SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems

Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia
Institute for the Wireless Internet of Things, Northeastern University, Boston, MA, U.S.A.
E-mail: {l.bonati, s.doro, s.basagni, t.melodia}@northeastern.edu

ABSTRACT

The cellular networking ecosystem is being radically transformed by *openness*, *softwarization*, and *virtualization* principles, which will steer NextG networks toward solutions running on "white box" infrastructures. Telco operators will be able to *truly* bring intelligence to the network, dynamically deploying and adapting its elements at run time according to current conditions and traffic demands. Deploying intelligent solutions for softwarized NextG networks, however, requires extensive prototyping and testing procedures, currently largely unavailable. To this aim, this paper introduces SCOPE, an open and softwarized prototyping platform for NextG systems. SCOPE is made up of: (i) A ready-to-use, portable open-source container for instantiating softwarized and programmable cellular network elements (e.g., base stations and users); (ii) an emulation module for diverse real-world deployments, channels and traffic conditions for testing new solutions; (iii) a data collection module for artificial intelligence and machine learning-based applications, and (iv) a set of open APIs for users to control network element functionalities in real time. Researchers can use SCOPE to test and validate NextG solutions over a variety of large-scale scenarios before implementing them on commercial infrastructures. We demonstrate the capabilities of SCOPE and its platform independence by prototyping exemplary cellular solutions in the controlled environment of Colosseum, the world's largest wireless network emulator. We then port these solutions to indoor and outdoor testbeds, namely, to Arena and POWDER, a PAWR platform.

CCS CONCEPTS

• **Networks** → **Mobile networks**; **Network experimentation**.

KEYWORDS

5G, NextG, Network Slicing, Data Collection, Data-driven Control.

ACM Reference Format:
Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. 2021. SCOPE: An Open and Softwarized Prototyping Platform for NextG Systems. In *The 19th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '21), June 24-July 2, 2021, Virtual, WI, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3458864.3466863

MobiSys '21, June 24-July 2, 2021, Virtual, WI, USA

ACM ISBN 978-1-4503-8443-8/21/07…$15.00
https://doi.org/10.1145/3458864.3466863

1 INTRODUCTION

Cellular networks are undergoing a radical transformation whereby traditional "black box" architectures—merely capable of operating under a few possible configurations dictated by equipment manufacturers—are being replaced by programmable infrastructures driven by open software and based on "white box" hardware [8, 9, 11, 32, 44, 48].

The guiding principles of this *Copernical revolution*, urged by increasing traffic demand and by the requirements of future generation ("NextG") cellular systems, are *open programmablity*, *softwarization* and *virtualization* [26, 37, 51]. By embracing these paradigms Telco Operators (TOs) will be able to: (i) Achieve swift control of the network and its functionalities by abstracting and orchestrating them as virtual network functions; (ii) implement differentiated service strategies by deploying network slices each tailored to specific traffic classes and users, and (iii) design and deploy custom algorithms for real-time network optimization for different network conditions and application requirements.

All these new capabilities will considerably facilitate and speed-up the establishment and advancement of NextG systems, in that open software will enable swift definition and deployment of applications over white box infrastructures [27, 37, 41]. However, while developing new software solutions for open architectures might be considerably easier than before, demonstrating their effectiveness, efficiency, reliability and robustness in a host of varying scenarios becomes paramount. Indeed, it is necessary to ensure that nothing jeopardizes the performance, stability, and security of the network and of the services provided to millions of users. It is therefore imperative for TOs to require that all networks algorithms and software components are extensively tested prior to actual deployment on the commercial infrastructure.

As owners of said infrastructure, in principle TOs could test new software solutions on it. Such a choice, however, is neither trivial nor cost-effective as trying out new solutions on the commercial network might induce undesirable behavior and cause unforeseen outages and financial loss. As an alternative, new solutions could be first tested in smaller laboratory setups. These trials, however, can only capture a limited number of Radio Frequency (RF) scenarios and would merely model small-scale deployment configurations, limiting their effectiveness and extent.

Controlled testing at scale is therefore needed. Indeed, solutions for wireless testing at scale are well on the way. Programs such as the Platforms for Advanced Wireless Research (PAWR) by the U.S. National Science Foundation are fostering the creation of large wireless testing sites [39]. *PAWR platforms* like POWDER [10, 19], COSMOS [43] and AERPAW [45], provide controlled access to a variety of real wireless testbeds with a relatively large set of white box hardware and software resources. *Colosseum*—the world's largest

network emulator [14]—also provides researchers with testing at scale, offsetting the site specificity of physical testbeds like the PAWR platforms through a fully controlled, programmable, and observable environment with hardware in the loop. Colosseum is a massive 256-by-256-channel RF emulator with programmable Software-defined Radios (SDRs), capable of emulating full-stack communications with abundant computational capabilities. Virtual worlds can be created as if the radios were operating in an open field, downtown area, forests or a desert, namely, emulating a vast variety of realistic wireless scenarios [47]. Whether real or emulated, using these platforms for designing and prototyping production-ready solutions is not straightforward because of the lack of uniform access and the host of different interfaces to their components.

In this paper, we address the key issue of facilitating platform-independent design by presenting a *Softwarized Cellular Open Prototyping Environment* (SCOPE), a development environment tailored to the design, prototyping and testing of solutions for the softwarized NextG cellular Radio Access Network (RAN). SCOPE consists of a virtualized container and an emulation environment with the following features:

- *Open and portable implementation.* SCOPE includes an open-source implementation of a 3rd Generation Partnership Project (3GPP)-compliant softwarized cellular Base Station (BS). The cellular BS adds novel capabilities to the srsLTE (now renamed "srsRAN") base implementation, such as RAN slicing, and Medium Access Control (MAC) and Physical (PHY)-layer functions. These allow, for instance, to run multiple virtual networks on top of the same physical infrastructure, with the option to select a different scheduling policy in each one of them. Fine-tuning the Modulation and Coding Scheme (MCS) of each mobile subscriber, and implementing downlink power-control schemes are also possible. These functionalities can either be controlled directly through the SCOPE open-source implementation, or through a set of Application Programming Interfaces (APIs), also enabling real-time reconfiguration of softwarized network elements. To make SCOPE platform-independent, we developed a ready-to-deploy Linux Containers (LXC) [13] instance of SCOPE to be deployed on LXC-enabled Linux machines.
- *Data collection capabilities.* SCOPE includes a data collection module for automatically recording the performance of the network. Collected data can be used at run time, e.g., to design adaptive solutions, or offline, e.g., to facilitate the design, training and testing of Machine Learning (ML)/Artificial Intelligence (AI) algorithms.
- *Prototyping RF and traffic scenarios.* To further facilitate NextG experimental research, we developed a set of real-world RF and traffic scenarios—namely, *SCOPE scenarios*—that can be run through the Colosseum emulator. SCOPE users can leverage them to test algorithms at scale under diverse channel conditions (e.g., position, mobility), network deployments (e.g., rural, urban) and traffic.
- *Repeatability, reproducibility and replicability at scale.* All SCOPE scenarios are executed in a deterministic way. This means that while channel coefficients and traffic change over time during an experiment, all experiments executed with the same scenario will experience the very same channel and traffic conditions. In this way, our system can be used to *prototype and fine-tune* solutions by experimenting at scale in repeatable environments before testing them in the field.

We demonstrate the effectiveness of SCOPE as a prototyping environment by: (i) Analyzing the statistical properties of SCOPE cellular scenarios representative of diverse urban environments, and (ii) showcasing exemplary ML and optimization applications. We also provide an example of how SCOPE can be used to port solutions prototyped on Colosseum to real-world heterogeneous testbeds, namely the indoor Arena [4], and the outdoor POWDER [10].

The remainder of this paper is organized as follows. Section 2 presents an overview of how to use SCOPE for wireless experiments. The open-RAN implementation of SCOPE and its capabilities are described in Section 3. The SCOPE emulation environment is presented in Section 4. Section 5 showcases use cases of SCOPE, including ML and optimization applications, and demonstrates the portability of SCOPE to different testbeds. Finally, related works are surveyed in Section 6, and conclusions are drawn in Section 7.

2 EXPERIMENTING WITH SCOPE

The lifetime of an experiment with SCOPE is illustrated in Figure 1. For enhanced clarity, we provide a step-by-step summary of how to use SCOPE on Colosseum. However, SCOPE can be instantiated on any LXC-enabled testbed (Section 5.3).

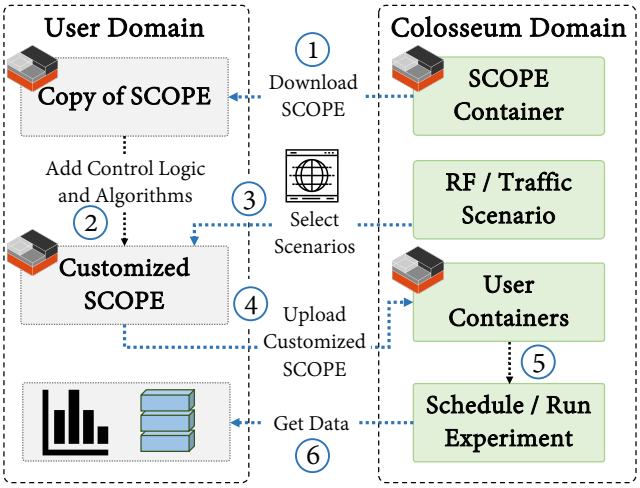


Figure 1: High-level lifetime of SCOPE experiments.

Users first download a copy of the SCOPE container (Section 3) that includes the code to instantiate BSs and User Equipments (UEs), as well as our SCOPE APIs (Section 3.3) for controlling key functions of the softwarized BS at run time (step 1 in the figure). Then, on their local machine (User Domain in the figure), SCOPE users add the desired control logic and algorithms to the container either via the SCOPE APIs or by interfacing directly with its open-source code. In this way, a "customized" instance of SCOPE is created (step 2). Users can now select RF and traffic scenarios for their experiment (Section 4.2) from the set of available SCOPE scenarios, through a dedicated Graphical User Interface (GUI) (step 3). After, researchers can upload their customized container (i.e., SCOPE with user-defined control logic) to Colosseum (step 4). They can then schedule an experiment (through a web GUI) specifying parameters such as the number of nodes (step 5).

In the case of Colosseum, each node, or Standard Radio Node (SRN), consists of a GPU-endowed server connected to one USRP X310. SRNs are fully programmable and serve as virtualized environments running LXC. This makes it possible to use them as either compute-only (e.g., edge or cloud servers) or compute-and-transmit (e.g., UE or BS) nodes.

As soon as the configuration phase is complete, Colosseum deploys the containers on the selected SRNs, starting the experiment. Network run-time metrics are saved in CSV format in the metrics and performance dataset. Users can either query the generated dataset at run time (e.g., using it as a feedback loop) or download it at the end of the experiment to refine the control logic (step 6).

3 SCOPE VIRTUALIZED CONTAINER

We designed SCOPE[1] to facilitate the development of new, adaptive solutions for NextG softwarized cellular systems. A SCOPE container, realized through LXC, is a flexible and ready-to-use prototyping toolkit for users to effortlessly instantiate platform-independent softwarized networks.

The main components of a SCOPE container are: (i) A softwarized cellular protocol stack of BSs, UEs and core network (Section 3.1); (ii) a data collection module for artificial intelligence and machine learning applications (Section 3.2), and (iii) a set of open APIs for users to interface with and control the other two components in real time (Section 3.3).

Users can instantiate SCOPE on any LXC-enabled Linux system (e.g., Colosseum, POWDER and Arena, as discussed in sections 4 and 5.3), and control the BSs and their configuration in real time with just a few lines of code through a set of high-level open APIs, or by modifying directly the open-source code. The APIs are directly interfaced with the data collection module, thus providing SCOPE users with a feedback loop to monitor the performance and state of the network and adapt the control strategy accordingly.

3.1 Softwarized Protocol Stack

The SCOPE BSs, UEs, and core network are based on the open cellular software srsLTE [25]. SCOPE advances the programmability and virtualization capabilities of srsLTE by considerably extending its functionalities to include network slicing, additional MAC-layer scheduling policies, and the ability of fine-tuning PHY-layer parameters at run time. The most relevant functionalities introduced by SCOPE are described in the remaining of this section.

Network Slicing. The SCOPE implemtation of network slicing supports the coexistence of multiple slices tailored to specific traffic classes and UEs on the same shared infrastructure. Our implementation makes it possible to slice the spectrum available at each BS and to dictate the resource allocation for each slice of the network. This is achieved by specifying how many downlink Resource Block Groups (RBGs) (and, thus, Physical Resource Blocks (PRBs)) are allocated to each slice. For example, SCOPE makes it possible to allocate more resources to slices associated with high data rate traffic, while giving less resources to those slices that have lower priority or no strict Quality of Service (QoS) requirements. Selecting the exact portion of the spectrum to allocate to a given slice is also possible. This is achieved through an allocation matrix that specifies which of the available RBGs should be allocated to the slice. Optionally, this allocation matrix can be periodically reloaded at run time to dynamically modify the slice resource allocation.

To facilitate the setup and instantiation of network slices, SCOPE APIs enable the users to specify the association between UEs and slices according to QoS requirements. The APIs also allow to assign different scheduling policies to each slice, and to modify them at run time. This makes it possible to define slice-specific control strategies. This feature is useful to evaluate how specific resource allocations affect different slices and the services they provide.

MAC-layer Scheduling. By default, srsLTE implements a round-robin scheduling algorithm. This limits the degrees of freedom researchers can enjoy in their experiments. SCOPE implements two additional, fairer MAC-layer scheduling algorithms: *Waterfilling* and *proportionally fair*. Both are implemented by first computing the amount of downlink PRBs required by the users based on the data they request, the MCS and the transport block size. Then, resources are granted and allocated according to the users requirements and slice capabilities (e.g., with an allocation proportional to the user request in case of the proportionally fair scheduler). Additional scheduling algorithms can be implemented and plugged in by SCOPE users by modifying the provided resource allocation routines. Scheduling policies can be reconfigured at run time for either the whole network or for selected slices via the SCOPE APIs.

By combining scheduling and slicing capabilities, users can control the performance of each UE and the service levels, and reconfigure the network at run time, if necessary. This effectively enables TOs to offer different levels of service and subscription to UEs.

PHY-layer Capabilities. At the PHY layer SCOPE offers the ability to fine-tune per-user downlink transmission power and MCS (both in uplink and downlink). The former is obtained by selecting the percentage of the maximum power to be assigned to downlink signals of the selected UEs. The latter is an integer number in the range $[0, 28]$ that can be used to change modulation scheme and coding rates of downlink and uplink transmissions on a per-user basis. Current available choices for the modulation are Quadrature Phase Shift Keying (QPSK), 16 Quadrature Amplitude Modulation (QAM) and 64 QAM. The relationship between MCS index, modulations and coding rate is defined by Tables 7.1.7.1-1, 7.1.7.1-1A and Table 8.6.1-1 in [2].

3.2 Data Collection Module

Artificial Intelligence is rapidly reshaping the way we design and operate cellular networks. Unfortunately, one of the issues that has plagued the research community for years is the almost complete lack of large-scale datasets to train ML models. Business and privacy concerns often keep TOs from publicly releasing their datasets, thus slowing down innovation and advancements in the field from the research community. Recently, the release of open-source software for cellular networks has enabled researchers to generate their own datasets. However, this is no easy task as testbeds are usually small-scale and can only model network behavior in limited setups. This makes it hard to train models that can be applied to diverse network deployments and conditions.

[1] SCOPE has been publicly released to the research community: https://www.colosseum.net/colosseum/cellular-software/

SCOPE aims at overcoming these limitations by including a data collection module for the creation of large-scale datasets over a wide number of realistic RF and traffic scenarios. Used on experimental testbeds such as Colosseum, SCOPE provides an effective tool for data collection and experimentation of ML/AI solutions in large-scale cellular networks. The main advantage over existing platforms is that SCOPE combines hardware-in-the-loop execution of cellular procedures with the reconfigurability of the RF channels provided by Colosseum. In this way, while other platforms are representative only of their physical deployment (e.g., location of radios, channel conditions), SCOPE (combined with Colosseum) supports data collection from a virtually infinite number of scenarios and channel conditions. This facilitates the design and testing of ML/AI solutions that are not tied to specific network deployments.

Detailed statistics (e.g., throughput, MCS, buffer size, slice PRBs) on the performance of each BS and UE are periodically logged by the SCOPE data collection module and stored in a CSV-formatted dataset. As we will discuss in Section 3.3, data generated by this module can be accessed via the SCOPE APIs at run time, allowing the implementation of closed-loop optimization and data-driven control routines.

3.3 Open APIs

A sample of relevant SCOPE Python APIs is shown in Table 1. They facilitate the run-time reconfiguration of network slicing, scheduling and PHY-layer parameters (Section 3.1), among others. They also allow SCOPE users to query the metrics and performance dataset (Section 3.2).

Table 1: Sample of relevant SCOPE Python APIs.

Function	Description
`enable_slicing`	Enables/disables network slicing globally
`set_slice_users`	Set UE-slice associations
`get_slice_users`	Get UE-slice associations
`set_slice_resources`	Sets resources allocated to the slice
`set_slice_scheduling`	Sets slice scheduling policy
`set_slice`	Sets slice scheduling and/or resources
`set_scheduling`	Sets global scheduling of the BS
`set_mcs`	Sets UE downlink/uplink MCS
`set_power`	Sets scaling factor for UE downlink signals
`read_metrics`	Reads metrics from dataset
`get_metric`	Returns value of a specific metric

For instance, network slicing can be enabled/disabled globally for the whole network (*enable_slicing*). If enabled, SCOPE APIs allow to set/get the UEs associated to each slice of the network (*set/get_slice_users*). The behavior of each slice can be configured in terms of allocated PRBs—which reflects the portion of spectrum available to the slice—and scheduling policy (*set_slice_resources*, *set_slice_scheduling*). Additionally, PRBs and scheduling of each slice can also be jointly set (*set_slice*). Besides allowing users to set the scheduling policy for each slice, SCOPE APIs also allow to set the global scheduling policy for the whole network (*set_scheduling*).

The PHY-layer configuration can be tuned by setting the downlink/uplink MCS of selected UEs, e.g., all the UEs of a certain slice (*set_mcs*), which directly impacts on the signal modulation and coding rate (see Section 3.1 and [2]). Additionally, the power level of signals for selected UEs can be tuned as well (*set_power*). Finally, SCOPE APIs allow interaction with the dataset generated by the data collection module of the BSs (Section 3.2). Specifically, they allow to read and get specific metrics values for any target time window (*read_metrics* and *get_metric*).

Listing 1 shows an example where SCOPE APIs are used to dynamically assign resources to each slice according to run-time performance read from the dataset.

```
1.  import scope_api as sc, time

2.  while experiment_running:
3.    wnd_metrics = sc.read_metrics(time_window)

4.    for slice_id, slice_metrics in wnd_metrics.items():
5.      slice_users = slice_metrics['ue']
6.      slice_rbg = slice_metrics['rbg']
7.      sc.set_mcs(slice_users, mcs_level, 'dl')

8.      if slice_metrics['buffer'] > threshold:
9.        sc.set_slice(slice_id, 'proportionally', slice_rbg + 2)
10.     else:
11.       sc.set_slice(slice_id, 'round-robin', slice_rbg - 2)

12.   time.sleep(timeout)
```

Listing 1: Example of SCOPE APIs.

While a SCOPE experiment is running (line 2), the user calls the SCOPE *read_metrics* API to read the performance metrics of each slice from the generated dataset and for the specified *time_window* (line 3). Metrics are stored in a dictionary (*wnd_metrics*) that can be accessed iteratively (line 4). Users can call the SCOPE *set_mcs* API to set a specific *mcs_level* for all UEs of the slice (line 7). Note that SCOPE APIs also enable the selection of MCS levels for each UE.

Listing 1 also shows an example of how SCOPE users can implement control logic policies. For example, users can change scheduling and network slicing policies when the metrics reported in SCOPE dataset meet certain conditions. For instance, if the size of the transmission buffer (*slice_metrics['buffer']*) is above/below a threshold (lines 8 and 10), the resources of each slice, e.g., scheduling policy and allocated RBGs, can be changed accordingly (*set_slice*, lines 9 and 11). Finally, the algorithm waits for a *timeout* (line 12) before reading the metrics from the dataset again.

We note that the SCOPE APIs are a tool provided to users to facilitate the reconfiguration of a variety of network parameters. They are not the only way to access lower-layer information and capabilities. Developers can still customize and modify the open-source cellular code and access parameters not currently available via the SCOPE APIs.

4 SCOPE EMULATION ENVIRONMENT

In this section, we detail SCOPE cellular emulation environment, executed through Colosseum. A system overview is shown in Figure 2 where we distinguish between two main parts: (i) *User domain*, and (ii) *Colosseum domain*.

User Domain. The user domain, running on the user local machine, is where researchers download the SCOPE container and use its APIs to implement their custom algorithms (Section 3). Here, SCOPE users interface with Colosseum to run their experiments

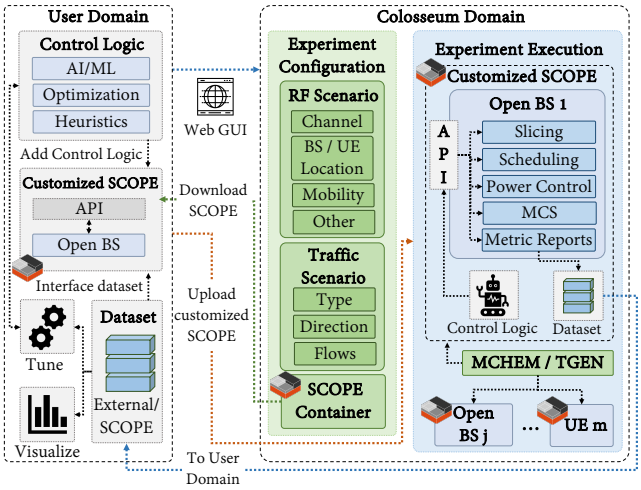


Figure 2: SCOPE emulation environment.

and test their devised solutions in various emulation setups. Specifically, users access Colosseum via a dedicated web GUI [14], select the SCOPE scenarios they want to run to evaluate their solutions, and eventually visualize and process the obtained results. Indeed, SCOPE allows to collect large amounts of data (e.g., throughput, transmission queue status, Channel Quality Information (CQI), PRBs allocated to each network slice, to name a few) that can be leveraged to train ML/AI models, or to design novel optimization and heuristic solutions for cellular applications.

Colosseum Domain. The operations executed on Colosseum can be divided into two different phases: (i) The *experiment configuration*, and (ii) the *experiment execution*.

During the experiment configuration, users setup the experiment to run on Colosseum. They select the desired RF and traffic scenarios and the duration of the experiment. Among others, they can specify which nodes act as BSs and which as UEs. After creating a customized instance of SCOPE with user-defined control logic, users can upload it on Colosseum. They now enter the experiment execution phase, where the specified experiments are actually run.

Experiments can be scheduled and started through a web GUI. Once the experiment begins, the user-customized SCOPE container is automatically deployed on the corresponding SRNs, and SCOPE RF and traffic scenarios of choice are executed by Colosseum Massive Channel Emulator (MCHEM) and Traffic Generator (TGEN) (Section 4.1). User-defined control logic, e.g., any script using SCOPE APIs to implement custom ML and/or optimization algorithms, is also started at run time and the dataset module acts as a feedback loop on the network performance and can be used to evaluate the impact of control decisions on each node.

When the experiment is over, the scenario execution ends and all the results collected during the experiment—stored in the metrics and performance dataset—are transferred to the user directory on Colosseum. This makes it possible to process results and save the dataset for future applications.

4.1 Creating Cellular Scenarios

This section provides details on the inner workings of SCOPE cellular scenarios. An overview of ready-to-use sample scenarios will be given in Section 4.2. Each scenario (Figure 3) consists of two macroblocks: The *RF scenario* and the *traffic scenario*.

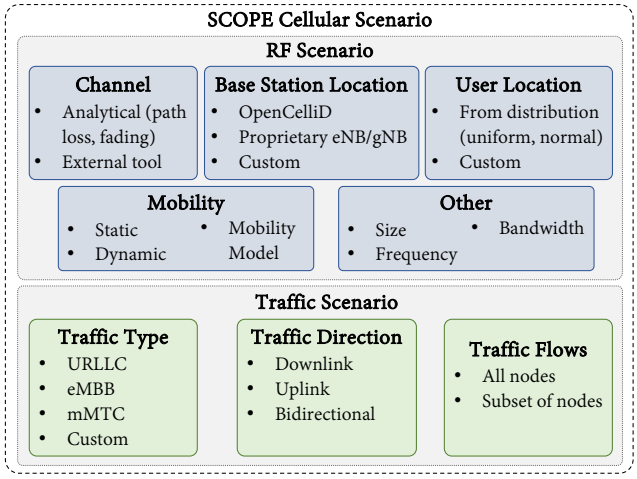


Figure 3: SCOPE cellular scenario components.

The ***RF Scenario*** specifies the *channel conditions* that each node experiences in the experiment. For each SRN, the scenario defines channel impulse responses that model path loss, fading and multipath effects. These channel coefficients are updated every millisecond and can be generated in different ways. Colosseum supports scenarios with channel coefficients generated via analytical models, ray tracing software or obtained via real-world measurements/channel sounders. Coefficients are then fed to MCHEM, which applies the corresponding channel taps to signals to/from each SRN.

To better understand how RF channels are emulated in Colosseum, Figure 4 depicts a high-level overview of the MCHEM channel emulation procedures.

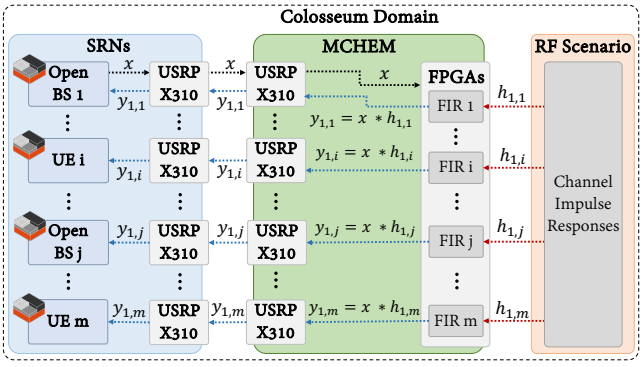
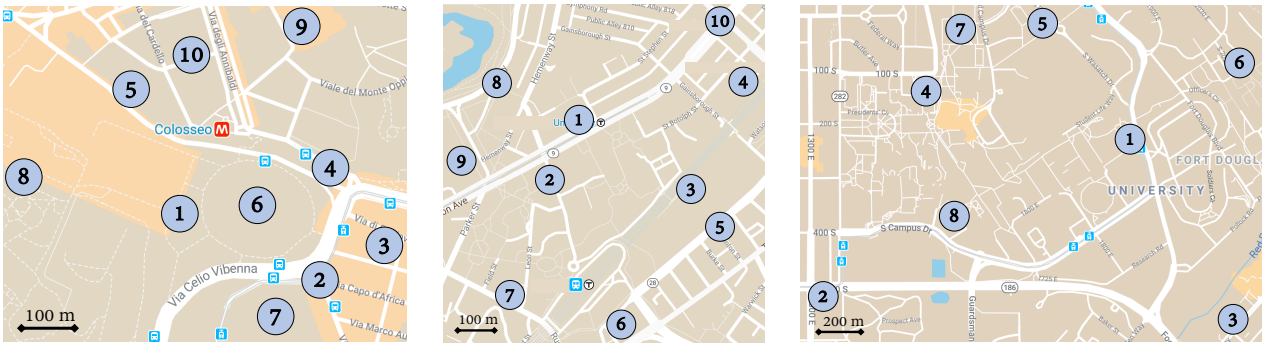


Figure 4: Emulation of RF scenarios via MCHEM.

When a signal x is generated by an SRN (e.g., open BS 1 in the figure), this gets transmitted by the USRP X310 of the SRN and received by the USRP X310 of MCHEM, which is connected to it via low-noise and low-loss SMA cables (Figure 4). The latter sends x to MCHEM Field Programmable Gate Arrays (FPGAs), where signal processing operations occur. These FPGAs are fed with the channel impulse responses corresponding to the RF scenario in use.



(a) Rome scenario (b) Boston scenario (c) POWDER scenario

Figure 5: Large-scale cellular scenario maps. The numbered blue circles mark the locations of the BSs on the map.

These impulse responses, namely $h_{k,m}$, where $k, m \in [1, M]$ are the transmitting and receiving nodes, and M is the number of nodes in the experiment, are convoluted to x by the FPGAs via Finite Impulse Response (FIR) filters. The resulting signals $y_{k,m} = x * h_{k,m}$ are then transmitted to the corresponding SRNs. It is worth mentioning that MCHEM forwards the convoluted signals $y_{k,m}$ to multiple SRNs and not just the intended receiver (Figure 4). In this way, Colosseum makes it possible to emulate interference among every node, akin to what would happen in an over-the-air cellular deployment.

Since channel conditions vary based on the *location* of the nodes, RF scenarios specify the position of each node for each instant of time (Figure 3). This makes it possible to run experiments on different cellular deployment configurations that mimic real-world cellular topologies. The location of the BSs can be specified via GPS coordinates, which can be either generated randomly, through statistical models, or derived from open-source 4G/5G datasets such as OpenCelliD [49]. Similarly, the location of UEs can be specified via GPS coordinates according to diverse probability distributions (e.g., uniform, normal), or historical data from TOs, if available.

Additionally, scenarios can specify *user mobility*, which in SCOPE can be static, i.e., the users do not move for the entire duration of the experiment, or dynamic, in which case the mobility model and speed can also be specified. Finally, RF scenarios allow to select the size of the emulated environment and RF frequency.

The ***Traffic Scenario*** specifies and configures the traffic flows among BSs and UEs (Figure 3). Traffic scenarios are handled by the Colosseum TGEN, which is built on top of Multi-Generator (MGEN), an open-source software that generates and controls realistic TCP/UDP traffic [50]. MGEN supports a variety of different classes of traffic with diverse QoS requirements, probability distributions, data rates and types of service. In this way, MGEN can be used to generate Ultra Reliable and Low Latency Communication (URLLC), enhanced Mobile Broadband (eMBB) and Massive Machine Type Communications (mMTC) traffic, which SCOPE can assign to different slices of the network and control independently (Section 3).

4.2 Sample Scenarios

In this section, we give an overview of sample SCOPE cellular scenarios in three different urban setups: (i) Rome, Italy; (ii) Boston, U.S., and (iii) Salt Lake City, U.S. (POWDER scenario). SCOPE cellular scenarios have been thoroughly designed leveraging the toolchain developed and validated by DARPA for the Colosseum network emulator. The very same toolchain has been used to create the scenarios of the Spectrum Collaboration Challenge, a $2M competition to foster collaboration in the wireless spectrum [18]. For the Rome and Boston scenarios, the locations of the BSs reflect real cell tower deployments extracted from the OpenCelliD database [49]. In the POWDER scenario, they mirror those of the rooftop BSs deployed in the Salt Lake City platform [40]. Each scenario includes from 8 to 10 BSs and up to 40 UEs, whose location and mobility can be selected by SCOPE users when setting up an experiment. We considered the following UE distribution configurations: (i) *Close* (UEs are randomly distributed within 20 m from the serving BS); (ii) *medium* (50 m), and (iii) *far* (100 m). We implemented three different mobility configurations: (i) *Static*, in which UEs do not move for the entire duration of the experiment; (ii) *moderate*, in which they move at an average speed of 3 m/s, and (iii) *fast*, in which their average speed is 5 m/s. In all the cases with mobility, UEs follow a random waypoint mobility model.

A graphical overview of SCOPE sample cellular scenarios is given in Figure 5, in which the numbered blue circles represent the locations of the cellular BSs on the map:

- The *Rome scenario* captures the dynamics of the city center of Rome, Italy. A total of 50 nodes are involved: 10 BSs and 40 UEs. This is the most dense scenario we developed in Colosseum and it covers an area of 0.5 km^2 (Figure 5a).
- The *Boston scenario* captures the dynamics of downtown Boston, U.S. A total of 50 nodes are involved: 10 BSs and 40 users. This scenario covers an area of 0.95 km^2 (Figure 5b).
- The *POWDER scenario* mirrors the setup of the rooftop BSs deployed in the POWDER platform in Salt Lake City, U.S. [40]. A total of 40 nodes are involved: 8 BSs and 32 UEs. This scenario is the sparsest with an area of 3.6 km^2 (Figure 5c).

Along with the RF scenario, we designed relevant traffic scenarios (Section 4.1). Given the ever-increasing popularity of video streaming platforms, we leveraged TGEN and MGEN to generate dedicated traffic scenarios that model uplink and downlink video streaming traffic flows among UEs and BSs.

4.2.1 Scenario Analysis. We now show results obtained by executing SCOPE in the above scenarios focusing on (i) providing

insights on how different topologies, distributions and mobility patterns affect network performance, and (ii) showcasing the statistical properties of SCOPE experiments.

To better highlight the major differences among the above scenarios, we consider the default SCOPE configuration where all UEs generate the same type of traffic, belong to the same slice, and are served via a round-robin scheduling algorithm (see Section 5 for more use cases). For each scenario, we measured downlink/uplink throughput and spectral efficiency for different distances and mobility configurations.

Overview of a SCOPE experiment. To give a better understanding of what running SCOPE experiments looks like, we first show results pertaining single experiment runs (one for each distance among BSs and UEs) in the Rome scenario with static UEs. The considered distances are: *Close*, *medium*, and *far*, as described earlier in this section.

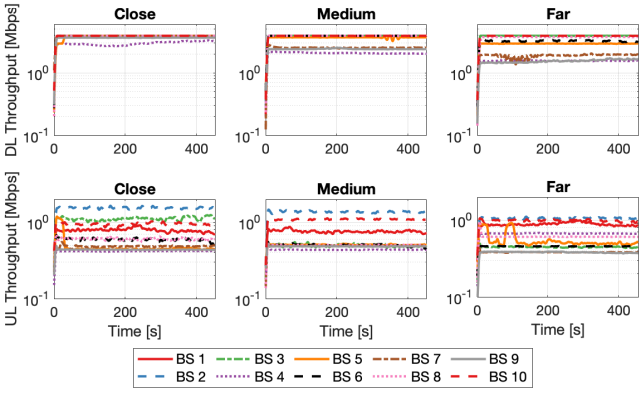


Figure 6: Downlink (DL) and uplink (UL) throughput in the Rome static scenario.

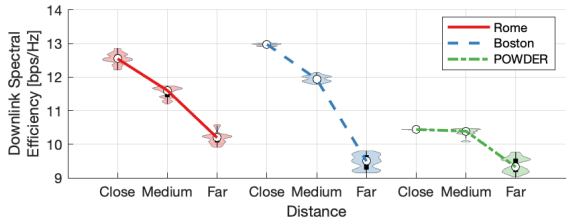
The measured downlink and uplink throughput is shown in Figure 6. As the distance among BSs and UEs increases, the gap among the downlink throughput of different BSs (top part of Figure 6) becomes larger. This is due to channel artifacts, e.g., path loss and fading, which become more significant at greater distances among UEs and BSs. Similarly, the uplink throughput (bottom part of Figure 6) decreases as the distance between BSs and UEs increases.

Statistical analysis. To illustrate the statistical properties of SCOPE experiments, and demonstrate how results do not vary significantly across multiple experiment repetitions. we performed more than 60 repetitions (> 10 hours) varying the distance among BSs and UEs, and the mobility of UEs.

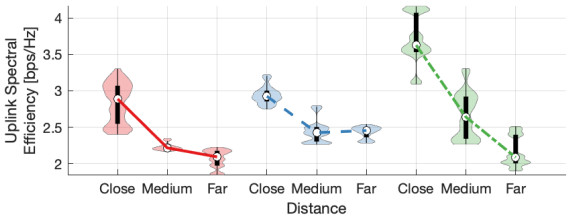
To present our results, we resort to violin plots, which show both the Probability Density Function (PDF) and distribution of the data measurements across several realizations (shaded areas in the figures), as well as their median (white dots). The black boxes show the 95% confidence intervals.

• *Static case.* Results for static cellular scenarios are shown in Figure 7. In this case, UEs do not move but they are placed at different distances from the BSs (*close*, *medium* and *far*).

Metrics for downlink and uplink performance are shown in Figures 7a and 7b, respectively. As expected, the spectral efficiency



(a) Downlink spectral efficiency

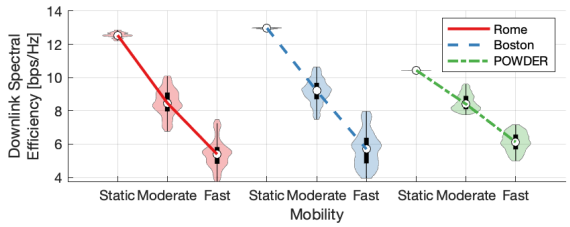


(b) Uplink spectral efficiency

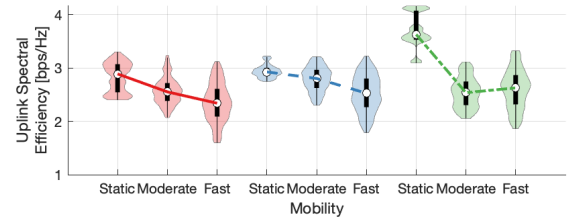
Figure 7: Spectral efficiency in the static scenarios.

decreases for both downlink and uplink as the UEs are placed further away from the BSs. However, despite few outliers, in all cases the data are distributed around the median (white dots in the figures), and exhibit tight 95% confidence intervals (black boxes).

• *Dynamic case.* Figure 8 shows the above metrics in the case of mobile nodes (*static*, *moderate*, and *fast* UE mobility).



(a) Downlink spectral efficiency



(b) Uplink spectral efficiency

Figure 8: Spectral efficiency in the dynamic scenarios.

The downlink performance is shown in Figure 8a; Figure 8b shows the uplink performance. Since the higher the speed, the more likely UEs are to move away from the serving BS, we observe a drop in performance as the speed increases. However, as for the static case (Figure 7), data are distributed around the median (white dots) with small 95% confidence intervals (black boxes). This demonstrates that several realizations of the same experiments achieve comparable results.

5 SCOPE USE CASES

In this section, we discuss relevant SCOPE use cases and examples of interest to the research community ranging from ML (Section 5.1) to traditional optimization control techniques (Section 5.2). We also demonstrate how SCOPE can be ported seamlessly on heterogeneous testbeds such as Arena [4] and POWDER [10] (Section 5.3).

5.1 Machine Learning

Our first use case is that of a researcher utilizing SCOPE to implement ML-based control algorithms for cellular networks (see Figure 9). In the user domain, ML/AI algorithms are designed and

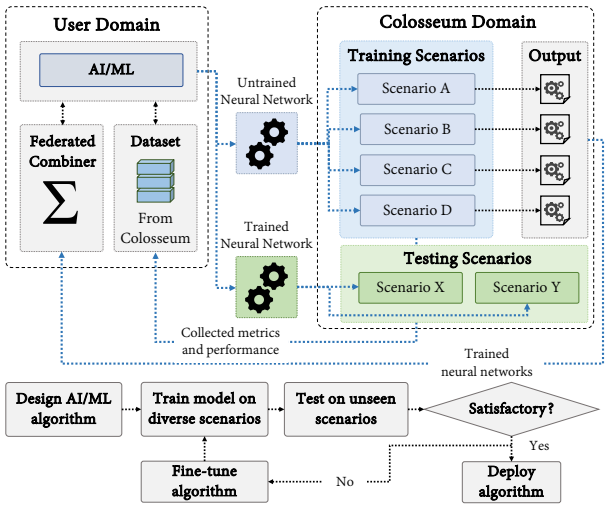


Figure 9: Machine learning use case.

interfaced with SCOPE through its open APIs (Section 3). The resulting version of the SCOPE container, which includes the user-defined control logic, is then transferred to Colosseum, where it is used to run experiments. Being able to run experiments on multiple scenarios enables a variety of ML applications. For example, researchers can train multiple copies of the same *neural network* on a subset of the available scenarios (*training scenarios* in Figure 9) leveraging the *metrics and performance dataset* generated by SCOPE. Then, they can use federated learning techniques to combine weights and develop more general models [53].

After the training is completed, the neural networks can be tested on a completely different and unseen set of scenarios (*testing scenarios* in the figure). This makes it possible to validate the generalization capabilities of the trained model and eventually fine-tune its weights, if necessary. Finally, after the devised algorithms work as expected, the model can be exported from SCOPE and deployed on production cellular networks or any other LXC-enabled testbed.

SCOPE for Deep Reinforcement Learning. To provide a practical example of how SCOPE can be used to prototype machine learning algorithms, we implemented a simple yet effective Deep Reinforcement Learning (DRL) algorithm [31, 35] using SCOPE container and scenarios. As shown in Figure 10, we trained a Deep Q-Network (DQN) agent—a well-established DRL solution for problems with discrete actions [34]—that reads periodically the metrics stored in the BS dataset and adapts slicing and scheduling strategies at run time to maximize the network throughput. In this case, each BS of the network hosts a dedicated DRL agent.

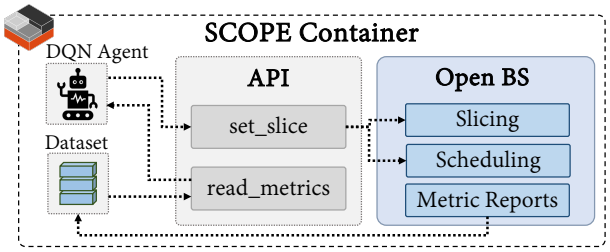


Figure 10: Deep Q-Network agent.

We considered a network with two different slices. The agent is required to select how many PRBs to allocate to each slice and which scheduling algorithm should be used to serve the UEs of the slice. The actions taken by the agent are then enforced via SCOPE (Figure 10), which reconfigures the BSs in real time. The state of the agent is generated by periodically reading the dataset entries corresponding to the most recent 10 s of the experiment (this frequency can be tuned as appropriate). These are then fed to the encoding portion of an autoencoder trained to create a latent representation (and thus with lower dimension) of the state of the system [52]. Due to space limitations, and since this is not the focus of this paper, we refrain from providing a detailed description of the DQN and autoencoder implementations.

We consider two different control configurations. In the first one, the agent makes decisions on the scheduling policy of each slice only. In the second, the agent controls both slicing and scheduling policies. First, we report the downlink throughput measured during the training of the agent (Figure 11) under the first configuration. Then, we show the number of packets to be transmitted to the UEs, PRBs and scheduling policy of each slice, and spectral efficiency for the two configurations (Figures 12 and 13). Finally, we compare the trained DQN agent with the case in which static scheduling policies are adopted by the network (Figure 14).

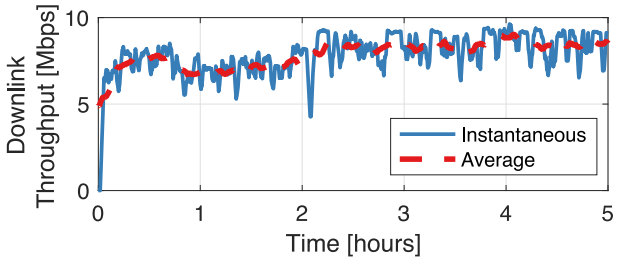
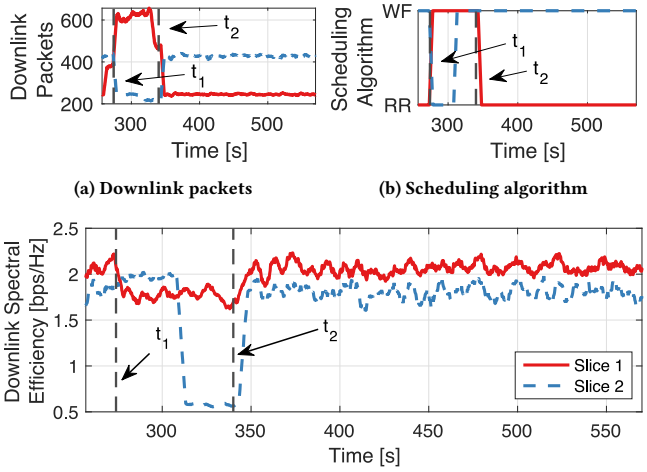


Figure 11: Downlink throughput of the DQN agent as a function of the training time with scheduling decisions only.

Recall that, at the beginning of the training, the agent is initialized with random weights. As shown in Figure 11, this means that actions computed in the first few epochs are taken at random and are generally sub-optimal. As the training goes on, the agent learns

how to effectively select strategies that achieve higher throughput values. After 2 hours of training, for instance, the agent is already capable of selecting actions that result in improved performance.

Figure 12 shows the results obtained when we test the DQN agent previously trained to make scheduling decisions for each slice only (i.e., no decisions on resource allocation).



(a) Downlink packets (b) Scheduling algorithm

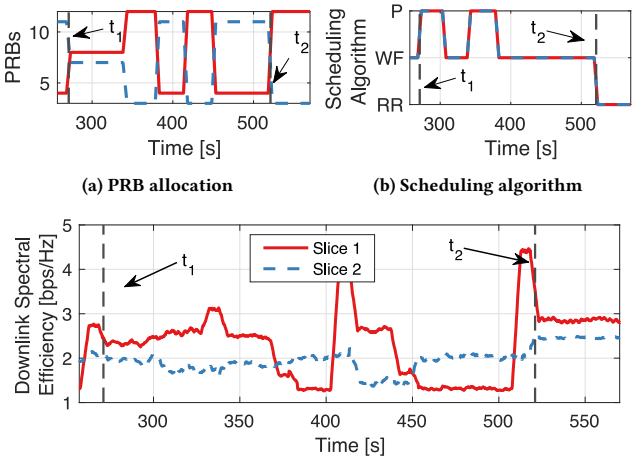
(c) Downlink spectral efficiency

Figure 12: Machine learning use case with scheduling decision policies.

At the beginning of the testing experiment, the trained DQN agent selects the round-robin (RR in the figure) and waterfilling (WF) scheduling algorithms for slices 1 and 2, respectively. As the number of packets for the UEs of slice 1 increases (solid red line in Figure 12a), the agent changes the scheduling policy of the slice from round-robin to a fairer waterfilling (t_1, Figure 12b). This action allows the network to maintain a good level of spectral efficiency (Figure 12c). On the other hand, the scheduling algorithm of slice 2, which sees a 20 s decrease in the packet arrivals for the UEs (dashed blue line in Figure 12a) is changed from waterfilling to round-robin, and changed back to waterfilling shortly after. Finally, as the burst of packets for the UEs of slice 1 ends, the scheduling policy of the slice is changed back to round-robin (t_2). We observe that the brief decrease in the spectral efficiency of slice 2 (Figure 12c) corresponds to a short time window with fewer packet arrivals (see Figure 12a).

Results obtained by testing on Colosseum the DQN agent trained to control both scheduling and slicing policies are shown in Figure 13. The DQN agent makes decisions on both PRB allocation, which affects the resources of the slices (Figure 13a), and scheduling policy (Figure 13b) of the two slices. Both decisions reflect on the downlink spectral efficiency (Figure 13c), which the agent tries to balance between the two slices.

When the testing experiment starts, both slices are served via the waterfilling scheduling algorithm (WF in Figure 13b). Then, at time instant t_1, the DQN agent observes the state of the network and modifies the resource allocation of each slice. As a result, the PRBs of slice 1 are increased, while those of slice 2 decreased (Figure 13a). Additionally, the agent selects the proportionally fair scheduling algorithm for both slices (P in Figure 13b). On the one



(a) PRB allocation (b) Scheduling algorithm

(c) Downlink spectral efficiency

Figure 13: Machine learning use case with scheduling and slicing decision policies.

hand, this causes the downlink spectral efficiency of slice 1 to increase (Figure 13c). On the other hand, slice 2 achieves the same spectral efficiency utilizing fewer resources. Similar decisions are made throughout the experiment to balance the spectral efficiency of the two slices. Finally, at time t_2, the DQN reallocates the PRBs of the two slices (11 PRBs for slice 1, and 4 for slice 2, recall that the BS uses 15 PRBs), and sets their scheduling policies to round-robin (RR in Figure 13b). This results in the two slices achieving similar, i.e., fair, levels of spectral efficiency.

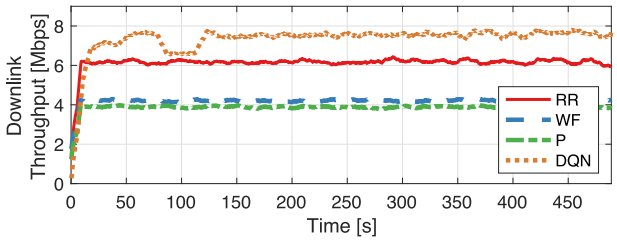


Figure 14: Downlink throughput of the DQN agent with scheduling decisions vs. fixed scheduling policies.

Finally, Figure 14 compares the network performance obtained running the DQN agent with scheduling decisions with that of the network running round-robin, waterfilling and proportionally fair scheduling policies. We notice that the sequential decision making of the DQN agent allows it to improve the overall downlink throughput with respect to fixed scheduling policies, thus adapting to the varying channel and traffic conditions.

5.2 Optimization and Heuristics

A major issue with many optimization algorithms and heuristics is that they often rely upon analytical models, assumptions and approximations that do not accurately reflect the real network behavior. SCOPE can help researchers refine these models and assessing their accuracy in real-world applications. A possible use case is shown in Figure 15. Similar to the ML use case (Section 5.1), SCOPE

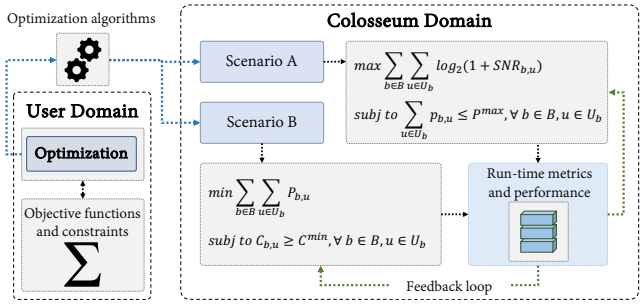


Figure 15: Optimization use case.

can be customized to include optimization- and heuristic-based control logic. This control logic can leverage SCOPE to interface with the BSs and control their configuration at run time. The customized container can then be uploaded to Colosseum and used to test different optimization/heuristic objectives. At the experiment run time, the user algorithms can leverage SCOPE metrics and performance dataset to have a feedback on the behavior of the network. Decisions can be made on the policies of the network (e.g., scheduling and slicing), with subsequent run-time reconfiguration of the BSs. Finally, when the user policies reach a satisfactory behavior, they can be reliably deployed on commercial cellular networks.

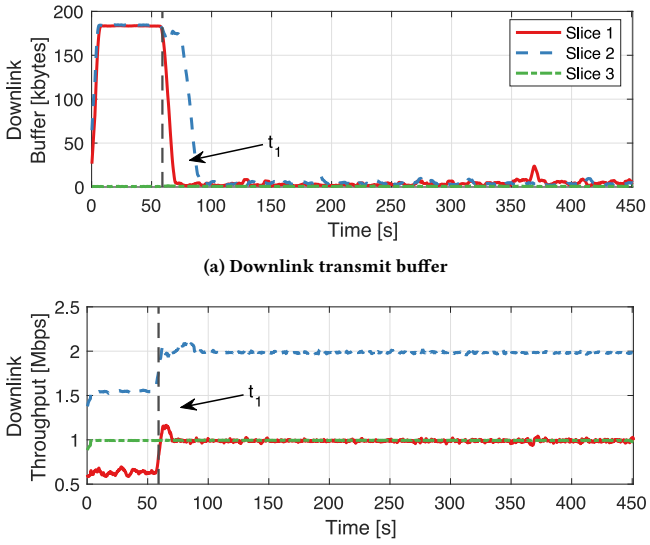
For the sake of illustration, in Figure 16 we show the downlink buffer size and throughput obtained by implementing a heuristic algorithm on SCOPE. We consider the case in which BSs serve three slices with different classes of traffic, UEs and QoS requirements: (1) One UE generating URLLC traffic with low-latency requirements; (2) two UEs with eMBB traffic with high data rates, and (3) one UE generating mMTC traffic with loose latency and data rate requirements.

The heuristic algorithm periodically reads data from SCOPE dataset and tunes the slice resources accordingly. Slice 1 (URLLC) is given additional resources when the BS has more packets to transmit to the UEs (i.e., when the size of the transmission buffer increases) to enable prompt communications. On the other hand, slice 2 (eMBB) is allocated more resources when the available bandwidth of the slice saturates. Finally, due to its loose performance requirements, slice 3 (mMTC) is served with fixed scheduling policies and is allocated two PRBs for the entire duration of the experiment.

As soon as the experiment starts, there is a surge in the traffic of slices 1 and 2, which causes an increase in the downlink buffer (Figure 16a). Following the above policy, resources of both slices are increased at time t_1 to prevent congestion of the transmission buffers. Additionally, the scheduling policy of slice 2 is also changed from round-robin to a fairer waterfilling. Because of this, at time t_1 we observe a prompt decrease in the buffer size of both slices and an increase of throughput (Figures 16a and 16b, respectively).

5.3 SCOPE Portability

In this section we illustrate how SCOPE can be ported to different testbeds. We start by prototyping an instance of SCOPE with 8 BSs serving 32 UEs on Colosseum. We then port it to Arena [4]—an indoor office testbed—and POWDER [10]—an outdoor large-scale platform part of the PAWR program [39]. In both testbeds the BS serves 2 UEs located at an average distance of 4.5 m for Arena, and



(a) Downlink transmit buffer

(b) Downlink throughput

Figure 16: Optimization use case for different classes of traffic: URLLC (slice 1), eMBB (slice 2), and mMTC (slice 3).

345 m for POWDER. In Arena the antennas of all devices (USRPs X310) are at the same height (hung off the ceiling); in POWDER the BS (USRP X310) is located on the rooftop of a 30 m-tall building and serves ground-level UEs (USRPs B210).

Since SCOPE uses virtualized LXC containers, porting it to different testbeds involves transferring the container to the target testbed, and configuring LXC to bridge the interfaces of the host machine to the container [12]. As in SCOPE the actual communication with the hardware radio is left to srsLTE, different SDRs can be used by installing the required drivers [46].

In this experiment, we measure the average downlink spectral efficiency of the BSs. Results are shown in Figure 17.

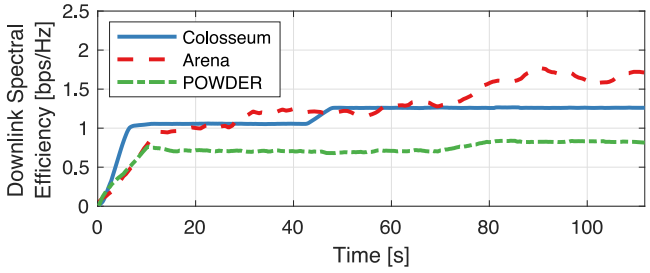


Figure 17: Downlink spectral efficiency of SCOPE ported on three different testbeds: Colosseum, Arena and POWDER.

We notice that SCOPE *achieves performances showing similar trends in three very different experimental facilities*: A wireless emulator, an indoor testbed and a large-scale outdoor platform. We also observe that SCOPE not only adapts to diverse environments, but also to different numbers of UEs. This demonstrates the *feasibility and effectiveness of prototyping solutions with SCOPE on Colosseum before testing them in the field.*

6 RELATED WORK

We are aware of only a handful of works concerning network slicing implementations for srsLTE-based cellular networks. Garcia-Aviles et al. propose a multi-slice service-orchestration framework [23] and implement it on a small-scale prototype [24], while Ayala-Romero et al. devise a deep learning approach for joint allocation of computational and radio resources [3]. Furthermore, D'Oro et al. proposed a multi-access edge computing (MEC) framework for resource orchestration on heterogeneous network slices [21], and demonstrated optimal network slicing solutions for small-scale 5G network deployments [20].

A proof of concept of RAN slicing on the 5G-EmPOWER platform is given by Coronado et al. [16]. Although this work offers interesting insights, the provided implementation considers only a single BS and two UEs. Koutlia et al. describe an experimental testbed with slicing support [30]. The focus of this work is primarily on policy enforcement, slice provisioning and admission control. Moreover, slices are statically allocated and they cannot be reconfigured at run time.

Finally, a centralized and cloud-based slicing framework based on network function virtualization is developed by Marinova et al. [33]. While network parameters can be reconfigured at run time, this comes at the cost of restarting and redeploying the network BSs and subsequent re-attachment of UEs. This operation requires a network downtime tens of seconds long, hardly acceptable given the sub-millisecond latency requirements of NextG networks. Furthermore, no open APIs are offered to interface user-defined control logic.

Table 2 provides a high-level comparison of the capabilities of the main SDR-enabled testbeds for NextG experimental research.

Table 2: SDR-enabled NextG testbeds.

Testbed	NextG Capabilities	Software	Deployment
Arena	sub-6 GHz	RAN & core	Large-scale office
Colosseum	sub-6 GHz, NFV/orchestration	RAN & core, O-RAN	Large-scale, programmable
CORNET	sub-6 GHz	RAN & core	Large-scale indoor
COSMOS	sub-6 GHz, mmWave, NFV/orchestration	RAN & core, O-RAN	Indoor, city-scale outdoor
Drexel Grid	sub-6 GHz	RAN & core	Large-scale indoor, virtual nodes
FIT testbeds	sub-6 GHz, NFV/orchestration	RAN & core, OSM	Large-scale indoor
IRIS	sub-6 GHz, cloud-RAN, NFV/orchestration	RAN & core	Large-scale indoor
NITOS	sub-6 GHz, NFV/orchestration	RAN & core, OSM	Large-scale indoor and outdoor, office
POWDER	sub-6 GHz, NFV/orchestration	RAN & core, O-RAN	Indoor, city-scale outdoor
SCOPE	sub-6 GHz	RAN & core	Large-scale, programmable

Arena [4], CORNET [15], and Drexel Grid [17] focus on sub-6 GHz RAN and core network applications in large-scale indoor setups [5, 7, 21, 42]. Colosseum [14], COSMOS [43], the FIT testbeds [22], NITOS [36], IRIS [28], and POWDER [10], instead, also encompass Network Function Virtualization (NFV) and orchestration through frameworks such as O-RAN and Open Source MANO (OSM) [6, 29, 37] (with COSMOS and IRIS also having mmWave and cloud-RAN capabilities, respectively). As for the deployment of the radio nodes, these testbeds target large-scale (city-scale for COSMOS and POWDER) indoor or outdoor setups. However, experiments run on them are constrained to the physical deployment of the testbed SDRs. The only exceptions are the Drexel Grid testbed, which includes a set of virtual nodes that interact with the physical ones, and Colosseum, which allows to emulate different wireless environments and setup with hardware-in-the-loop.

Finally, SCOPE positions itself as a prototyping platform for sub-6 GHz NextG solutions that complements the above testbeds by providing open APIs for real-time reconfiguration of cross-layer RAN functionalities along with a portable and platform-independent containerized implementation. If instantiated on the above testbeds, SCOPE is able to target indoor and outdoor wireless deployments by seamlessly adapting to their underlying physical infrastructure.

Looking at the broader NextG scene, in the last few years several industry consortia have been focusing on developing solutions to redesign and revolutionize cellular networks. The most noteworthy of these efforts is O-RAN, which disaggregates network functionalities and enables their virtualized execution on different hardware components [38]. This is done through O-RAN RAN Intelligent Controller (RIC), which enables centralized control of the RAN at different time scales and granularities. This component can also host third-party applications, called *xApps*, which interact with the RAN APIs to control the 3GPP Central Units (CUs) and Distributed Units (DUs) [1, 8]. It is worth mentioning that SCOPE is *O-RAN-ready*. Indeed SCOPE APIs can seamlessly interface with the RAN by using similar routines and structures as the ones defined for O-RAN xApps, and control the network elements at run time [6].

7 CONCLUSIONS

We presented SCOPE, a development environment for softwarized and virtualized NextG cellular networks. SCOPE provides: (i) A ready-to-use portable open-source cellular container with flexible 5G-oriented functionalities; (ii) data collection tools, such as dataset generation functions for recording cellular performance and metrics, and for facilitating data analysis; (iii) a set of APIs to control and reprogram key functionalities of the full cellular stack at run time, without requiring redeploying the network, and (iv) an emulation environment with diverse cellular scenarios closely matching real-world deployments for precise prototyping NextG network solutions. We showcased SCOPE usage in the Colosseum network emulator and demonstrated its flexibility by porting it to real-world testbeds, both indoor (Arena) and outdoor (POWDER). Researchers can use SCOPE to design, implement and test novel control solutions on large-scale real-world cellular scenarios with different topologies, mobility patterns, channel and traffic characteristics. Finally, we discussed how SCOPE can be used for practical prototyping of ML, optimization and heuristic algorithms.

ACKNOWLEDGMENTS

This work was partially supported by the U.S. National Science Foundation under Grant CNS-1923789 and the U.S. Office of Naval Research, Grants N00014-19-1-2409 and N00014-20-1-2132.

REFERENCES

[1] 3GPP. 2018. *Study on CU-DU Lower Layer Split for NR.* Technical Report (TR) 38.816. 3rd Generation Partnership Project (3GPP). http://www.3gpp.org/DynaReport/38816.htm Version 15.0.0.

[2] 3GPP. 2020. *Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures.* Technical Specification (TS) 36.213. 3rd Generation Partnership Project (3GPP). http://www.3gpp.org/DynaReport/36213.htm Version 16.2.0.

[3] J.A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Pérez, A. Banchs, and J.J. Alcaraz. 2019. VrAIn: A Deep Learning Approach Tailoring Computing and Radio Resources in Virtualized RANs. In *Proceedings of ACM MobiCom* (Los Cabos, Mexico).

[4] L. Bertizzolo, L. Bonati, E. Demirors, A. Al-Shawabka, S. D'Oro, F. Restuccia, and T. Melodia. 2020. Arena: A 64-antenna SDR-based Ceiling Grid Testing Platform for Sub-6 GHz 5G-and-Beyond Radio Spectrum Research. *Computer Networks* 181 (November 2020), 1–17.

[5] L. Bonati, S. D'Oro, L. Bertizzolo, E. Demirors, Z. Guan, S. Basagni, and T. Melodia. 2020. CellOS: Zero-touch Softwarized Open Cellular Networks. *Computer Networks* 180 (October 2020).

[6] L. Bonati, S. D'Oro, M. Polese, S. Basagni, and T. Melodia. 2020. Intelligence and Learning in O-RAN for Data-driven NextG Cellular Networks. *arXiv:2012.01263 [cs.NI]* (December 2020).

[7] L. Bonati, S. D'Oro, F. Restuccia, S. Basagni, and T. Melodia. 2021. SteaLTE: Private 5G Cellular Connectivity as a Service with Full-stack Wireless Steganography. In *Proceedings of IEEE INFOCOM.* Virtual Conference.

[8] L. Bonati, M. Polese, S. D'Oro, S. Basagni, and T. Melodia. 2020. Open, Programmable, and Virtualized 5G Networks: State-of-the-Art and the Road Ahead. *Computer Networks* 182 (December 2020), 1–28.

[9] D. Brake. 2020. A U.S. National Strategy for 5G and Future Wireless Innovation. https://itif.org/publications/2020/04/27/us-national-strategy-5g-and-future-wireless-innovation.

[10] J. Breen, A. Buffmire, J. Duerig, K. Dutt, E. Eide, M. Hibler, D. Johnson, S. Kumar Kasera, E. Lewis, D. Maas, A. Orange, N. Patwari, D. Reading, R. Ricci, D. Schurig, L. B. Stoller, K. Van der Merwe, K. Webb, and G. Wong. 2020. POWDER: Platform for Open Wireless Data-driven Experimental Research. In *Proceedings of ACM WiNTECH.* London, United Kingdom.

[11] G. Brown. 2020. TIP OpenRAN: Toward Disaggregated Mobile Networking. https://cdn.brandfolder.io/D8DI15S7/as/qc19tk-54bsw-305pae/TIP_OpenRAN_-Heavy_Reading_May_2020-_White_Paper.pdf.

[12] Canonical. 2021. LXD Network Configuration. https://linuxcontainers.org/lxd/docs/master/networks.

[13] Canonical. Accessed April 2021. Linux Containers. https://linuxcontainers.org.

[14] Colosseum. Accessed April 2021. https://www.colosseum.net.

[15] CORNET. Accessed April 2021. https://cornet.wireless.vt.edu.

[16] E. Coronado, S.N. Khan, and R. Riggio. 2019. 5G-EmPOWER: A Software-Defined Networking Platform for 5G Radio Access Networks. *IEEE Transactions on Network and Service Management* 16, 2 (June 2019), 715–728.

[17] K. R. Dandekar, S. Begashaw, M. Jacovic, A. Lackpour, I. Rasheed, X. R. Rey, C. Sahin, S. Shaher, and G. Mainland. 2019. Grid Software Defined Radio Network Testbed for Hybrid Measurement and Emulation. In *Proceedings of IEEE SECON.* Boston, MA, USA.

[18] DARPA. 2021. Spectrum Collaboration Challenge. https://archive.darpa.mil/sc2.

[19] R. Doost-Mohammady, O. Bejarano, and A. Sabharwal. 2020. Good Times For Wireless Research. In *Proceedings of ACM WiNTECH.* London, United Kingdom.

[20] S. D'Oro, L. Bonati, F. Restuccia, and T. Melodia. 2021. Coordinated 5G Network Slicing: How Constructive Interference Can Boost Network Throughput. *IEEE/ACM Transactions on Networking* (April 2021), 1–14.

[21] S. D'Oro, L. Bonati, F. Restuccia, M. Polese, M. Zorzi, and T. Melodia. 2020. SI-EDGE: Network Slicing at the Edge. In *Proceedings of ACM Mobihoc.* Virtual.

[22] FIT Testbeds. Accessed April 2021. https://fit-equipex.fr.

[23] G. Garcia-Aviles, M. Gramaglia, P. Serrano, and A. Banchs. 2018. POSENS: A Practical Open Source Solution for End-to-End Network Slicing. *IEEE Wireless Communications* 25, 5 (October 2018), 30–37.

[24] G. Garcia-Aviles, M. Gramaglia, P. Serrano, F. Gringoli, S. Fuente-Pascual, and I. Labrador Pavon. 2020. Experimenting with Open Source Tools to Deploy a Multi-service and Multi-slice Mobile Network. *Computer Communications* 150 (January 2020), 1–12.

[25] I. Gomez-Miguelez, A. Garcia-Saavedra, P.D. Sutton, P. Serrano, C. Cano, and D.J. Leith. 2016. srsLTE: An Open-source Platform for LTE Evolution and Experimentation. In *Proceedings of ACM WiNTECH.* New York City, NY, USA.

[26] L. Hardesty. 2020. Altiostar Joins Mavenir to Develop Open RAN Radios for U.S. Carriers. https://www.fiercewireless.com/wireless/altiostar-joins-mavenir-to-develop-open-ran-radios-for-u-s-carriers.

[27] HP Enterprise. 2020. HPE Speeds Up 5G Adoption with Cloud Native 5G Core Software Stack, Available as-a-Service. https://www.hpe.com/us/en/newsroom/press-release/2020/03/hpe-speeds-up-5g-adoption-with-cloud-native-5g-core-software-stack-available-as-a-service.html

[28] IRIS. Accessed April 2021. http://iristestbed.eu.

[29] P. Karamichailidis, K. Choumas, and T. Korakis. 2019. Enabling Multi-Domain Orchestration using Open Source MANO, OpenStack and OpenDaylight. In *Proceedings of IEEE LANMAN.* Paris, France.

[30] K. Koutlia, R. Ferrús, E. Coronado, R. Riggio, F. Casadevall, A. Umbert, and J. Pérez-Romero. 2019. Design and Experimental Validation of a Software-Defined Radio Access Network Testbed with Slicing Support. *Wireless Communications and Mobile Computing* (June 2019), 1–17.

[31] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim. 2019. Applications of Deep Reinforcement Learning in Communications and Networking: A Survey. *IEEE Communications Surveys & Tutorials* 21, 4 (May 2019), 3133–3174.

[32] S. Marek. 2020. 5G's Future Includes a Cloud-native Architecture Complete with Containers. https://www.lightreading.com/cloud-native-nfv/5gs-future-includes-a-cloud-native-architecture-complete-with-containers/d/d-id/762116.

[33] S. Marinova, T. Lin, H. Bannazadeh, and A. Leon-Garcia. 2020. End-to-end Network Slicing for Future Wireless in Multi-region Cloud Platforms. *Computer Networks* 177 (August 2020), 1–10.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *arXiv preprint arXiv:1312.5602* (2013).

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (February 2015), 529–533.

[36] NITOS. Accessed April 2021. https://nitlab.inf.uth.gr/NITlab/nitos.

[37] O-RAN Alliance Conducts First Global Plugfest to Foster Adoption of Open and Interoperable 5G Radio Access Networks. 2019. https://static1.squarespace.com/static/5ad774cce74940d7115044b0/t/5dfba8fb1326ae1bcf4a8b6f/1576773884092/O-RAN-2019.12.19-EC-C-PR-on-2019-Plugfest-v1.0.pdf.

[38] O-RAN Alliance White Paper. 2018. O-RAN: Towards an Open and Smart RAN. https://www.o-ran.org/resources.

[39] Platforms for Advanced Wireless Research. Accessed May 2021. https://www.advancedwireless.org.

[40] POWDER Deployment. 2021. https://www.powderwireless.net/area.

[41] Rakuten. 2020. How Elegant Software Can Make 5G Networks More Resilient. https://rakuten.today/blog/5g-network-reliability-lightreading.html.

[42] R.M. Rao, V. Marojevic, and J.H. Reed. 2019. Analysis of Non-Pilot Interference on Link Adaptation and Latency in Cellular Networks. In *Proceedings of IEEE VTC2019-Spring.* Kuala Lumpur, Malaysia.

[43] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, H. Krishnaswamy, S. Maheshwari, P. Skrimponis, and C. Gutterman. 2020. Challenge: COSMOS: A City-Scale Programmable Testbed for Experimentation with Advanced Wireless. In *Proceedings of ACM MobiCom.* London, United Kingdom.

[44] M. Robuck. 2020. AT&T on Target for Virtualizing 75% of its Network by 2020. https://www.fiercetelecom.com/telecom/at-t-target-for-virtualizing-75-its-network-by-2020.

[45] M.L. Sichitiu, I. Guvenc, R. Dutta, V. Marojevic, and B. Floyd. 2020. AERPAW Emulation Overview. In *Proceedings of ACM WiNTECH.* London, United Kingdom.

[46] Software Radio Systems. Accessed May 2021. srsLTE (srsRAN) Repository. https://github.com/srsran/srsRAN.

[47] M. Tehrani-Moayyed, L. Bonati, P. Johari, T. Melodia, and S. Basagni. 2021. Creating RF Scenarios for Large-Scale, Real-Time Wireless Channel Emulators. In *Proceedings of Mediterranean Communication and Computer Networking Conference (MedComNet).* Virtual.

[48] The Status of Open Source for 5G. 2019. 5G Americas Whitepaper. https://www.5gamericas.org/wp-content/uploads/2019/07/5G_Americas_White_Paper_The_Status_of_Open_Source_for_5G_Feb_2019.pdf.

[49] Unwired Labs. Accessed April 2021. OpenCelliD. https://opencellid.org.

[50] U.S. Naval Research Laboratory. 2019. Multi-Generator. https://www.nrl.navy.mil/itd/ncs/products/mgen.

[51] D. Vallis. 2020. 5G Breaks from Proprietary Systems, Embraces Open Source RANs. https://www.5gtechnologyworld.com/5g-breaks-from-proprietary-systems-embraces-open-source-rans.

[52] W. Wang, Y. Huang, Y. Wang, and L. Wang. 2014. Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction. In *Proc. of IEEE/CVF CVPR Workshops.* 490–497.

[53] Q. Yang, Y. Liu, T. Chen, and Y. Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology* 10, 2 (2019), 1–19.