This is the authors' accepted version of the article. The final version published by IEEE is L. Lo Schiavo, J. A. Ayala-Romero, A. Garcia-Saavedra, M. Fiore and X. Costa-Perez, "YinYangRAN: Resource Multiplexing in GPU-Accelerated Virtualized RANs," IEEE INFOCOM 2024 - IEEE International Conference on Computer Communications, 2024, pp. TBD, doi: TBD.

YinYangRAN: Resource Multiplexing in GPU-Accelerated Virtualized RANs

Leonardo Lo Schiavo*†, Jose A. Ayala-Romero‡, Andres Garcia-Saavedra‡, Marco Fiore†, Xavier Costa-Perez§‡
* Universidad Carlos III de Madrid, Spain, † IMDEA Networks Institute, Spain
‡ NEC Laboratories Europe GmbH, Germany, § i2CAT Foundation and ICREA, Spain

***Abstract*—RAN virtualization is revolutionizing the telco industry, enabling 5G Distributed Units to run using general-purpose platforms equipped with Hardware Accelerators (HAs). Recently, GPUs have been proposed as HAs, hinging on their unique capability to execute 5G PHY operations efficiently while also processing Machine Learning (ML) workloads. While this ambivalence makes GPUs attractive for cost-effective deployments, we experimentally demonstrate that multiplexing 5G and ML workloads in GPUs is in fact challenging, and that using conventional GPU-sharing methods can severely disrupt 5G operations. We then introduce YinYangRAN, an innovative O-RAN-compliant solution that supervises GPU-based HAs so as to ensure reliability in the 5G processing pipeline while maximizing the throughput of concurrent ML services. YinYangRAN performs GPU resource allocation decisions via a computationally-efficient approximate dynamic programming technique, which is informed by a neural network trained on real-world measurements. Using workloads collected in real RANs, we demonstrate that YinYangRAN can achieve over 50% higher 5G processing reliability than conventional GPU sharing models with minimal impact on co-located ML workloads. To our knowledge, this is the first work identifying and addressing the complex problem of HA management in emerging GPU-accelerated vRANs, and represents a promising step towards multiplexing PHY and ML workloads in mobile networks.**

I. Introduction

Radio Access Network (RAN) virtualization enables baseband processing on commercial off-the-shelf (COTS) computing platforms. The approach decouples base station (BS) tasks from dedicated hardware, and eases the disaggregation of monolithic and close BS equipment into minimal Radio Unit (RU) hardware connected to cloud-oriented platforms that take care of all computationally intensive signal processing tasks.

Virtualized RANs (vRANs) based on this concept promise to bring unprecedented flexibility in the management of PHY- and MAC-layer processing, and yield the potential to disrupt traditional vendor lock-ins that have historically stagnated the RAN ecosystem, stifling competition and innovation. It is thus unsurprising that vRANs have garnered significant attention from the mobile telecommunications industry in the context of 5G: driven by the O-RAN Alliance [1], virtually all leading industry players are investing in 5G vRAN development [2], [3], with forecasts that the technology will eclipse conventional RANs and generate revenues of $20 billion by 2028 [4], [5].

As illustrated in Fig. 1, 5G BSs are disaggregated into the RU, which performs basic RF operations, the Distributed Unit (DU), responsible for PHY- and MAC-layer tasks, and the Centralized Unit (CU), in charge of the highest BS layers.

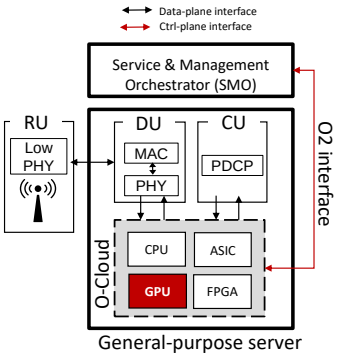

Fig. 1. O-RAN Cloud system.

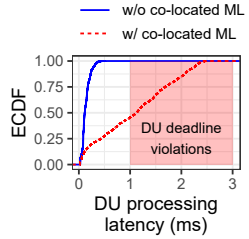

Fig. 2. Empirical CDF of the 5G DU PHY processing latency using (i) a dedicated NVIDIA A100 GPU (blue line), and (ii) the same GPU shared with ML workloads.

Virtualized RANs implement most of DU and CU tasks as software components that run on virtualized platforms, typically containers. However, the most compute-intensive DU operations (e.g., Forward Error Correction, or FEC) are offloaded to a Hardware Accelerator (HA) *to meet certain processing latency targets*. Traditional HAs include PCIe boards integrating ASICs [6] or FPGAs [7], and are mandatory to execute compute-intensive PHY tasks with guaranteed latency in industry-grade mobile systems [8].

Recently, GPUs have emerged as a compelling alternative for 5G DU HAs, with commercial solutions developed by market leaders like NVIDIA [9], [10]. Leveraging the massive computational parallelism offered by arrays of streaming multiprocessors (SMs), GPUs promise high-performance DU load processing [11]. As an example, Fig. 2 (solid blue curve) shows the results of tests we run on a NVIDIA A100 GPU serving a 5G 100-MHz DU workload recorded from operational RANs: the GPU can keep the processing latency distribution well below 1 ms, a conventional target in 5G [8].

A feature unique to GPUs, and a major selling point by vendors pushing for wide adoption of the technology, is their potential to be shared between computationally intensive 5G PHY-layer processing and the different Machine Learning (ML) workloads that will be run at DUs [11]. Indeed, many DU operations require fast automated decision-making that is expected to build on advanced ML models, which in turn need GPU resources to execute efficiently; examples include traffic load forecasting at RU level [12], [13], policing of radio resource schedulers [14], [14], possibly with awareness of compute resources [15], [16], just to name a few. The capability of multitasking such ML operations with DU PHY functions enables a cost-effective utilization of high-priced HAs, justifying their adoption over ASIC/FPGA-based HAs.

However, having DU and ML workloads share GPU resources is far from obvious in practice. For instance, our experiments (dashed red curve) in Fig. 2 show that co-locating a realistic ML inference task with the DU workloads in the A100 GPU using the conventional software concurrency methods (see §II) results in substantial disruption of the 5G pipeline operation. Indeed, the PHY processing latency becomes highly erratic, with a low 50% probability of meeting latency targets that is an unacceptable industry-grade performance.

In light of these results, it does become evident that GPU-based vRAN acceleration introduces new resource control challenges that are alien to traditional radio systems, or to conventional vRANs based on ASIC and FPGA HAs. Specifically, to maximize gains, it is crucial to devise robust mechanisms that aptly allocate GPU resources between *inelastic* PHY-layer processing at DUs and *elastic* ML workloads. These mechanisms must effectively learn the complex relationships between time-varying wireless communication dynamics [8] and their associated computational needs, and then share GPU resources so that DUs meet their processing targets while ML processing throughput is maximized.

The task is complicated by a number of obstacles.

- First, industry standards enforce tight DU processing latency of around 1 ms [8] to be upheld with a given probability to provide a pre-determined *reliability* target: therefore, one must hard-guarantee enough GPU resources to meet that requirement.
- Second, DU PHY workloads are intrinsically stochastic, as they are contingent on random factors such as user mobility patterns or environmental scatterers, and allocating GPU resources to them is arduous.
- Third, as we will later demonstrate, GPU resource reallocation involves considerable overheads, which must be factored into the optimal system control.
- Lastly, as also depicted in Fig. 1, O-RAN specification platforms only permit compute-control decisions at non-real time intervals (seconds or even minutes) through an O2 interface between the Service and Management Orchestrator (SMO) and the O-Cloud hosting GPU resources used by virtualized DUs, which creates a significant disconnect between long-timescale decision-making and short-timescale system operation.

We propose YinYangRAN, a resource control solution designed to be integrated into the O-RAN's Service Management and Orchestration (SMO). YinYangRAN dynamically allocates the minimum necessary GPU resources to 5G DUs to ensure maximal reliability, which enables reusing spare resources for other ML services thereby enhancing the overall efficiency of costly GPU platforms. Our solution is rooted in a two-stage approach derived from an approximate dynamic programming technique known as Certainty Equivalent Control (CEC). The first stage is a more compute-intensive procedure where an approximation of the optimal cost is computed *offline* using CEC. The second stage is a lightweight, dynamic resource allocation algorithm designed for online operation. This decision-making mechanism is assisted by a neural network trained with real-world traffic loads that predicts the distribution of the DU reliability to ensure the satisfaction of the latency targets. In summary, our work makes the following contributions.

- We perform extensive measurements with high-end processors and industry-grade PHY pipelines, and characterize for the first time the performance of high-end GPUs for computationally intensive LDPC decoding. Relying on recent strategies for GPU resource sharing, we prove that there exist substantial opportunities for multiplexing DU processing with ML workloads.
- We show, however, that such a sharing space is complex, as it depends in entangled ways on the heterogeneous *contexts* of 5G demands and on unique GPU resource reconfiguration delays, all of which create trade-offs between decoding reliability and resource utilization efficiency that are difficult to model and exploit.
- We devise YinYangRAN, a solution to efficiently manage GPU resources, based on approximated dynamic programming. Using an experimental prototype and data from operational RANs, we demonstrate that YinYangRAN achieves over 50% higher processing reliability than conventional GPU sharing models with minimal impact on co-located ML workloads.

To the best of our knowledge, this paper is the first work highlighting the reliability risk of blindly sharing GPUs for both 5G PHY and ML workloads; and YinYangRAN is the first solution to efficiently solve this problem.

II. Background

In this section, we present a primer on 5G RAN (§II-A), and introduce modern techniques to sharing GPUs (§II-B).

A. 5G New Radio

O-RAN 5G base stations are composed of a Radio Unit (RU) handling basic radio tasks such as FFTs, a Distributed Unit (DU) taking care of PHY, MAC, and RLC layers [17], and a Central Unit (CU) managing higher layer tasks.

The 5G PHY/MAC interface is called New Radio (NR). Our focus is on sub-6GHz bands, which allow radio bandwidth configurations of up to 100MHz for each carrier, and provide a flexible *numerology* $\mu = \{0, 1, 2\}$. The basic unit of spectrum is a resource block (RB), which is comprised of 12 subcarriers with $15 \cdot 2^{\mu}$-KHz spacing. The time domain is segmented into 1-ms subframes, each containing 2^{μ} slots that typically include 14 OFDM symbols with a duration of $66.7 \cdot 2^{-\mu}\mu$s. Every Transmission Time Interval (TTI) that usually takes one slot, the DU's MAC gathers a set of bits from/to every active User Equipment (UE) into a Transport Block (TB) per user. The size of each TB is contingent upon the numerology, the amount of buffered data, a RB scheduling policy, and a modulation and coding scheme (MCS) chosen based on the signal-to-noise ratio (SNR). Hence, every TTI, the DU has to process a PHY workload with a specific *context*, i.e., a set of TBs and their characteristics (SNR, MCS, number of RBs).

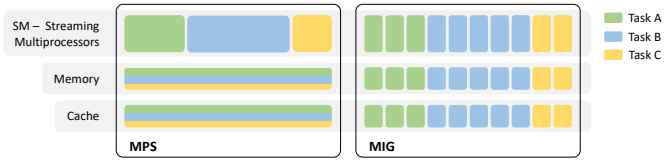

Fig. 3. MPS versus MIG.

On the transmitting end, TBs are split into code blocks (CBs) each with its own CRC fields. Filler bits adjust the CB size to meet the LDPC encoder's demands used for FEC, generating a codeword with parity bits. Lastly, the codeword is matched to the capacity of the allocated RBs (as per the MCS) via rate matching. On the receiving side, a soft-output detector calculates the data reliability as log-likelihood ratios (LLR), known as *soft bits*. An LDPC decoder then converts these soft bits into hard bits using an iterative belief propagation algorithm. The TB is rebuilt once all its CBs are successfully decoded. For additional details, refer to [18]. 3GPP and O-RAN specify concrete processing latency bounds between UEs and CUs [17], [19]. Consequently, the most-compute intense PHY tasks, usually FEC, have stringent latency targets [8], [16], which are usually met by a hardware accelerator (HA) like a GPU as considered in this paper.

B. GPU Resource Sharing

As stated in §I, GPU-based HAs are gaining the attention of mobile operators thanks to their capability of reusing computing resources for ML workloads [11]. For instance, NTT Docomo are set to roll out GPU-accelerated 5G trials this year, and SoftBank have recently explored a proof-of-concept about sharing the resources of a GPU between 5G DU workloads and ML-based edge computing applications [10].

There are essentially three methods to share GPU resources. One is application-level sharing, which leverages conventional single-process concurrency methods, such as CUDA streams and multi-threading. While remarkably simple, time-sliced context switching can create significant and non-controllable overhead, ruling out per-process performance guarantees that are critical in PHY-layer processing pipelines.

Process-level sharing can be achieved with Multi-Process Sharing (MPS), first introduced on NVIDIA's Kepler architectures and further enhanced on Volta. MPS assigns subsets of GPU compute units, known as streaming multi-processors (SM), to individual partitions for specific processes. Although MPS offers SM isolation to circumvent context switching overheads, cache and memory resources are still shared between processes without any isolation, as illustrated in Fig. 3. This can potentially lead to large overheads when memory-intensive processes compete for GPU resources.

Finally, hardware-level sharing can be achieved with Multi-Instance GPU (MIG), a feature appeared with Ampere GPUs (e.g., NVIDIA A100). As shown in Fig. 3, MIG facilitates true hardware isolation, guaranteeing Quality of Service (QoS) and resource allocation. Specifically, MIG enables the division of a GPU into fully isolated sub-GPUs, which in practice only share the PCI interface bandwidth towards the CPU.

While it ensures full isolation, MIG presents two notable drawbacks compared to MPS. First, MIG offers only a limited set of partition options — for example, the smallest partition in a 40GB NVIDIA A100 provides 1/7th of SMs and 5GB of memory. Conversely, MPS allows for more granular SM splitting. Second, GPU repartitioning using MIG incurs substantial overhead, on the order of several seconds, as opposed to a few hundred milliseconds with MPS; we will empirically assess this in §III. Ultimately, these limitation makes MIG less suitable for exploiting GPU multiplexing opportunities real-world 5G vRANs, as discussed in more details in §III-D.

III. Analysis

Re-using spare HA resources from the O-Cloud to handle tasks external to PHY processing is a unique opportunity to build more cost-effective vRAN systems. Previous works have explored this strategy with CPU resources [8], yet industry-grade DUs hardly rely upon CPU for hardware acceleration [20]. GPUs are a much more realistic candidate for HA, yet, as anticipated in §I, they also pose novel challenges.

A. Practical trade-offs in DU/ML GPU sharing

The industry imposes PHY processing latency targets that range from 0.5 to 3 ms, which the DU must satisfy with a certain probability to attain a certain reliability goal. In order to optimize vRAN cost-efficiency, it is critical to allocate to DUs just the right amount of GPU resources needed to meet their target, so as to free as much computing capacity as possible to maximize ML performance and GPU utilization. The conflicting goals create a trade-off of fulfilling 5G processing latency targets and maximizing the ML throughput.

We experimentally characterize this trade-off by analyzing a commercial solution for GPU-accelerated 5G processing on a NVIDIA A100 GPU with 40 GB of RAM. Motivated by real-world RAN workloads (see §III-C), we consider a wide range of workload contexts, i.e., combinations of MCS, RBs, SNR, and concurrent users in a 5G 100-MHz DU. We also run ML tasks on the same GPU and measure the related throughput as the number of inferences per second. More specifically, we deploy the TES-RNN model [21], which is a state-of-the-art predictor in mobile traffic forecasting that blends statistical modeling and deep neural networks to predict traffic loads. All this is done with the control of the GPU computing resources (SMs) via MPS by assigning a given amount of resources to the DU and its complementary value to the ML workload.

The result of these experiments is illustrated in Fig. 4. The plot depicts in blue the FEC processing latency of the DU under diverse GPU resource allocations, along the abscissa. Specifically, the dark blue lines show the highest and lowest latency performance recorded across all possible contexts; the light blue area in between illustrates the possible operating region of the DU for any intermediate context, with thin lines therein showing the latency profiles of some sample contexts. For illustration purposes, the dashed horizontal line highlights one of the most conventional PHY-layer processing latency targets, i.e., 1 ms, which the DU must meet.

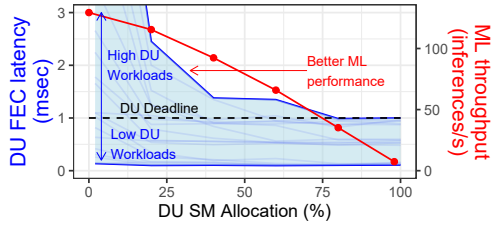

Fig. 4. DU performance (FEC processing latency, in blue) *vs* ML performance (inferences per second, in red) for different SM allocations and DU contexts (combinations of number of concurrent TBs, SNR, MCS, and RBs). The x-axis indicates the allocation for DU workloads. The allocation for ML workload is the complementary of the x-axis ((100-x)%). The light blue area depicts the operating region for DU FEC workloads across a wide range of contexts.

We observe that the latency curves span very diverse values, both above and below the 1-ms target. Most relevantly, the minimum fraction of GPU SMs required to fulfil the target ranges all the way from 1% to 80% depending on the DU context. In other words, our experiments demonstrate for the very first time that sub-6GHz 5G PHY-layer processing only needs a (potentially very small) portion of the full capacity of a modern GPU, hence there exists a substantial space for multiplexing DU requests with other ML workloads. To prove this point, as explained above, we also run a trained ML model on the SMs of the same GPU not allocated to the DU processing. The red curve in Fig. 4 shows that the throughput of the ML model, in terms of inferences per second, is strongly dependent on the amount of spare SMs it is allowed to use. Thus, operating DUs only with the limited GPU resources they really necessitate opens the way to significant gains in terms of ML performance for co-located edge and O-RAN services.

B. Characterizing DU processing latency in GPU HAs

The multiplexing space identified above is not homogeneous, as different contexts entail very diverse latency curves for DU processing depending on the dedicated SMs. We shed light on the GPU resource requirements of DU workloads by empirically measure the latency performance when processing a single 100-MHz TB with a wide range of combinations of MCS and SNR on the setup described in §III-A, while also sweeping through a number of SM allocations for that task.

Fig. 5 presents excerpts of the results, for three GPU configurations where 15%, 50% and 100% of the available SMs are reserved to DU processing. In each plot, the FEC processing latency is reported as a function of the SNR and the MCS of the demand. A higher MCS typically results in increased latency because it encodes more bits into the TB, thereby requiring a longer decoding time. For a given MCS, a lower SNR can further inflate latency, as the decoder may necessitate more iterations over the data to decode it. Moreover, reducing the allocation of SM computational resources naturally increases the latency too.

However, most importantly, we observe that the relationship between latency, MCS, SNR, and the amount of SM resources forms complex patterns. These intricate patterns challenge the formulation of tractable mathematical models. Such a complex FEC latency behavior is further compounded by different

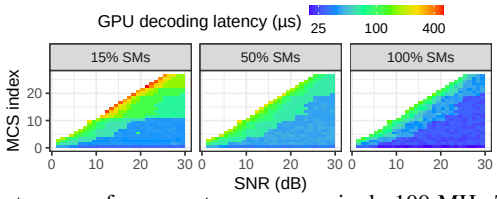

Fig. 5. Latency performance to process a single 100-MHz TB on an NVIDIA A100 GPU with different combinations of MCS, SNR, and allocation of GPU SM resources.

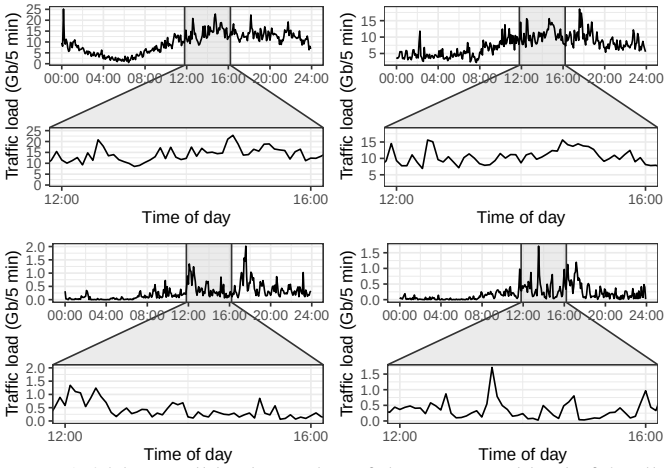

Fig. 6. A 24-hour cell load snapshot of the aggregated load of 4 cells from a large operator in a major European city.

allocations of radio resources (which impact the size of the TB through the number of allocated RBs) and the number of concurrent users in the system (which affects the number of TBs that the GPU must process simultaneously). Due to space constraints, these additional variables are not illustrated here.

C. Dynamic DU workloads

In addition to highly heterogeneous performance across different contexts, DU loads are also affected by the strong temporal variance of demands generated by mobile subscribers. As an example, Fig. 6 depicts 24-hour snapshots of the real-world load dynamics recorded at five-minute granularity in a few 4G and 5G RAN cells with Falcon [22] and 5GSniffer [23]. The plots also zoom in into a 4-hour window. Clearly, the time series show large fluctuations, which translate in very different burdens on the DU, and hence on its required amount of GPU resources. Yet, these patterns are predictable as demonstrated in [21], so we can anticipate the DU resource requirements with appropriate data-driven models.

Ultimately, our results unveil the need to adapt the allocation of GPU resources to changing vRAN conditions in terms of both traffic demands and workload contexts. This calls for dynamic control solutions that can amortize the cost of expensive GPU-based HAs by rescuing computing power for concurrent ML workloads without compromising DU reliability.

D. Overheads in GPU resource reconfiguration

To meet the above-mentioned goal, one last aspect of GPU-accelerated DU operation that we need to characterize are the overheads incurred by GPU resources re-configurations. These are an important factor when taking decisions on updating the share of SMs allocated to DUs and ML workloads.

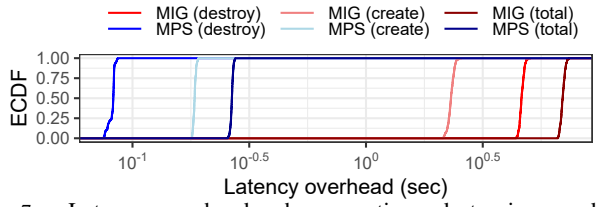

Fig. 7. Latency overhead when creating, destroying, and re-configuring (creating+destroying) a GPU slice using MIG *vs* MPS technology. Logarithmic x-axis.

In §II, we introduced two methods for sharing GPU resources: MIG and MPS. As mentioned therein, MIG provides full resource isolation among tasks, while MPS only partitions SMs by keeping memory and caches as shared resources. The two approaches induce very different latency overheads when re-configuring a GPU slice, as reported in Fig. 7. The plot summarizes 200 random re-configurations on our NVIDIA A100 GPU with both MIG (red) and MPS (blue): each re-configuration changing the partitioning of SMs across tasks, which implies tearing down the existing SM configuration and creating a new one. It is worth noting that during the re-configuration process the GPU cannot run any task.

As shown in the figure, the median latency is fairly constant for each strategy. More precisely, re-configuring a MIG partition in the target GPU takes 6.9 seconds, against the 0.266 seconds of re-allocating SM resources using MPS. During such re-configuration intervals, we let the higher-priority DU fall back to software processing, e.g., using available CPU cores in the O-Cloud, which can process the workload at a cost in energy consumption [24]. We provide more details in §VI. As continued and substantial performance drops during several seconds are hardly acceptable in production-grade systems, MIG does not seem a viable option for GPU-based HA sharing. On the other hand, the delays introduced by MPS appear bearable for non-real time O-RAN operations that occur at the order of seconds.

We recall that MPS does not provide memory or cache isolation, which could potentially cause resource contention and lower raw performance than MIG. Yet, unlike MIG that is constrained to a small set of possible partition configurations, MPS provides full flexibility when allocating SM resources (as also captured by Fig. 4), which can compensate for the shared memory with a better adaptation to DU workload dynamics.

In conclusion, based on the experimental analysis above, we advocate for a dynamic MPS-based GPU resource allocation method, and will adopt this strategy in the rest of the paper.

IV. Problem Formulation

We consider an O-Cloud platform comprising one GPU that runs two services at the same time: 5G PHY operations offloaded from a virtualized DU, and a deep learning model offloaded from a general-purpose ML service. We use MPS (introduced in §II-B) to allocate GPU resources (SMs) to each of the services. At each TTI, a set of TBs arrives at the DU to be decoded. Each TB is characterized by its SNR, MCS, and TB size (number of bits). Note that the computational capacity needed by the DU to decode these TBs depends on the number of received TBs that changes at every TTI and the SNR, MCS, and TB size of each TB (see §III).

Smaller SM allocations for the DU may imply that some TBs are not decoded in time and therefore are discarded. However, once the DU has enough computational resources to decode all the TBs in time, there is no further benefit in assigning more SM resources to this end, which describes an *inelastic* service. Conversely, co-located AI/ML services, which are competing for the same GPU resources, can improve throughput performance if more SM resources were allocated (see again §III), which describes an *elastic* service.

Following the standard O-RAN architecture, we operate in the Service and Management Orchestrator (SMO) with a time granularity $\delta = 1$ second or higher, where $t = 0, 1, 2 \dots$ denotes the decision periods. The normalized SM allocation for the DU at t is denoted by $a_t \in \mathcal{A} \subseteq [0, 1]$, where $\mathcal{A}$ is a discrete set with all possible allocations; and the set of TBs received by the DU during period t is denoted by $\mathcal{T}_t$.

Now, we let $\phi_t = \Phi(\mathcal{T}_{t-1}, D)$ be a 2-dimensional histogram of the DU contexts (TBs per TTI: SNR and size) that characterize the observed traffic, where D is the number of histogram bins in each dimension. Consequently, we define the system state as $s_t = (\phi_t, a_{t-1}) \in \mathcal{S}$, where $\mathcal{S}$ is the state space.

At each decision period t, an SM allocation a_t is selected. Our goal is to minimize the allocation of SM resources to DU workloads to maximize the performance of the co-located ML service. If configuration a_t differs from the one selected in the previous decision period, i.e., if $a_t \neq a_{t-1}$, the GPU needs to be re-configured, which, as mentioned above, causes ML service disruption during a re-configuration period of duration r. We model this re-configuration cost as:

$$\Delta(s_t, a_t) = \begin{cases} \frac{r}{\delta} \cdot (1 - a_t) & \text{if } a_t \neq a_{t-1} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

where δ corresponds to the duration of a decision period t.

Note that the cost acts as a proxy for the throughput of the AI/ML application, following an inverse correlation with it. Therefore, we model the overall system cost at period t as

$$C(s_t, a_t) = a_t + \Delta(s_t, a_t), \tag{2}$$

We now denote the ratio of TBs timely processed before their latency target at t by $\zeta(s_t, a_t)$, also referred to as *reliability*. Then, we formulate our problem as a finite horizon Markov Decision Process (MDP) with constraints.

$$\begin{aligned} \min_{a_0,\dots,a_{T-1}} \quad & J(s_0; a_0, \dots, a_{T-1}) \\ \text{s.t.} \quad & \gamma^{\tau}\left[\zeta_t(s_t, a_t)\right] > 1 - \epsilon, \qquad \text{for } t = 1, \dots T \end{aligned} \tag{3}$$

where

$$J(s_0; a_0, \dots, a_{T-1}) := \mathbb{E}\left\{ C_T(s_T, a_T) + \sum_{t=0}^{T-1} C_t(s_t, a_t) \right\} \tag{4}$$

is the cost of a sequence of actions $\{a_0, \dots, a_{T-1}\}$ and an initial state s_0. In addition, $C_T(s_T, a_T)$ is the termination cost, a_T is the termination action, $\gamma^{\tau}[Z]$ provides the τ-quantile of

a distribution Z, and ϵ sets the reliability target. Note that the reliability of the system denoted by $\zeta(\cdot)$ is random due to the intrinsic stochasticity of the radio access network. By adjusting the value of τ, we can balance the trade-off between constraint satisfaction probability and cost. In this way, with this formulation, we capture all the important aspects of our problem: (i) the cost of the different SM allocations for the AI/ML application; (ii) the critical reliability constraint of the DU; and (iii) the impact of the re-configurations on the long-term performance of the system.

To obtain the minimum cost and consequently the optimal sequence of actions, we rely on the principle of optimality [25], which states that the optimal sequence of actions for a truncated tail subproblem (e.g., from t' to T) is also optimal for the full problem. Therefore, the optimal cost $J^*(s_t)$ for a given state s_t can be obtained recursively as follows:

$$\begin{aligned} J_t^*(s_t) := \min_{a\in\mathcal{A}} \ & \mathbb{E}\left\{C_t(s_t,a) + J_{t+1}^*(s_{t+1})\right\} \\ \text{s.t.} \quad & \gamma^\tau\left[\zeta_t(s_t,a_t)\right] > 1-\epsilon \qquad \text{for } t=1,\dots T. \end{aligned} \tag{5}$$

where $J_T^*(s_T) := C_T(s_T, a_T)$.

Note that the optimal cost cannot be computed in practice as defined above because future traffic conditions $\phi_{t'}$ for $t' > t$ are unknown in advance. In the next section, we present a control strategy to overcome this limitation.

V. YinYangRAN

We present two algorithms in this section. YinYangRAN-Full, an approximate solution to (3) rooted in certainty equivalent control principle (§V-A); and YinYangRAN-Lite, a simplified version of YinYangRAN-Full suitable when $r/\delta \to 0$ (§V-B). Finally, in §V-C, we present a solution to ensure that the lowest quantiles of the DU reliability performance are above the required target.

A. YinYangRAN-Full (YYR-Full)

To solve the problem defined in eq. (3), we propose an approximate control strategy, named YinYangRAN-Full (YYR-Full), that relies on certainty equivalent control (CEC) [26].

Our solution comprises two phases. In an offline phase, we replace the uncertain metrics (in our case, future traffic demands) with their estimated values. Thus, based on the CEC principle, the problem in eq. (3) becomes deterministic and we can compute an estimation of the optimal cost. Second, during online operation, YinYangRAN-Full sequentially selects an action based on system observations m and the estimation of the optimal cost computed in the offline phase.

1) Offline phase: We define $\tilde{\phi} = \{\tilde{\phi}_0 \dots \tilde{\phi}_{T-1}\}$ as the sequence of expected traffic demands. Using $\tilde{\phi}$ in eq. (5), we compute an estimation of the optimal cost as follows:

$$\begin{aligned} \tilde{J}_t(\tilde{s}_t) := \min_{a_t\in\mathcal{A}} \ & \mathbb{E}\left\{C(\tilde{s}_t,a_t) + \tilde{J}_{t+1}(\tilde{s}_{t+1})\right\} \\ \text{s.t.} \quad & \tilde{\zeta}_t(\tilde{s}_t,a_t,\tau) > 1-\epsilon \qquad \text{for } t=1,\dots T \end{aligned} \tag{6}$$

where $\tilde{J}_T(\tilde{s}_T) := \Delta(\tilde{s}_T, a_T)$, $\tilde{s}_t := (\tilde{\phi}_t, a_{t-1})$, and $\tilde{\zeta}_t(\tilde{s}_t, a_t, \tau)$ is an approximation of the τ quantile of the reliability for the pair $(\tilde{s}_t, a_t)$. The latter can be obtained, for instance, by training a neural network with a quantile loss as we detail in §V-C. The termination cost is defined as the re-configuration cost of a predefined termination action a_T. To accomplish this phase, we use eq. (6) to compute the estimation of the optimal cost for all possible values of $\tilde{s}$, i.e., $(T+1)\times|\mathcal{A}|$ values in total, which are stored in memory.

More formally, the approximation of the optimal cost detailed in eq. (6) has a complexity of $O(T\times|\mathcal{A}|^2)$. Note that, even when we consider a large T, this phase is computed offline and hence its execution time is not a limitation. Nevertheless, if the offline execution time is a limitation for large T values, we propose in §V-B a lighter version, which ignores the re-configuration costs and is suitable when the re-configuration costs are negligible, i.e., when $r/\delta \to 0$.

2) Online phase: During online operation, YYR-Full selects each a_t solving a one-step look-ahead problem:

$$\begin{aligned} \min_{a_t\in\mathcal{A}} \quad & C(s_t,a_t) + \tilde{J}_{t+1}(\tilde{\phi}_{t+1}, a_t) \\ \text{s.t.} \quad & \hat{\zeta}(s_t,a_t,\tau) \geq 1-\epsilon. \end{aligned} \tag{7}$$

Note that the online phase of YYR-Full uses s_t, which includes actual DU traffic load observations ϕ_t. Thus, the current cost is computed accurately according to system observations, and using $\tilde{J}_{t+1}(\cdot)$ we estimate the impact of current actions in the future cost. These two terms provide a computationally efficient farsighted decision-making mechanism.

We make two remarks. First, the CEC strategy obtains optimal results when the future states match their estimations. However, when the observed traffic conditions differ from their expectation, this strategy still provides good empirical results as we show in the next section. Second, eq. (7) can be solved with low complexity as it only requires searching over the action space and therefore its complexity is $O(|\mathcal{A}|)$. This makes YYR-Full suitable for online operation in real systems.

B. YinYangRAN-Lite (YYR-Lite)

Due to the re-configuration cost of our problem, it is very important for our control strategy to be farsighted. An over-adaption to the changes in the traffic load can lead to a large number of re-configurations, whose cost should be taken into account in the design of an efficient control scheme.

However, in some cases, the re-configuration cost can be considered negligible. For example, if the duration of each decision period δ is in the order of minutes; or when the re-configuration time of the GPU r is very small. In these cases, the impact of the future cost on the decision-making tends to zero and we can rely on a simpler version of YYR-Full.

In such a case, at every decision period t, the objective is to select greedily the minimum SM allocation for the DU a_t that satisfies the reliability constraint, that is,

$$\begin{aligned} \min_{a_t\in\mathcal{A}} \quad & a_t \\ \text{s.t.} \quad & \hat{\zeta}(s_t,a_t,\tau) \geq 1-\epsilon. \end{aligned} \tag{8}$$

Note that with this approach, which we name YinYangRAN-Lite (YYR-Lite), the offline phase used by YYR-Full can be omitted as we rely on a greedy version of the online phase.

C. Reliability satisfaction scheme

A critical aspect of our problem is reliability satisfaction because the computing workloads of the DU are inelastic. The challenges of reliability satisfaction are deeply analyzed in §III. In addition, as we operate in time scales of one second or longer, the observed values of reliability for a given traffic load and MPS configuration are stochastic, due to the intrinsic randomness of the RAN (due to, e.g., mobility of the users, app-dependent traffic generation, etc.). Taking this into consideration, the goal is to minimize the probability of missing DU latency targets by learning the quantiles of the reliability function. Thus, we can formulate an SM allocation problem that ensures that the lowest quantiles of the distribution are above the required reliability target.

To this end, we define $F_Z(z)$ as the cumulative distribution function (CDF) of Z. For a given quantile $\tau \in [0,1]$, the value of the quantile function is defined as $q_\tau = F_Z^{-1}(\tau)$. The quantile regression loss is an asymmetric convex function that penalizes overestimation error with weight τ and underestimation error with weight $1-\tau$:

$$\mathcal{L}^\tau(\hat{q}_\tau) := \mathbb{E}_{z\sim Z}\left[\rho_\tau(z-\hat{q}_\tau)\right], \text{ where} \tag{9}$$

$$\rho_\tau(u) := u\cdot(\tau - \Gamma_{\{u<0\}}) \quad \forall u \in \mathbb{R} \tag{10}$$

where $\hat{q}_\tau$ is the quantile function estimation, and $\Gamma_{\{x\}}$ takes value 1 when the condition x is met and 0 otherwise. We let the reliability estimator $\hat{\zeta}(\cdot)$ have N outputs, which approximates the set $\{q_{\tau_1},\ldots,q_{\tau_N}\}$. Thus, we train $\hat{\zeta}(\cdot)$ using stochastic gradient descent to minimize the following joint objective:

$$\sum_{i=1}^{N} \mathcal{L}^{\tau_i}(\hat{q}_{\tau_i}). \tag{11}$$

It is worth mentioning that the quantile regression loss presents a discontinuity at zero, which limits its practical performance when using function approximators such as NNs. To overcome this limitation, we consider the quantile Huber loss [27]. This loss function has an asymmetric squared shape in an interval $[-\kappa,\kappa]$, and reverts to the standard quantile loss outside of this interval:

$$L_\kappa(u) := \begin{cases} \frac{1}{2}u^2 & \text{if } |u| \le \kappa \\ \kappa(|u| - \frac{1}{2}\kappa) & \text{otherwise.} \end{cases} \tag{12}$$

Thus, the asymmetric variation of the Huber loss is

$$\rho_\tau^\kappa(u) := |\tau - \delta_{\{u<0\}}|\frac{L_\kappa(u)}{\kappa}. \tag{13}$$

Finally, the quantile Huber loss can be derived by introducing $\rho_\tau^\kappa(u)$ in eq (9). Note that when κ tends to zero the quantile Huber loss reverts to the quantile regression loss.

VI. Experimental evaluation

System. We implemented an O-RAN system as depicted in Fig. 8. The O-Cloud server is an HP server with an Intel Xeon Gold 6240R CPU at 2.4GHz with 16 cores and an NVIDIA A100 GPU. O-RAN specifies an Acceleration Abstraction Layer (AAL) in between HAs and network functions such as DUs [28]. The AAL abstracts the O-Cloud resources as *Logical Processing Units (LPU)*. We implemented O-RAN's AAL using Intel DPDK's Wireless Baseband Device Library (BBDev)[1]. Like DPDK's solutions for Ethernet, BBDev provides an abstraction for DU tasks through *devices* that can be used as O-RAN LPUs [29]. We implemented two LPUs to execute (i) the GPU DU processor, and (ii) instances of Intel FlexRAN [30] on a CPU pool for fallback DU operation, which is triggered by an LPU broker on top of the LPUs.

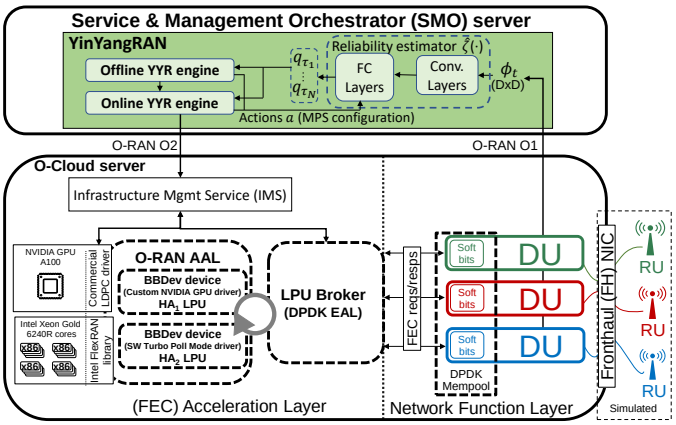

Fig. 8. System implementation.

YinYangRAN controls the O-Cloud by enforcing actions a_t using O-RAN O2 interface, and receives DU load data ϕ_t through O1 interface, as shown in the figure. According to O-RAN, the O-Cloud infrastructure is locally managed by an Infrastructure Management Serivce (IMS). To this end, we implement an interface between the GPU driver and the IMS, to control the allocation of SM resources, and another interface between the LPU Broker and the IMS, to fall back to DU software processing during the GPU re-configuration period.

RUs and UEs are simulated following real-world wireless load patterns like those presented in §III-C. To this end, we encode and modulate the corresponding user data according to the 5G specification and add noise to match the observed patterns. We also let the same ML model we introduced in §III to concurrently use the spare GPU resources of the O-Cloud.

YinYangRAN. We implemented YinYangRAN using Python and PyTorch. For the reliability estimator $\hat{\zeta}(\cdot)$, we use a neural network with two convolutional layers of 8 and 4 filters of dimension 3x3, respectively, and two fully connected layers of 256 units each. The convolutional layers receive as input the traffic characterization ϕ and the fully connected layers receive the output of the convolutional layers plus the selected action a_t. We set the size of the traffic characterization histogram to $D = 5$ and we learn $N = 10$ quantiles. In our evaluations, we consider the lowest quantile $\tau_1 = 0.005$ to be higher than the reliability target, i.e., $\hat{q}_{\tau_1} > 1-\epsilon$. We train $\hat{\zeta}(\cdot)$ offline considering diverse traffic loads and SNR patterns obtained from real-world traces introduced in §III-C.

[1]https://doc.dpdk.org/guides/prog_guide/bbdev.html

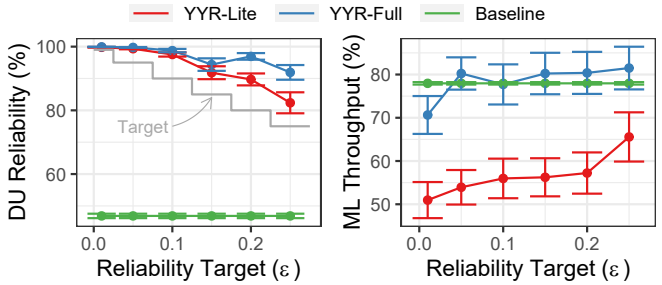

Fig. 9. DU reliability (left) and ML throughput (right) *vs* reliability target ϵ. Comparison between YYR-Full, YYR-Lite, and a baseline.

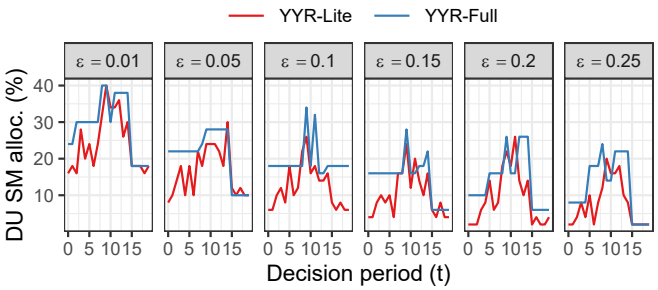

Fig. 10. Temporal evolution of the DU SM allocations made by YYR-Full and YYR-Lite for a subtrace of 20 decision periods.

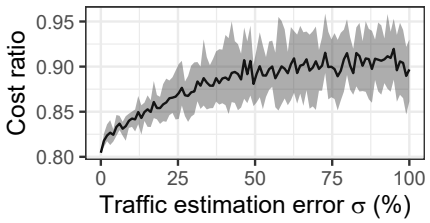

Fig. 11. Ratio of the cost (eq. (2)) of YYR-Full over YYR-Lite as a function of the prediction error. The shaded area cover the 10th and 90th quantiles of the measurements.

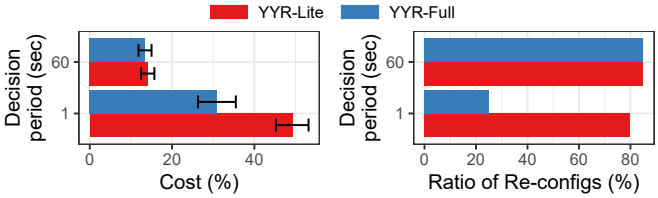

Fig. 12. Impact of the duration of each decision period on cost (left) and rate of re-configurations (right) for YYR-Full and YYR-Lite.

We compare both of our solutions, YYR-Full and YYR-Lite, with a baseline approach that deploys both DU and ML workloads on the same GPU using conventional software concurrency methods, i.e., with no SM isolation between them. Unless otherwise stated, we set $\epsilon = 0.1$, the decision period duration $\delta = 1$ s, we conservatively set $r = 0.5$ s, and present mean performance values with standard errors as error bars.

A. Impact of the reliability target

We first assess the system performance for different reliability targets. To this end, Fig. 9 shows the relative DU reliability (left) and the ML throughput (right) for different targets ϵ as shown in the x-axis. We observe both YYR-Full and YYR-Lite adapt to ϵ, trading off ML throughput as ϵ gets smaller (more stringent target). In contrast, a DU-agnostic baseline solution achieves poor reliability performance as it relies on the default GPU scheduler, which aims to be fair between workloads, which penalizes the inelastic DU demands.

Note that YYR-Full achieves higher ML throughput than YYR-Lite. This is because, though the re-configuration overhead is substantial ($r/\delta = 0.5$), YYR-Full takes into account the joint impact of re-configuring on the instantaneous cost and on the future cost (farsighted decision-making). In contrast, YYR-Lite selects the best action every decision period ignoring the impact of the re-configuration overhead in both the short and long terms (myopic decision-making).

We can confirm this in Fig. 10, which shows the temporal evolution of the DU SM allocations made by our solutions during 20 decision periods of the above experiments. Both YYR-Full and YYR-Lite adapt the allocation of SM resources to the time-varying load, which explains the ML throughput variance shown in Fig. 9 (right), to guarantee meeting the reliability target. However, the myopic nature of YYR-Lite enforces a substantially higher number of re-configurations than YYR-Full, which explains YYR-Lite's throughput loss.

B. Impact of the traffic estimation error

Although the farsighted decision-making of YYR-Full shows better performance than YYR-Lite, it relies on traffic predictions $\tilde{\phi}$ that can deviate from the actual traffic observed in the network. To evaluate this, Fig. 11 depicts the ratio of the cost of YYR-Full over YYR-Lite when we artificially induce different traffic estimation errors. To this end, we add a zero-mean Gaussian error with variance σ^2 to the output of our estimator. The figure shows values of σ relative to the mean load observed. When the traffic estimations are very accurate ($\sigma \to 0$), we observe around 20% cost savings when using YYR-Full with respect to using YYR-Lite. Evidently, these savings reduce as σ increases. However, even when σ is equal to the mean load estimated ($\sigma = 100\%$), YYR-Full achieves around 10% cost savings in average over YYR-Lite.

C. Decision period

In the experiments shown before, YYR-Full shows better performance than YYR-Lite because the latter is a myopic approach that does not consider the cost of frequently re-configuring the GPU. As we discussed in §V, this cost depends on the ratio between re-configuration time r and the duration of each decision period δ. To assess this, we show in Fig. 12 the cost and the ratio of re-configuration (percentage of time periods that the algorithms re-configures the GPU) for two decision lengths δ: one second (as in the previous experiments), which sets $r/\delta = 1/2$ and one minute, which sets $r/\delta = 1/120$. As expected, when we $\delta = 1$ min, YYR-Lite provides the same performance as YYR-Full because the impact of greedily re-configuring the GPU every decision period is negligible in this case.

D. GPU Capacity

Finally, we evaluate the impact of the GPU capacity (in terms of the number of available SMs) on the system performance for all the solutions. Fig. 13 depicts the DU reliability (left) and the ML throughput (right) as a function of the GPU's available number of SMs. Remarkably, YYR-Full and YYR-Lite attains the reliability target ($\epsilon = 0.01$) in all cases at the cost of ML throughput losses. The baseline solution also sacrifices ML throughput when the GPU capacity shrinks, but it sacrifices DU reliability as well without any prioritization.

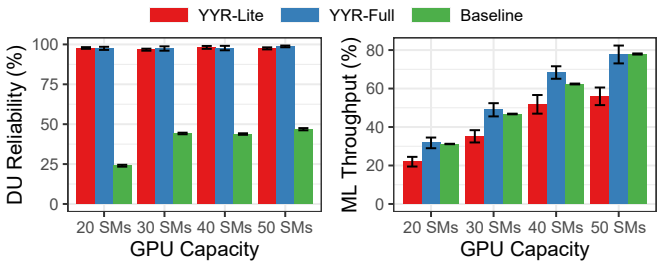

Fig. 13. Impact of the GPU computational capacity on DU reliability (left) and ML throughput (right).

VII. Related Work

A. O-RAN Control and Virtualization

The advent of O-RAN has motivated substantial research on radio control and management. OrchestRAN [31], for instance, is a notable solution capable of orchestrating data-driven models in the O-RAN context, offering intent-based control operations for mobile operators. This orchestration tool has been successfully validated through ColO-RAN [32], a comprehensive testbed equipped with software-defined radios-in-the-loop. The authors of [14] present an O-RAN control algorithm that employs Bayesian learning theory to derive energy-efficient radio policies. It was subsequently extended in [33] to account for co-located edge services. However, they assume dedicated and statically pre-assigned computing resources for both the base station and edge services.

The concept of shared computing resources is not new. For example, Nuberu [16] is a DU design that ensures reliability on virtualized platforms, balancing reliability with network delay. Nevertheless, it lacks efficient mechanisms for sharing computing resources within the platform. Some research has investigated CPU resource sharing in the context of vRANs. vrAIn [15] utilizes a deep deterministic policy gradient (DDPG) to allocate CPU resources across multiple base stations; and Concordia [8] employs a quantile decision tree for CPU resource sharing between a base station and third-party applications. However, these works do not address sharing of hardware accelerators, which have become the industry-standard approach for processing PHY workloads.

Other related studies include GPF [34], an ultra-fast ($\sim 100\mu$s) GPU-accelerated radio resource scheduler, and [35], an approach for sharing RU front-ends.

B. GPU Sharing

Recently, NVIDIA's MIG technology has been the focus of extensive attention due to its capability to create rigid partitions of GPU resources (see §II). An exhaustive characterization, in terms of performance and energy consumption with various state-of-the-art deep learning models, is provided in [36]. However, as evidenced in §III, MIG struggles with the dynamic multiplexing of GPU resources due to high reconfiguration overheads.

In response, MISO [37] uses MPS to predict the most effective MIG partitions for diverse jobs, thereby mitigating the challenge of exploring different partitions with MIG alone. Yet, MISO fails to address the issue of re-configuring slices for dynamically changing workloads. Both Salus [38] and TGS [39] offer lightweight GPU sharing. Salus enables two primitives: quick job switching and memory sharing, fostering fine-grained GPU sharing among applications. TGS provides transparent GPU sharing among deep learning training workloads in container clouds. However, these solutions do not adequately handle the two fundamentally disparate workloads generated by DUs (inelastic) and by ML tasks (elastic).

NVIDIA Aerial is a promising avenue for accelerating DU workloads using GPUs [11]. In collaboration with NVIDIA, NTT Docomo is on track to initiate GPU-accelerated 5G trials this year, and SoftBank has recently piloted a proof-of-concept around sharing GPU resources between 5G DU workloads and ML-based edge computing applications [10]. Despite this, they do not yet propose viable brokering solutions for online sharing of GPU resources between virtualized DUs and third-party applications like those running ML models.

VIII. Conclusions

GPU-accelerated general-purpose computing platforms can process the PHY-layer workload of 5G Distributed Units concurrently with Machine Learning (ML) operations. This approach promises to increase flexibility and cost-effectiveness compared to conventional RAN virtualization techniques based on ASICs or FPGAs. However, blindly sharing GPU resources using conventional sharing methods can severely disrupt the reliability of 5G processors, which generate inelastic loads, especially when balancing these with ML tasks, which are elastic. In response, we have developed YinYangRAN, an innovative O-RAN-compliant solution that manages GPU-based HAs, ensuring 5G processing reliability and maximizing the throughput of concurrent ML services.

YinYangRAN employs an efficient approximate dynamic programming technique for GPU resource allocation, informed by a neural network trained on real-world measurements. Tests with real RAN workloads show that YinYangRAN significantly outperforms traditional GPU sharing models, achieving over 50% higher reliability with minimal impact on ML tasks.

Acknowledgements

This work has been supported by the European Commission through Grant No. SNS-JU-101097083 (BeGREEN), 101139270 (ORIGAMI), and 101017109 (DAEMON), and by MINECO/SETELECO and NextGeneration-EU through UNICO I+D Grants 2022/0005395 (CLARION) and TSI-063000-2021-52 (AEON-ZERO), and CERCA Programme.

REFERENCES

[1] A. Garcia-Saavedra and X. Costa-Pérez, "O-RAN: Disrupting the Virtualized RAN Ecosystem," *IEEE Communications Standards Magazine*, vol. 5, no. 4, pp. 96–103, 2021.

[2] Heavy Reading, "Heavy Reading's Accelerating Open RAN Platforms Operator Survey," *White Paper*, June 2021.

[3] RCR Wireless News, "From greenfield to brownfield: Open RAN in 2022 (With large scale carrier commitments in place, what's next for the Open RAN ecosystem?)," *Editorial Report*, October 2021.

[4] ABI Research, "Open RAN." Market Data Report, 2020.

[5] Dell'Oro Group, "Advanced Research Reports on Open RAN." Report, July 2022.

[6] Silicom, "Silicom's eASIC ACC100 FEC Accelerator," Nov. 2022.

[7] Intel, "Enabling 5G Wireless Acceleration in FlexRAN: for the Intel® FPGA Programmable Acceleration Card N3000," 2020.

[8] X. Foukas and B. Radunovic, "Concordia: Teaching the 5g vran to share compute," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, (New York, NY, USA), p. 580–596, Association for Computing Machinery, 2021.

[9] Soma Velayutham, "NVIDIA CEO Introduces Aerial — Software to Accelerate 5G on NVIDIA GPUs," Oct. 2019.

[10] Keith Dyer, "AI everywhere all the time," 2023.

[11] A. Kelkar and C. Dick, "Nvidia aerial gpu hosted ai-on-5g," in *2021 IEEE 4th 5G World Forum (5GWF)*, pp. 64–69, 2021.

[12] J. Lee, S. Lee, J. Lee, S. D. Sathyanarayana, H. Lim, J. Lee, X. Zhu, S. Ramakrishnan, D. Grunwald, K. Lee, and S. Ha, "Perceive: Deep learning-based cellular uplink prediction using real-time scheduling patterns," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, (New York, NY, USA), p. 377–390, Association for Computing Machinery, 2020.

[13] A. Banchs, M. Fiore, A. Garcia-Saavedra, and M. Gramaglia, "Network intelligence in 6g: Challenges and opportunities," in *Proceedings of the 16th ACM Workshop on Mobility in the Evolving Internet Architecture*, pp. 7–12, 2021.

[14] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "Bayesian online learning for energy-aware resource orchestration in virtualized rans," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pp. 1–10, IEEE, 2021.

[15] J. A. Ayala-Romero, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, A. Banchs, and J. J. Alcaraz, "vrain: A deep learning approach tailoring computing and radio resources in virtualized rans," in *The 25th Annual International Conference on Mobile Computing and Networking*, pp. 1–16, 2019.

[16] G. Garcia-Aviles, A. Garcia-Saavedra, M. Gramaglia, X. Costa-Perez, P. Serrano, and A. Banchs, "Nuberu: Reliable ran virtualization in shared platforms," in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, MobiCom '21, (New York, NY, USA), p. 749–761, Association for Computing Machinery, 2021.

[17] O-RAN Alliance, "Cloud Architecture and Deployment Scenarios for O-RAN Virtualized RAN (O-RAN.WG6.CADS-v04.00) ." Technical Report, Oct. 2022.

[18] Y. Blankenship, D. Hui, and M. Andersson, *Channel Coding in NR*, pp. 303–332. Cham: Springer International Publishing, 2021.

[19] 3rd Generation Partnership Project (3GPP), "3GPP TR 38.913; Technical Specification Group Radio Access Network; Study on Scenarios and Requirements for Next Generation Access Technologies; (Release 17)." Technical Report, 2022.

[20] E. A. Papatheofanous, D. Reisis, and K. Nikitopoulos, "Ldpc hardware acceleration in 5g open radio access network platforms," *IEEE Access*, vol. 9, pp. 152960–152971, 2021.

[21] L. Lo Schiavo, M. Fiore, M. Gramaglia, A. Banchs, and X. Costa-Perez, "Forecasting for network management with joint statistical modelling and machine learning," in *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pp. 60–69, 2022.

[22] R. Falkenberg and C. Wietfeld, "FALCON: An accurate real-time monitor for client-based mobile network data analytics," in *2019 IEEE Global Communications Conference (GLOBECOM)*, (Waikoloa, Hawaii, USA), IEEE, Dec. 2019.

[23] N. Ludant, P. Robyns, and G. Noubir, "From 5g sniffing to harvesting leakages of privacy-preserving messengers," in *2023 2023 IEEE Symposium on Security and Privacy (SP) (SP)*, (Los Alamitos, CA, USA), pp. 3146–3161, IEEE Computer Society, may 2023.

[24] J. Ding, R. Doost-Mohammady, A. Kalia, and L. Zhong, "Agora: Real-time massive mimo baseband processing in software," in *Proceedings of the 16th international conference on emerging networking experiments and technologies*, pp. 232–244, 2020.

[25] D. Bertsekas, *A Course in Reinforcement Learning*. Athena Scientific, 2023.

[26] D. Bertsekas, *Dynamic programming and optimal control: Volume I*, vol. 4. Athena scientific, 2012.

[27] P. J. Huber, "Robust estimation of a location parameter," *Breakthroughs in statistics: Methodology and distribution*, pp. 492–518, 1992.

[28] O-RAN Alliance, "O-RAN Acceleration Abstraction Layer – General Aspects and Principles (O-RAN.WG6.AAL-GAnP-v04.00) ." Technical Specification, Oct. 2022.

[29] O-RAN Alliance, "O-DU Low Project Introduction," 2022.

[30] Intel, "FlexRAN LTE and 5G NR FEC Software Development Kit Modules," May 2019.

[31] S. D'Oro, L. Bonati, M. Polese, and T. Melodia, "Orchestran: Network automation through orchestrated intelligence in the open ran," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 270–279, 2022.

[32] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Colo-ran: Developing machine learning-based xapps for open ran closed-loop control on programmable experimental platforms," *IEEE Transactions on Mobile Computing*, pp. 1–14, 2022.

[33] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Pérez, and G. Iosifidis, "Edgebol: Automating energy-savings for mobile edge ai," in *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*, pp. 397–410, 2021.

[34] Y. Huang, S. Li, Y. T. Hou, and W. Lou, "Gpf: A gpu-based design to achieve 100 μs scheduling for 5g nr," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, (New York, NY, USA), p. 207–222, Association for Computing Machinery, 2018.

[35] J. Mendes, X. Jiao, A. Garcia-Saavedra, F. Huici, and I. Moerman, "Cellular access multi-tenancy through small-cell virtualization and common rf front-end sharing," *Computer Communications*, vol. 133, pp. 59–66, 2019.

[36] B. Li, V. Gadepally, S. Samsi, and D. Tiwari, "Characterizing multi-instance gpu for machine learning workloads," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 724–731, IEEE, 2022.

[37] B. Li, T. Patel, S. Samsi, V. Gadepally, and D. Tiwari, "Miso: exploiting multi-instance gpu capability on multi-tenant gpu clusters," in *Proceedings of the 13th Symposium on Cloud Computing*, pp. 173–189, 2022.

[38] P. Yu and M. Chowdhury, "Fine-grained gpu sharing primitives for deep learning applications," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 98–111, 2020.

[39] B. Wu, Z. Zhang, Z. Bai, X. Liu, and X. Jin, "Transparent {GPU} sharing in container clouds for deep learning workloads," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 69–85, 2023.