable infrastructure such as Open Radio Access Network (Open-RAN), which allows for scenario-specific optimization through AI/ML-driven control loops This shift has sparked a new wave of wireless network co-design, where researchers develop AI-optimized 5G algorithms, such as link adaptation, mobility management, or RL-based resource allocation, that is tightly coupled with application requirements [1]. However, understanding how these algorithmic decisions affect end-to-end performance remains a major challenge due to complex interactions across multiple layers of the RAN protocol stack. As a result, there is a growing need for highly available digital twins, which can accurately reflect the behavior of a real-world cellular network.

Such Digital Twin need to follow two main requirements: *High-fidelity* and *Low-cost*. High fidelity refers to the ability to emulate the full protocol stack with fine-grained physical layer attributes such as accurate channel models. Full-stack is required for capturing essential cross-layer effects such as retransmissions, buffering, and packet reordering. Without this realism, critical metrics like latency, jitter, and throughput become distorted, limiting the accuracy of the twin for evaluating AI-driven control loops and network co-design. It also requires accurate PHY channel modeling beyond simplified abstractions (e.g., averaged Signal-to-Noise Ratio(SNR) or Channel Quality Indicator (CQI) traces) to capture multipath fading, Doppler spread, and time-frequency selectivity-factors critical for realistic evaluation of functions like Modulation and Coding Scheme (MCS) selection [2], beamforming [3], and scheduling [4]. The second requirement is a low cost, ensuring that such fidelity can be achieved on commodity CPUs rather than specialized FPGA- or GPU-based testbeds. This requirement is crucial for scalability and accessibility, enabling widespread experimentation and AI-optimized application development.

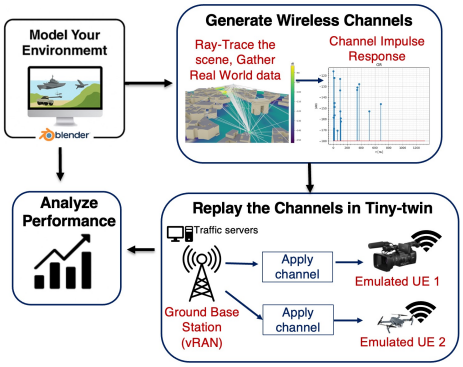
However, state-of-the-art Digital Twins do not meet these requirements as shown through three broad classes in Fig.1(a). The first class comprises basic low-cost simulators, such as MATLAB 5G toolbox, Sionna, or ns-3, with a focus on link-level behavior or simplified protocol models. While useful for targeted analysis, they lack integration with a full 5G protocol stack, limiting their utility for high-fidelity end-to-end evaluation. The second class primarily targets the upper layers of the protocol stack [5], [6]. While simplifying deployment, this abstraction also sacrifices fidelity, especially regarding PHY wireless channel impairments. The third class rely on custom hardware accelerators or FPGA-based platforms, needed to emulate the electromagnetic environment with high fidelity, such as Colosseum [7] and Nvidia Digital Twin [8]. While accurate, these systems are high-cost, hardware-specific, and difficult to replicate with extended setup durations, forming a significant barrier to broader experimentation and rapid prototyping. Thus, there exists a trade-off between cost and fidelity.

To address the critical need for high-fidelity and low-cost wireless experimentation platform, we present **Tiny-twin**—a CPU-Native, full-stack digital twin framework for 5G networks. Positioned as a middle ground between abstract L2 simulators and hardware-intensive emulators (Fig.1a), Tiny-Twin is designed for low-cost and high-fidelity framework. First, it delivers *low-cost* by capturing fine-grained physical layer modeling entirely in software, allowing execution on commodity CPUs without reliance on specialized accelerators, thereby improving deployability and broadening accessibility.

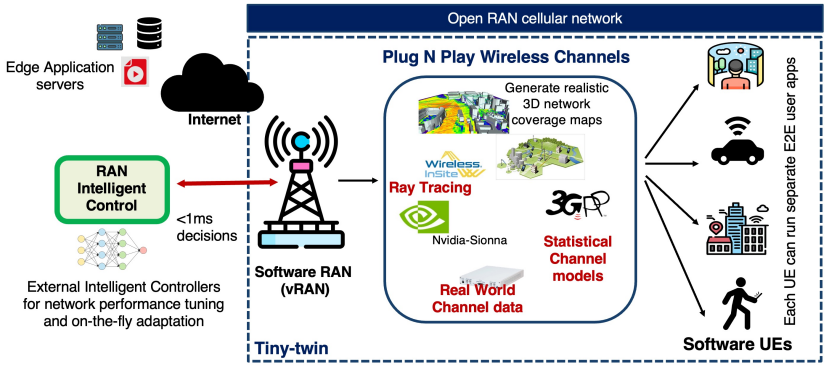
A key challenge is serving multiple computationally de-



a) Continuum of testing platforms for wireless systems. Tiny-Twin provides high fidelity while maintaining low cost



b) Simulate, Replay and Evaluate the 5G network in Tiny-twin



c) Tiny-twin: Low-cost and High-fidelity 5G Digital Twin

Fig. 1: Tiny-Twin Overview: Building a low-cost high-fidelity Digital Twin for Wireless Cellular Networks

manding UEs in the digital twin environment This is possible by multiple system-level optimizations, such as parallelizing convolution computation, sparse convolution, and CPU pinning techniques (See Section III-C) Secondly, Tiny-Twin operates as a *high-fidelity* plug-and-play channel replay engine (Fig.1 (b), (c)), uniquely capable of injecting time-varying channel traces into a live 5G full protocol stack built atop OpenAirInterface [9], achieving performance remarkably close to real time. Channel traces can originate from a diverse set of sources, including 3GPP statistical models, ray-traced urban deployments, simplified synthetic abstractions, or most importantly, complex real-world measurement campaigns. The ability to capture and replay real-world wireless environments with high fidelity is a defining strength of Tiny-Twin. It enables developers to emulate specific channel scenarios repeatedly and deterministically, supporting rigorous benchmarking, algorithm comparison, and application validation. Finally, its tight integration with a real-time RIC platform [1] provides fine-grained control interfaces for AI-driven adaptation, enabling the development and debugging of closed-loop control policies that respond to sub-millisecond wireless fluctuations in a low-cost and high-fidelity environment.

We make the following contribution in this paper:

- We design and implement Tiny-Twin, a low-cost, high-fidelity, CPU-native digital twin for full-stack 5G cellular networks, enabling realistic experimentation on commodity hardware without specialized accelerators.
- We identify key challenges in achieving channel realism and fast execution on CPUs and introduce system-level optimizations that enable real-time multi-tap channel convolution with over 20 channel taps.
- We demonstrate that Tiny-Twin scales to support up to 10 concurrent user equipments (UEs) while maintaining real-time operation and protocol-level fidelity.
- We develop an open-source Grafana-based monitoring framework that provides real-time visibility into PHY-, MAC-, and end-to-end system metrics, enabling detailed analysis and closed-loop experimentation.
- We make the Tiny-Twin codebase publicly available[1] to support reproducibility and facilitate further research in low-cost, high-fidelity cellular digital twins.

II. Related Work

Table I highlights how our approach compares to existing cellular digital twin frameworks.

High-Fidelity, High-Cost DT: Frameworks that focus on high-fidelity emulation of the radio access network (RAN) include Colosseum's digital twin platform [10]. It leverages USRP software-defined radios (SDRs) and FPGA-accelerated I/Q processing to emulate 5G signal propagation. This fidelity comes at a high cost, requiring terabytes of I/Q storage and commercial-grade SDR infrastructure. Similarly, NVIDIA's Aerial platform [8], [17] utilizes A100 and H100 GPUs, as well as BlueField DPUs, to enable GPU-accelerated PHY simulation and ray-tracing-based wireless modeling. These platforms support research in AI-native RANs, but their reliance on datacenter-class hardware limits adoption by developers and researchers operating under resource constraints. High-end GPU nodes (e.g., A100/H100) can cost $25K–$40K per server and require specialized cooling, whereas x86 CPU systems can be built for under $3K, enabling scalable and cost-efficient deployment. In contrast, Tiny-Twin is low-cost as it is computationally feasible on commodity CPU hardware, eliminating the need for FPGAs or GPUs. Tiny-twin allows us to retain channel-aware features essential for protocol

[1]Tiny-Twin software: https://github.com/ucsdwcsng/Tiny_Twin

TABLE I: Comparison of Digital Twin frameworks

Framework	Channel Realism	Full-stack Processing	Support Protocol-level study	Hardware Requirement	Compute Time	System Scalability
Colosseum DT [10]	✓	✓	✓	FPGA	1ms	✓
NVIDIA DT [8]	✓	✓	✓	GPU	1ms	✓
ns-3 [11]	✗	✗	✓	CPU	high	✓
Sionna [12]	✗	✗	✗	GPU	N/A	✗
Chronos [5]	✗	✓	✓	CPU	N/A	✓
CellReplay [6]	✗	✗	✗	CPU	N/A	✓
Mahimahi [13]	✗	✗	✗	CPU	N/A	✓
OWDT [14]	✓	✓	✓	GPU	high	✗
OAI rfsim [9]	✗	✓	✓	CPU	8ms	✗
SRSRAN ZMQ [15]	✗	✓	✓	CPU	1ms	✗
UERANSIM [16]	✗	✓	✓	CPU	N/A	✓
Tiny-twin (ours)	✓	✓	✓	**CPU**	**2ms**	✓

evaluation, while avoiding the full complexity of detailed PHY signal processing.

Low-Fidelity Low-Cost DT: On the other hand, event driven simulators such as ns-3 [11] and UERANSIM [16] have been widely adopted for 5G network research. But they often struggle to capture the full fidelity of a real-time 5G protocol stack. These platforms typically abstract away lower-layer timing constraints and complete protocol functionality, which contributes to the well-known sim-to-real gap. As a result, protocols and algorithms validated in such environments may fail to generalize or behave as expected under real-world deployment conditions.

In parallel, network emulators like Mahimahi [13]and CellReplay [6] are widely used in the systems community to evaluate transport protocols, adaptive bitrate (ABR) algorithms [18], and congestion control schemes [19], [20] by replaying pre-recorded network traces from commercial deployments. While convenient, these trace-driven tools lack a real protocol stack and cannot model feedback-driven behaviors such as hybrid automatic repeat request (HARQ), scheduling adaptations, or stack-level retransmissions. They operate only at the network layer, offering no control over the 5G stack. As a result, they are unsuitable for studying various cellular network function algorithms, such as adaptive bitrate policies, or application-aware scheduling, on end-to-end performance. Other full-stack frameworks like Chronos [5] and [21] provide end-to-end 5G emulation environments, but they deliberately omit fine-grained PHY-layer modeling for scalability. Furthermore, full-stack digital twins trying to achieve high fidelity, such as the Open Wireless Digital Twin (OWDT) [14], are being developed to concentrate on modeling the mobility of a single User Equipment (UE) within a simulated environment. A key function of these twins is the estimation of wireless channel characteristics leveraging raytracing techniques. However, they do not present a scalable system design that can support multiple users while maintaining PHY fidelity, and suffer from high compute time for a multi-user system, thereby not scalable. In contrast, Tiny-Twin provides high-fidelity by leveraging an open-source 5G protocol stack, enabling experimentation with network algorithms and RIC-based real-time optimization applications [1]. Our high-fidelity design includes a finite-tap, time-domain channel convolution engine to support realistic PHY effects.

III. Tiny-Twin System Design

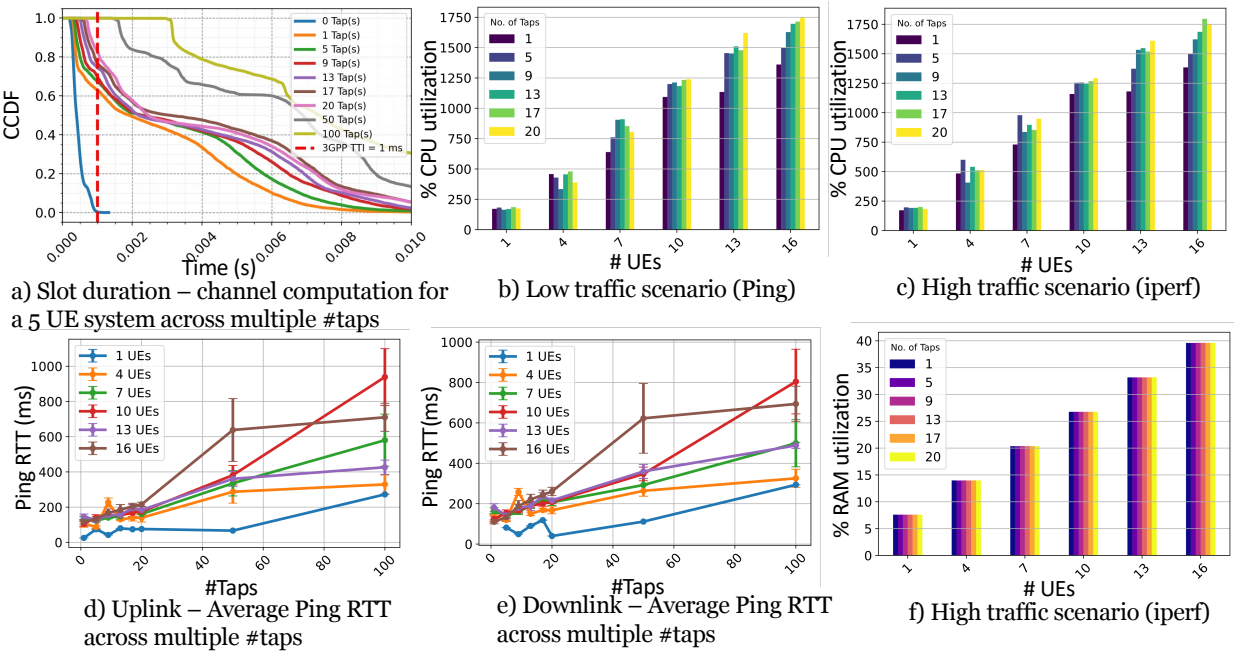
This section highlights the fundamental design principles for a low-cost and high-fidelity digital twin, the limitations of baseline simulators in achieving high-fidelity on a low-cost x86 hardware, presents benchmarks revealing key bottlenecks, and introduces our system design and optimization, including parallelized, sparse Convolution, and CPU Pinning to overcome bottlenecks.

A. Low-Cost, High-Fidelity Digital Twin Design Principles

Designing a practical digital twin for cellular networks requires balancing two competing objectives: *high fidelity*, to faithfully reproduce real-world network behavior, and *low cost*, to ensure accessibility and scalability on commodity hardware. In this work, we distill a set of design principles that define high fidelity in the context of next-generation cellular digital twins and guide the architectural choices behind Tiny-Twin.

Defining High-Fidelity Beyond Abstract Metrics High fidelity in a cellular digital twin extends beyond matching average link-level metrics such as signal-to-noise ratio (SNR), channel quality indicator (CQI), or throughput. Instead, fidelity must be defined in terms of the twin's ability to reproduce *end-to-end behavior under closed-loop operation*, where feedback mechanisms across the physical (PHY), medium access control (MAC), radio link control (RLC), transport, and application layers interact dynamically. Below, we define the requirements for a high-fidelity digital twin as illustrated in Table I.

Channel realism is central to this objective because wireless channels directly drive feedback loops such as link adaptation, hybrid automatic repeat request (HARQ) retransmissions, scheduling decisions, buffer dynamics, and congestion control. Simplified abstractions, such as replaying SNR or CQI traces, fail to capture these interactions and often lead to misleading conclusions. A high-fidelity digital twin must therefore operate on *IQ-level signals*, allowing physical-layer impairments to naturally propagate through the full protocol stack.



a) Slot duration – channel computation for a 5 UE system across multiple #taps

b) Low traffic scenario (Ping)

c) High traffic scenario (iperf)

d) Uplink – Average Ping RTT across multiple #taps

e) Downlink – Average Ping RTT across multiple #taps

f) High traffic scenario (iperf)

Fig. 2: Challenges with Vanilla system

Full-Stack Visibility and Cross-Layer Effects: High-fidelity digital twins must expose the *entire cellular protocol stack*, from baseband signal processing to application-layer performance. While many simulators focus on isolated layers, such separation obscures cross-layer effects that are critical for evaluating modern cellular systems, particularly AI-driven control loops and application-aware scheduling.

Protocol-Level Study involves the detailed examination and analysis of the functional layers above the physical (PHY) layer, including MAC, RLC, and higher transport protocols, within the digital twin's full 5G stack.

Compute Time Fidelity captures the ability of a digital twin to advance through cellular slots *quickly and consistently*, enabling large-scale AI/ML experimentation within practical wall-clock time. We quantify this by measuring the *slot duration*, defined as the wall-clock time to complete all PHY- and MAC-layer computations for a single slot, and analyzing its distribution over long executions.

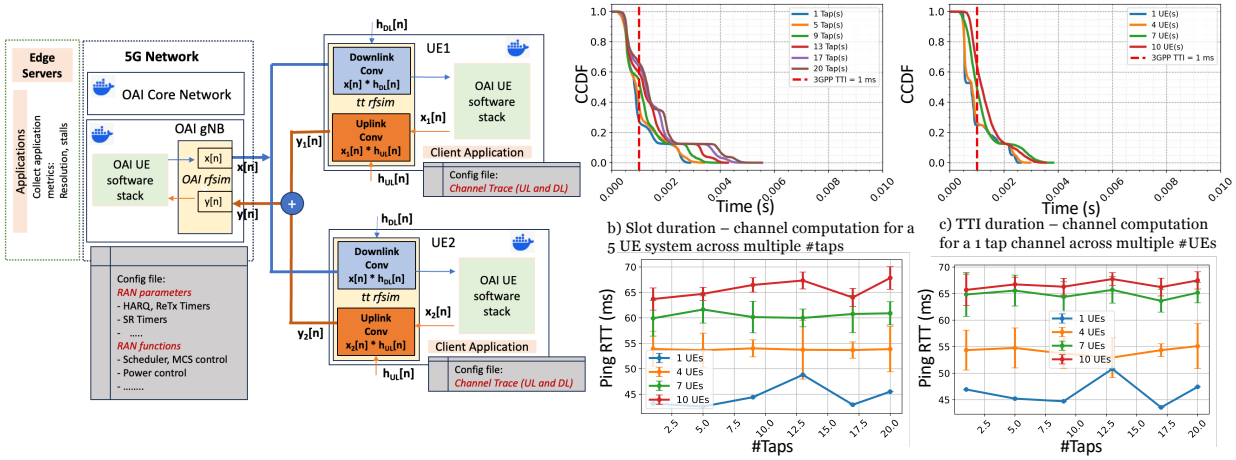
This property is critical for learning-based control systems such as EdgeRIC [1], which require observing and acting over large numbers of consecutive slots during training and inference. Slow or highly variable slot execution makes training impractically time-consuming and can bias learned policies with artifacts of the emulation. By enabling fast and stable slot-level execution on commodity CPUs, Tiny-Twin supports scalable AI-driven RAN experimentation without specialized hardware.

Low-Cost Through CPU-Native Design Achieving the above fidelity requirements typically demands specialized hardware such as field-programmable gate arrays (FPGAs) or graphics processing units (GPUs), significantly increasing cost and deployment complexity. Tiny-Twin adopts a contrasting design philosophy: *CPU-native fidelity*. By restructuring PHY computation, parallelizing per-user equipment (UE) processing, and exploiting sparsity in realistic channel impulse responses, Tiny-Twin achieves fine-grained channel modeling without hardware accelerators. This low-cost design is not merely an implementation detail, but a core principle that enables scalability, reproducibility, and widespread adoption. By operating entirely on commodity x86 CPUs, Tiny-Twin makes high-fidelity cellular experimentation accessible while retaining the essential physical- and protocol-layer realism required for next-generation network co-design.

System Scalability We define System Scalability as the maximum number of concurrent User Equipment (UEs) that the environment can sustain with preserved physical-layer fidelity. Achieving high scalability on commodity hardware, a core goal of CPU-native frameworks like Tiny-Twin, is essential because many emerging research challenges in 5G and NextG, such as resource scheduling, interference management, and mobility handovers, are fundamentally multi-user problems. By enabling the accurate and repeatable emulation of numerous UEs simultaneously, the digital twin transitions from a single-device testing tool into a comprehensive platform for rigorously evaluating novel protocols and algorithms under realistic, heavy network loads.

B. System benchmarks and challenges with the vanilla system

We use the Open Air Interface (OAI) [9], a widely adopted open-source 5G cellular stack, as the foundation for our framework. It supports a complete end-to-end 5G network, including the core, gNB, and UE components, executed in standalone containers, enabling modular deployment and controlled experimentation. OAI includes a built-in simulation mode called `rfsim`, which allows emulated RF signal exchange between the gNB and UEs over a TCP-based interface, it follows a



a) Tiny-twin Framework

b) Slot duration – channel computation for a 5 UE system across multiple #taps

c) TTI duration – channel computation for a 1 tap channel across multiple #UEs

d) Uplink – Average Ping RTT across multiple #taps

e) Downlink – Average Ping RTT across multiple #taps

Fig. 3: Tiny-twin system design and benchmarks

telnet-style client-server model, where IQ samples are transmitted over socket connections between the gNB and each UE. In the default `rfsim` configuration, all channel convolutions are applied at the receiving node. This means, in downlink, the gNB transmits a raw (unconvolved) IQ stream over its server socket. Each UE acts as a telnet client, receives this stream, and applies a linear convolution with its own time-varying channel impulse response to simulate wireless propagation. In uplink, each UE transmits an unconvolved IQ stream to the gNB server. The gNB then performs channel convolution independently for each UE and aggregates the resulting streams. The channel effect is represented by: $y_u(t) = x(t) * h_u(t)$, where $x(t)$ is the transmitted IQ signal, $h_u(t)$ is the UE-specific channel impulse response, and $y_u(t)$ is the received, convolved signal. The convolution operator $*$ denotes linear convolution and varies independently for each receiver. However, in its default configuration, the system applies only a single-tap scalar fading factor, effectively reducing the convolution to a per-symbol scaling operation, rather than a multi-tap filter. We extended this functionality to support multi-tap channel convolution by enabling the `rfsim` module to read time-varying CIR traces from external files, thus apply realistic, per-UE linear filtering. We refer to this setup as our vanilla system.

While vanilla system provides a functional baseline for channel simulation, its architectural design leads to significant performance bottlenecks. As discussed earlier, all channel convolutions are applied at the receiver side, which is particularly inefficient on the uplink path. The gNB receives uplink IQ streams serially from each UE, applies convolution independently for each stream, and only then aggregates them. This serialized, compute-heavy pipeline introduces a critical bottleneck—especially under realistic multi-tap channel conditions.

Figure 2 presents a comprehensive benchmarking study of the vanilla system across multiple performance dimensions. All experiments were performed on a system equipped with an AMD Ryzen 9 7900X processor (24 cores, 5.2 GHz), 32 GB RAM, and 1.8 TB of storage with a total system cost of approximately $1528.00, ensuring a low cost compared to specific hardware. In Figure 2(a), we observe the TTI computation duration for a 5-UE setup under different channel tap configurations. We observe that the 90th percentile compute time reaches 8 ms even with a 10-tap channel. For larger tap values such as 50 or 100, the TTI durations rise significantly, clearly highlighting that convolution costs dominate processing time. These timing violations directly impact end-to-end performance, as seen in Figure 2(d) and 2(e), preventing the system from reaching high fidelity. Here, we observe that the uplink and downlink ping RTT increase sharply with tap count and number of UEs. Beyond 4 UEs and 10 taps, RTT often exceeds several hundred milliseconds, and connections may fail altogether.Figure 2(b) and 2(c) show CPU utilization across low-traffic (ping) and high-traffic (iperf) scenarios, respectively. Utilization increases with the number of UEs, though it remains relatively stable across tap configurations, suggesting that user scaling is the dominant driver of compute load. Figure 2(f) shows RAM utilization under iperf, which also increases steadily with user count but remains within acceptable bounds. These results confirm that, despite not saturating CPU or memory resources, the system suffers from severe latency violations and degraded responsiveness due to inefficient compute allocation. TTI durations frequently exceed several ms, and ping round-trip times (RTTs) scale poorly with tap complexity and number of UEs. The core issue is that the PHY-layer channel convolutions are not effectively parallelized, resulting in underutilization of available system resources. The subsequent subsections detail a restructured system that allows us to mitigate the identified bottlenecks and achieve high fidelity on a low-cost x86 CPU.

C. Tiny-twin Architecture Optimizations

Building upon the insights from Section III-B, we build the Tiny-twin framework to efficiently emulate per-UE wireless scenarios, as illustrated in Figure 3(a). Built atop software-defined network components, Tiny-twin enables end-to-end

execution of real client applications and supports high-fidelity modeling of the signal chain down to the baseband IQ level. Our system implementation involves the following optimizations:

Parallelizing Convolution Computation: To address the primary bottleneck identified in Section III-B—the serial channel convolution at the gNB—we adopt a UE-localized convolution processing approach. We update the OAI *rfsim* mode to shift the entire channel convolution module into individual UE processes. As a result, both uplink and downlink channel convolutions are executed independently within each UE container, thus leveraging per-UE parallelism, distributing the most computationally intensive PHY operation across multiple processes and compute cores.

Sparse Convolution: To reduce the computational cost per convolution operation, we employ a sparse convolution strategy. While each UE maintains the full multi-tap channel profile, convolution is performed only with the top-n dominant taps at each time step. This significantly reduces the number of multiply-accumulate operations while preserving fidelity in modeling the channel's primary effects.

CPU Pinning: To reduce variability and scheduling overheads inherent in shared CPU environments, we employ CPU pinning. This dedicates specific logical CPU cores to key Tiny-twin processes (e.g., two cores per UE container), reducing cache contention and improving processing consistency bu reducing cpu switching time, particularly important for systems aiming for consistent timing behavior.

Performance Benchmarks: We evaluate the performance of the Tiny-twin system incorporating the architectural design and optimizations described above. Figure 3 (b) and (c) present the Transmission Time Interval (TTI) boundary distribution, demonstrating a significant reduction in latency compared to the unoptimized baseline (Figure 2a). The 90^{th} percentile TTI latency is reduced to approximately 2ms, representing a substantial 4x improvement in meeting timing constraints and achieving high fidelity. Furthermore, Figures 3(d) and (e) present the Ping RTT measurements across varying numbers of channel taps. Notably, the RTT remains largely constant regardless of the number of taps. These benchmarks demonstrate that the Tiny-twin system can efficiently support detailed multi-tap channel convolution executed entirely on standard CPU. Our design allows effective emulation with 20 taps per UE while maintaining performance metrics. This represents a highly fine-grained physical layer model achievable without requiring specialized hardware acceleration for the convolution task.

This finding challenges the conventional assumption that hardware acceleration is essential for PHY-intensive operations and establishes the feasible operational limits for channel detail when running a full-stack emulation environment on low-cost CPU-based compute platforms.

IV. Tiny-twin System Evaluations

To evaluate Tiny-twin, we conduct a series of system-level experiments that span channel integration pipelines,

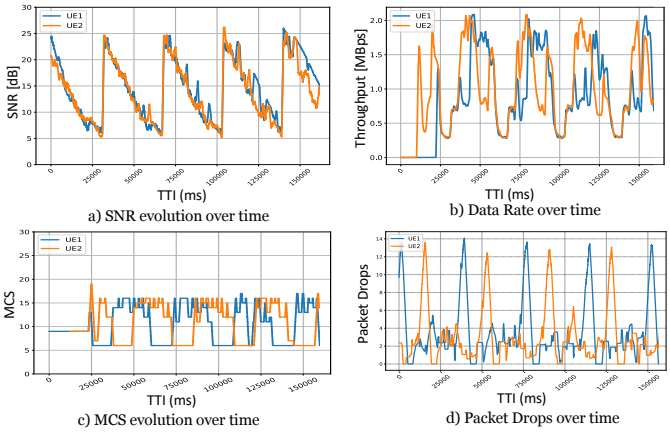


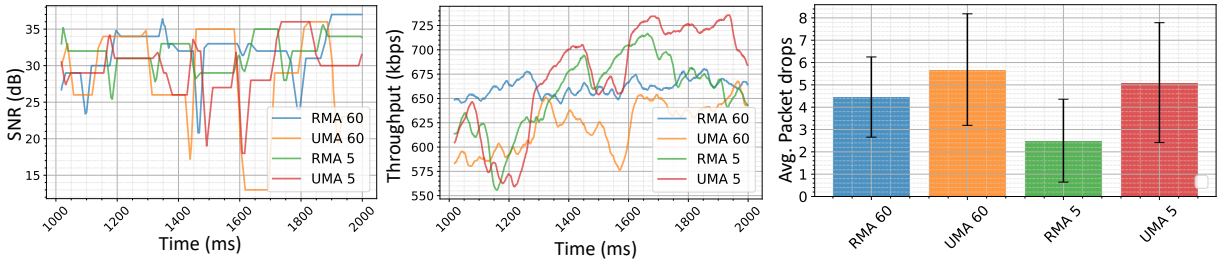
Fig. 4: Sanity Check of channel impelemetation on Tiny-twin

and full-stack responsiveness to dynamic wireless conditions. These evaluations validate our channel representation choices, demonstrate compatibility with diverse propagation environments, and highlight Tiny-twin's ability to capture end-to-end performance impacts often missed by abstract models.

Tiny-twin compatible channel traces Tiny-twin enables experimentation with a wide range of wireless channels, including synthetic, ray-traced, model-based, or real-world over-the-air CIRs, by integrating them as time-varying, multi-tap channel impulse response (CIR) traces. These traces are represented as discrete complex tap values over time, designed to be fed into the per-UE channel convolution modules discussed in Section III. To transform a given continuous CIR into this format, we perform a resampling process to extract uniformly spaced discrete channel taps onto a fixed delay grid for each time step. Crucially, these traces are structured with a temporal resolution of 1 ms, which is typically finer than the channel coherence time. This high-resolution sampling inherently captures the effects of Doppler spread and temporal fading dynamics, ensuring realistic time-varying behavior is reflected during emulation. Tiny-twin thus provides a flexible framework for integrating diverse wireless conditions into a 5G network environment for end-to-end testing.

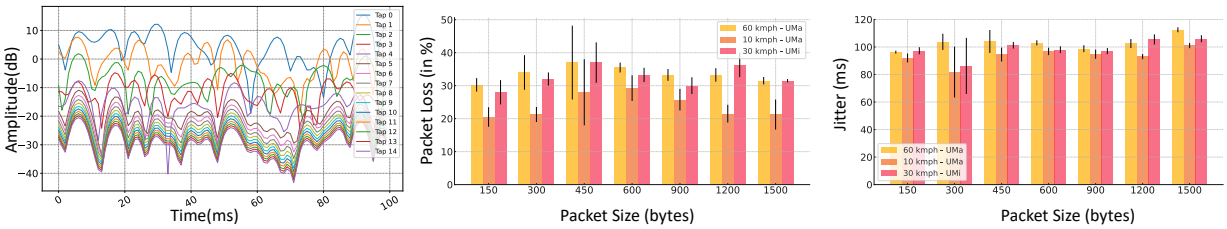
In this study, we demonstrate Tiny-twin's flexibility by injecting three types of channel traces: (i) 3GPP statistical channels based on standardized PDPs for UMa, UMi, and RMa environments, with time variation synthesized via a Jakes-based Doppler model at 3.5 GHz [22]; (ii) ray-traced channels generated using Sionna's ray-tracing module [12], capturing site-specific multipath effects across indoor/outdoor LoS and NLoS scenarios by sampling CIRs along mobility paths; and (iii) real-world channels derived from the ARGOS dataset [23]. We will continue expanding our collection of channel traces, and upon acceptance, we will release all datasets along with the Tiny-twin framework to support reproducible research.

Sanity Check: Microbenchmarks with Synthetic channel trace on tiny-twin To verify the correctness of our channel implementation, we conduct a controlled microbenchmark using a synthetic channel trace where the SNR gradually degrades over time in a periodic fashion. This setup is designed



a) SNR variations over time b) Data Rate variations over time c) Average packet drops at MAC per TTI

Fig. 5: Plug-N-Play Channels on Tiny-twin



a) Extracted Channel Taps from CIR – UMa Model b) Packet Size vs Loss % c) Packet Size vs Jitter

Fig. 6: E2E Application (IRTT) metrics on Tiny-twin

to validate the convolution operations in the physical layer and their propagation through the stack. Figure 4 illustrates the evolution of key performance indicators, such as SNR, MCS, throughput, and packet drops, across two UEs under identical channel dynamics. As expected, the modulation and coding scheme (MCS) adapts responsively to the changes in SNR, showcasing proper link adaptation. Correspondingly, we observe a drop in throughput as SNR decreases, and a rise in packet drops during low-SNR periods. Notably, the peaks in throughput and packet drops are inversely aligned, reaffirming that the system reacts to deteriorating channel conditions. Notably, these drops emerge from full-stack interactions that would be missed using simplified SNR or CQI-based traces. This demonstrates Tiny-twin's utility in capturing the nuanced effects of realistic channel dynamics on end-to-end 5G performance.

Plug-N-Play Channels on Tiny-twin Figure 5 demonstrates Tiny-twin's ability to emulate diverse channel profiles across a range of mobility scenarios. Subfigure (a) shows that high-mobility channels (RMA/UMA 60) exhibit greater SNR fluctuations than their low-mobility counterparts. This variability directly impacts throughput (b), where UMA 5 achieves the highest rates due to stable SNR, while RMA 60 shows pronounced dips. Subfigure (c) reveals that RMA 5 has the lowest MAC-layer packet drops, confirming the link between channel stability and reliability.

Figure 6 further illustrates how these PHY-layer dynamics translate into observable application-layer effects. Subfigure (a) presents CIR taps from a UMA model, revealing rich temporal multipath structure. Subfigures (b) and (c) show that increased mobility and larger packet sizes exacerbate packet loss and jitter, as captured using the IRTT tool [24]. Notably, the UMA 60 km/h trace consistently incurs higher jitter and loss than its lower-mobility counterpart. Together, these results demonstrate that Tiny-twin preserves the end-to-end effects of physical channel variation, including effects often masked in simplified models that rely solely on SNR or CQI traces.

Monitoring framework on Tiny-twin Figure 7 indicates Tiny-twin's monitoring framework for two different scenarios. We use Grafana for the dashboard in combination with Prometheus for data collection [25], [26]. Subfigure (a) shows the impact of asymmetric physical resource block(PRB) allocation on Tiny-Twin using EdgeRIC [1]. Initially, EdgeRIC allocates a greater proportion of Physical Resource Blocks (PRBs) to the green User Equipment (UE) and a smaller proportion to the yellow UE. This asymmetric resource allocation results in an observed higher throughput for the green UE and a corresponding larger buffer occupancy for the yellow UE. This allocation strategy is then reversed in the second phase, where the yellow UE is prioritized to quickly drain its accumulated buffer, while the green UE's throughput is reduced, causing its buffer to fill up. Sub Subfigure (b) specifies the impact of various channel patterns. Initially, the SNR has a repetitive pattern, and both the throughput and buffers follow the same pattern. Then the SNR drops, which leads to an upsurge in buffer. Finally, with good channel quality and a high SNR, the buffer size is reduced.

V. Conclusion and Future Work

Tiny-Twin established a foundational utility by successfully demonstrating that high-fidelity, real-time Physical (PHY) layer emulation is feasible on accessible commodity CPUs within a single-cell, multi-User Equipment (UE) environment. This foundational success provides a direct roadmap for ex-

a) Monitoring the impact of PRB allocation for two UE system using EdgeRIC on Tiny-Twin

b) Monitoring the impact of various channel patterns for two UE system on Tiny-Twin

Fig. 7: The real-time monitoring graphical user-interface for Tiny-twin

panding the system's operational scope, thereby maximizing the platform's research applicability and fully realizing the vision of a large-scale, accessible digital twin for NextG research. Our future efforts are focused on three core axes of architectural growth and research capability.

Architectural Scalability: The single-CPU server evaluation proved the core innovation by enabling high-fidelity PHY emulation on commodity hardware, achieving fast compute for over 10 UEs. Future development will rigorously implement distributed, cluster-based operation by leveraging the inherent isolation of UE processes into separate containers. This essential architectural evolution will eliminate the single-server bottleneck, allowing us to facilitate exhaustive large-scale (100+ user) simulations previously only possible on specialized, proprietary testbeds.

Extending Network Realism: To capture the complexity of real-world cellular deployments, our next objective is to expand the system's realism by integrating support for multi-base station operation and the systematic study of Inter-Cell Interference (ICI). This will directly leverage the planned cluster-based scaling. Future work will extend the channel integration interface to programmatically allow each isolated UE process to receive and sum signals from all neighboring gNB cores across the cluster. This capability will unlock the rigorous testing of interference-sensitive network functions, including sophisticated advanced scheduling protocols under interference and critical mobility management (handover) protocols in a truly representative digital environment.

Ethical concerns: This work does not raise any ethical issues.

References

[1] W.-H. K. et al., "EdgeRIC: Empowering real-time intelligent optimization and control in NextG cellular networks," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024.

[2] H. Y. et al., "ADR-X: ANN-Assisted wireless link rate adaptation for Compute-Constrained embedded gaming devices," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1331–1349.

[3] S. S. R. J. et al., "Mimo-ric: Ran intelligent controller for mimo xapps," ser. ACM MobiCom '24, 2024, p. 2315–2322. [Online]. Available: https://doi.org/10.1145/3636534.3701548

[4] Y. C. et al, "Channel-Aware 5g RAN slicing with customizable schedulers," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023.

[5] U. P. et al., "Towards scalable and cost-effective ran emulation leveraging the public cloud," in *Proceedings of the 26th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '25, 2025, p. 43–48. [Online]. Available: https://doi.org/10.1145/3708468.3711895

[6] W. S. et al., "CellReplay: Towards accurate record-and-replay for cellular networks," in *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*, 2025.

[7] T. M. et al., "Colosseum, the world's largest wireless network emulator," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '21, 2021.

[8] X. L. et al., "6G Digital Twin Networks: From Theory to Practice," 2022.

[9] N. N. et al., "Openairinterface: A flexible platform for 5g research," *SIGCOMM Comput. Commun. Rev.*, 2014.

[10] D. V. et al., "Colosseum as a digital twin: Bridging real-world experimentation and wireless network emulation," 2023.

[11] G. F. R. et al., "The ns-3 network simulator." in *Modeling and Tools for Network Simulation*. Springer, 2010, pp. 15–34.

[12] J. H. et al., "Sionna," 2022, https://nvlabs.github.io/sionna/.

[13] R. N. et al., "Mahimahi: A lightweight toolkit for reproducible web measurement," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, p. 129–130, aug 2014. [Online]. Available: https://doi.org/10.1145/2740070.2631455

[14] T. I. et al., "Open wireless digital twin: End-to-end 5g mobility emulation with openairinterface and ray tracing," *IEEE Access*, 2025.

[15] "srsRAN, Inc." https://www.srslte.com/, 2023.

[16] "ueransim," https://github.com/aligungr/UERANSIM, 2021.

[17] "NVIDIA Aerial," https://developer.nvidia.com/aerial, 2022.

[18] H. M. et al., "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017.

[19] A. L. et al., "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017.

[20] P. G. et al., "ABC: A simple explicit congestion controller for wireless networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, Feb. 2020.

[21] U. G. et al., "Poster: Tiny-twin: A lightweight and verifiable digital twin for nextg cellular networks," in *Proceedings of the 25th International Workshop on Mobile Computing Systems and Applications*, ser. HotMobile '24, 2024, p. 145. [Online]. Available: https://doi.org/10.1145/3638550.3643625

[22] "Study on channel model for frequencies from 0.5 to 100 ghz," 3rd Generation Partnership Project (3GPP), Technical Report ETSI TR 138 901 V14.3.0, 2017.

[23] X. Z. et al., "Directional training for fdd massive mimo," *IEEE Transactions on Wireless Communications*, 2018.

[24] "Pete heist. 2021." https://github.com/heistp/irtt, 2021.

[25] "Grafana: The open observability platform," https://grafana.com.

[26] "Prometheus: An open-source systems monitoring and alerting toolkit," https://prometheus.io.