Concordia: Teaching the 5G vRAN to Share Compute

Xenofon Foukas
Microsoft
Cambridge, United Kingdom
xefouk@microsoft.com

Bozidar Radunovic
Microsoft
Cambridge, United Kingdom
bozidar@microsoft.com

ABSTRACT

Virtualized Radio Access Network (vRAN) offers a cost-efficient solution for running the 5G RAN as a virtualized network function (VNF) on commodity hardware. The vRAN is more efficient than traditional RANs, as it multiplexes several base station workloads on the same compute hardware. Our measurements show that, whilst this multiplexing provides efficiency gains, more than 50% of the CPU cycles in typical vRAN settings still remain unused. A way to further improve CPU utilization is to collocate the vRAN with general-purpose workloads. However, to maintain performance, vRAN tasks have sub-millisecond latency requirements that have to be met 99.999% of times. We show that this is difficult to achieve with existing systems. We propose Concordia, a userspace deadline scheduling framework for the vRAN on Linux. Concordia builds prediction models using quantile decision trees to predict the worst case execution times of vRAN signal processing tasks.The Concordia scheduler is fast (runs every 20 μs) and the prediction models are accurate, enabling the system to reserve a minimum number of cores required for vRAN tasks, leaving the rest for general-purpose workloads. We evaluate Concordia on a commercial-grade reference vRAN platform. We show that it meets the 99.999% reliability requirements and reclaims more than 70% of idle CPU cycles without affecting the RAN performance.

CCS CONCEPTS

• **Networks → Mobile networks**; **Wireless access points, base stations and infrastructure**; *Network reliability*; **Cloud computing**; • **Computer systems organization → Real-time systems**; • **Computing methodologies → Machine learning**.

KEYWORDS

vRAN, 5G, mobile networks, edge computing, NFV, real-time scheduling, machine learning, prediction model

ACM Reference Format:
Xenofon Foukas and Bozidar Radunovic. 2021. Concordia: Teaching the 5G vRAN to Share Compute. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–27, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3452296.3472894

1 INTRODUCTION

The Radio Access Network (RAN) is a part of the cellular network infrastructure that includes base stations (cells), and is responsible for converting data packets into wireless radio waveforms and back. It includes wireless physical layer operations that perform various complex signal processing tasks. Conventional mobile base stations include specialized hardware boxes called BaseBand processing Units (BBUs), implementing the physical layer of each cell.

A forthcoming 5G trend is to extend network function virtualization (NFV) to the radio access network (vRAN), to deploy RAN workloads on commodity hardware. Many benefits of virtualization apply to the vRAN: vendor lock-in mitigation, flexible upgrades, rapid roll-out of new standards and services and a potential for cost reduction. This trend is real and several operators have deployed or are deploying vRANs [42, 84, 113, 114], creating a market for hundreds of thousands of servers and millions of CPU cores [71], that is projected to claim more than $3 billion by 2025 [95] and more than $6 billion by 2030 [94]. A new cellular operator in Japan, Rakuten, is running its entire vRAN for a national cellular network on commodity hardware [19], and the new US green-field operator Dish plans to do the same [34].

Unlike other virtualized network appliances, vRAN signal processing algorithms are very compute intensive. Virtualized BBUs running these algorithms can consume more than 60% of the overall required compute resources of the vRAN [39, 107, 118]. Given the huge scale of network deployments, it is important to reduce the computational cost of virtualized BBUs.

A common way to increase vRAN efficiency is *RAN pooling* (or BBU pooling), which involves sharing compute resources among several cells. The virtualization and pooling of RAN tasks takes advantage of the statistical multiplexing gains of cells [20, 104, 116]. For example, one of the most demanding RAN tasks is decoding [72] and its computational load is proportional to the wireless traffic volume. If a RAN pool serves cells in both a residential and an office area, the peak throughput (and thus the compute requirement) is likely to stay similar throughout the day, as users move between offices and homes [47].

However, existing RAN pooling schemes only leverage long-term (e.g. diurnal) changes in traffic demand and other opportunities for statistical multiplexing at much lower time scales are not yet explored. For example, our measurements described in Section 2.2 look at a RAN pool with 3 cells and show that the median traffic volume per slot is 0.2KB. The 99th percentile traffic volume per slot is 2.5 KB, which is more than 10× larger than the median. As one needs to provision the RAN pool compute capacity for the peak traffic, the pool will be substantially underutilized for most of the time. Furthermore, in a common example of a multi-cell 100MHz deployment configured for time division multiplexing (discussed in Section 2.2), we see more than 50% of the CPU cores assigned to the RAN pool being left unutilized even at the peak cell traffic, due to the difference in the computational demands of uplink and downlink signal processing. We observe similar sharing opportunities in other common use cases.

One obvious way to mitigate such inefficiencies would be to deploy other general-purpose (and best-effort) workloads on the CPUs when they are not fully utilized by the RAN. One such example is ML training and classification workloads (e.g. video analytics) that have to run at the edge due to privacy concerns or due to low latency requirements [58, 108, 119, 120]. Another example are local content caching and delivery workloads, deployed by third-parties or by the operators, with the goal of minimizing the user latency and reducing the backhaul traffic strain [11, 77, 102, 109]. Furthermore, in the context of cellular networks, example workloads could include network functions relevant to the higher layers of the RAN protocol stack and the cellular core (e.g. CU control and data plane, 5G UPF), as well as management and control functions (e.g. for vRAN orchestration, monitoring and analytics) [54, 77, 85, 100]. Collocating such functions at the vRAN edge instead of using the hyperscale cloud could be particularly beneficial in the context of private LTE/5G networks, to enable cost-efficient and autonomous edge deployments and/or to mitigate privacy concerns regarding the cellular data of users [30, 31, 54, 77].

The collocation problem and its performance effects has been studied extensively in the literature (e.g. [13, 65, 67, 68, 75, 78, 81, 83]). The particular challenge with RAN tasks is their very stringent timing constraints – *with task deadlines in 10s or 100s of microseconds* – where every deadline violation can cause a service degradation to the end user – *imposing a standard requirement that deadlines have to be met 99.999% of time*. On the one hand, most of the existing solutions have been designed with tail latencies that are insufficient for the RAN [13, 65, 75, 81, 83]. On the other hand, solutions that do provide microsecond level of control (e.g. [51]) require applications to run in non-standard operating systems or using specific APIs, making them incompatible with conventional workloads, like containers, running alongside the RAN. As a result, and in order to mitigate the problem of tail latency and achieve the desired RAN performance, a standard practice is to isolate the RAN from other workloads as much as possible by dedicating cores (c.f. [21, 74]) and effectively waste idle CPU cycles.

To address this problem we built *Concordia*, a system that recovers unused CPU cycles in vRAN pools for general workloads without violating the strict timing requirements of vRAN pool tasks. The Concordia design views the vRAN as the high priority workload, with a maximum scheduling priority. All other workloads are considered as best-effort and as such can be pre-empted by the vRAN at any point in time. To achieve this, Concordia uses a userspace deadline scheduler that leverages ideas from the mixed-criticality systems space [17]. The scheduler is fed with predictions of the worst-case execution time (WCET) of each RAN task. It uses the predictions to calculate and *proactively* reserve the least number of cores required to perform the vRAN pool operation in the next slot (e.g. 1ms), releasing the rest of the cores to the OS for other tasks. This is done at a 20 μs granularity, allowing *Concordia* to adjust the scheduling decision faster than RAN traffic fluctuations and compensating for unpredictable OS scheduling latencies that exist in a non real-time OS such as Linux.

A key requirement of Concordia is an accurate estimate of the RAN tasks' WCETs. Predicting WCETs has been extensively studied in the context of both hard and soft real-time systems [18, 111]. A common assumption of such works is that each task can be characterized with a single WCET prediction value, without any parameterization. In contrast, the runtime of a RAN task (and thus its WCET) can vary significantly depending on several tens of input parameters (e.g. relevant to the traffic load, cell configuration, etc), as quantified in Section 4. We show that previous works, ignoring this parameterization, lead to overly pessimistic scheduling and poor CPU utilization.

To overcome this limitation, Concordia proposes a novel ML-based WCET parameterization and prediction method that is composed of an offline and an online phase. During the offline phase (vRAN deployed in isolation), Concordia constructs a quantile decision tree for each RAN task, classifying different WCET predictions into leaf nodes depending on the tasks' input parameters to minimize the variance of collected WCET samples in each leaf. The predictions are further updated and improved online (in the presence of other workloads), using a fast online approximation method that compensates for the contention (e.g. on the cache) caused by the collocated workloads. To our knowledge, this is the first work that provides such a parametrized WCET prediction mechanism.

We implement *Concordia* on top of Intel FlexRAN v20.02 [48, 62], a state-of-the-art 4G and 5G reference implementation that is used in most of today's commercial vRAN deployments (c.f. [84, 113, 114]). We evaluate it by collocating various 5G RAN traffic workloads with best-effort workloads that are representative of envisioned collocation scenarios (Nginx, Redis for content caching, SQL for cellular core and content caching and MLPerf for ML training). We show that we can recover up to 70% of unused CPU cycles while maintaining the operational requirements of the RAN. To the best of our knowledge, this is the first system that allows 5G vRAN to allow other workloads to recover unused vRAN CPU cycles.

In summary, we make the following contributions:

(1) We design Concordia, a userspace vRAN task scheduling framework that allows general purpose workloads to run in parallel without affecting the RAN performance. The design leverages the observation that the WCET of vRAN tasks can be predicted with high confidence to estimate the required number of CPU cores. It also continuously adapts its estimation to release unused cores for other workloads (§ 3).
(2) We develop a novel machine learning method for the parameterized prediction of the WCET of signal processing tasks using quantile decision trees.The model further adapts its WCET predictions at runtime to the observed RAN traffic load and the system-level contention from other workloads (§ 4).
(3) We build Concordia based on the reference vRAN solution of Intel FlexRAN v20.02 [48, 62] (§ 5). Our evaluation on 5G vRAN cells with realistic collocated workloads (§ 6) shows that Concordia can provide 99.999% reliability in meeting RAN processing deadlines, while reclaiming up to 70% of the CPU cores.
This work does not raise any ethical issues.

2 BACKGROUND & MOTIVATION

2.1 vRAN overview

vRAN operations and requirements: Radio transmissions and receptions in vRAN occur in regular Transmission Time Intervals (TTIs) or *slots*. Depending on the cell configuration, a slot can last between 62.5us and 1ms [1]. A set of signal processing tasks have to be processed in each slot, starting at the beginning of the slot

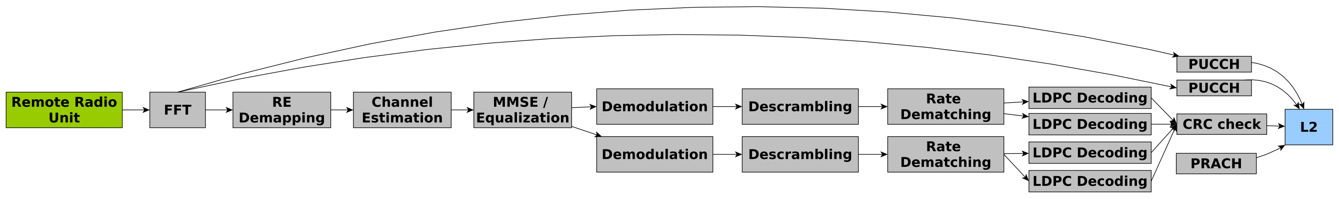


Figure 1: Example of uplink signal processing tasks DAG for 5G NR.

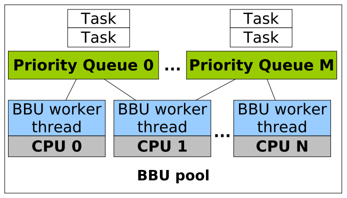
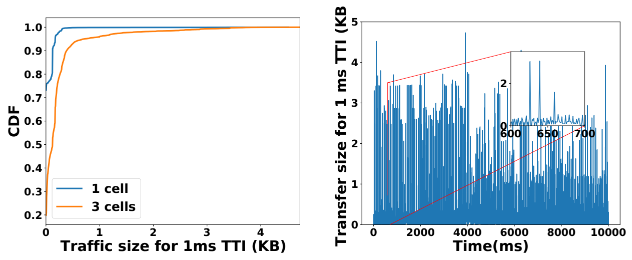


Figure 2: High-level overview of vRAN pool design.



(a) CDF of single cell and 3 cells aggregate **(b) Traffic fluctuation in 3 cells aggregate**

Figure 3: LTE cell traffic characteristics.

and having to finish by the end of the same or a subsequent slot (depending on the implementation).

The dependencies of the signal processing tasks executed within a slot can be described with Directed Acyclic Graphs (DAGs). Fig 1 illustrates such a (simplified) DAG for the case of 5G uplink, with each shaded node corresponding to a different signal processing task (see Appendix A.1 for a 5G downlink example and a brief description of the most significant tasks). For example, LDPC decoding uses the output of rate dematching and cannot start before the dematching has finished. The exact DAG structure depends on various input parameters. There can be multiple active DAGs at any time (e.g. an Rx and a Tx DAG, or DAGs from adjacent slots), and tasks from the same DAG can run in parallel (e.g. multiple LDPC decoding operations on different cores). These DAGs have deadlines and if a vRAN pool fails to process a DAG by a given deadline, the packets transmitted or received in the corresponding time slot are dropped. As some of them can carry control information, the impact of a loss can also affect a long term state of the user connection. For this reason, it is standard practice to impose 99.999% of reliability [29, 112].

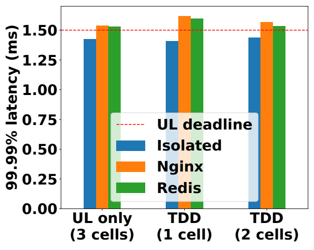
vRAN implementation: A typical vRAN implementation uses a queue-based worker thread model (c.f. [75]) for processing signal processing tasks (Fig 2). Such a design is used (in variations) in most existing vRAN implementations, including Intel's FlexRAN [48, 62], OpenAirInterface [43, 52] and Agora [28]. Here we describe FlexRAN as a concrete example we use throughout the paper. The vRAN pool is composed of a number of worker threads, each pinned to a CPU core. Each signal processing task is assigned to a priority queue, waiting to be processed by a worker thread. The vRAN pool can support more than one priority queues and each worker thread can be associated with one or more queues, allowing a fine-grained control of the assignment of signal processing tasks to CPU cores. To minimize the latency, the worker threads are typically configured to use a high priority scheduling policy (e.g. SCHED_FIFO in Linux) that can only be preempted by the highest priority kernel threads (e.g. watchdog threads). Each worker thread checks the priority queue(s) and picks the earliest deadline task (Earliest Deadline First - EDF). Once the task processing is finished, the worker thread generates zero or more new tasks according to the corresponding DAG model. The worker thread can keep one of the generated tasks to process next for improved cache efficiency, while the rest are placed back in the priority queue to be picked up by another worker. If the queue is empty, a worker thread can choose to either busy wait to minimize the latency (leading to 100% core utilization) or to yield, allowing other workloads to run. Once more tasks are generated (e.g. by another worker thread), a sleeping thread is signaled to wake up and restart processing.

2.2 Sharing opportunities

A common practice is to pool requests from multiple cells on the same vRAN pool to exploit statistical multiplexing. However, cell traffic is bursty in nature at much finer time scales, due to a number of factors (e.g., number of active users, their signal quality, the behavior of higher layer protocols, etc [6, 16, 53]). Therefore, pooled traffic is bursty even when aggregated. This can be observed in Fig. 3, for a 10s snippet of an 1 hour uplink traffic trace captured during rush hour (around 12pm) from three neighboring LTE cells in the area around the central train station of Cambridge UK, using the Falcon sniffer [33]. We see that the changes in the traffic size happen at a millisecond time scale. Moreover, a single cell is completely idle 75% of the TTI slots. If a vRAN pool aggregates 3 cells, it is only idle 20% of the TTI slots, but still mostly processes short packets and a median transfer size per slot is 0.2KB, which is 10× less than the 95th percentile. If we provision the vRAN pool compute capacity for peak traffic, it will be substantially underused most of the time. We verify the same happens for the entire hour we measured. Similar observations are drawn from the works and traffic traces in [6, 104].

Config	# cores	Avg CPU util (%)
UL only (3 cells)	4	42
TDD (1 cell)	5	38
TDD (2 cells)	12	33

(a) vRAN CPU utilization (UL – uplink, TDD – standard 5G time division between UL/DL)



(b) Slot processing deadline violations

Figure 4: vRAN CPU utilization and interference effects

To get an intuition of what happens with larger pool sizes, consider n cells, each with transfer sizes modeled as a simple Gaussian $\mathcal{N}(\mu, \sigma^2)$. The aggregate traffic is then $\mathcal{N}(n\mu, n\sigma^2)$, with the average traffic growing linearly and the variance growing as a square root. The peak to average ratio diminishes with n, but the actual wasted CPU cycles are proportional to the standard deviation (the difference between peak and average), and grow proportionally with $\sqrt{n}$. Thus, the problem persists even in the ideal pooling case with very large pools and uniform traffic per cell, something that rarely happens in practice.

Further statistical multiplexing opportunities arise in the common 5G deployment scenario of time division multiplexing, due to the significant difference in the compute requirements of the uplink and downlink processing [72, 107]. To quantify this, we set up a vRAN pool using Intel FlexRAN v20.02 on an optimized server (described in Section 6). We deploy workloads similar to the one reported above, varying the number of cells and the type of traffic. For each case we measure the CPU utilization and the minimum number of CPU cores required to process the peak traffic. As shown in Fig 4a, the average utilization of the required cores for any of the scenarios under study is at most 42%.

The aforementioned observations, along with the fact that the traffic load of cells can greatly fluctuate throughout the day, as various studies of real mobile networks have revealed [104, 117], creates an opportunity for significantly improving the utilization of edge servers, by sharing the compute resources left idle by the vRAN with other collocated workloads.

2.3 Challenges in sharing vRAN

To exploit the sharing opportunities described above, a vRAN pool has to meet deadlines with high reliability. This is challenging on a general purpose compute environment, even optimized for low-latency [21]. To illustrate this, we study the scenarios of Fig. 4a and measure how the vRAN processing latency is affected by other workloads. We consider three cases; (i) the vRAN pool is running in isolation (recommended FlexRAN configuration [49]), (ii) two Nginx servers are running in containers on the same CPU cores as the vRAN pool, saturated with HTTP requests and (iii) two Redis containers are running on the same cores as the vRAN pool, saturated with GET/SET operations. In all cases, we use the default FlexRAN setup where the vRAN workload is running with maximum real-time priority and the other workloads are running only when a vRAN pool worker thread is idle and yields.

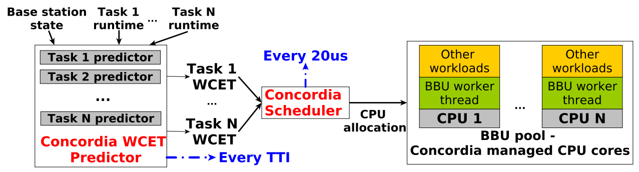


Figure 5: High-level design of Concordia

For each case we run a 5 minutes experiment and measure the 99.99% processing latency of the signal processing tasks. The DAG deadline is set to 1.5ms following the requirement of 5G enhanced Mobile Broadband services (eMBB) for a one way processing delay below 4ms [32], including MAC/RLC processing and fronthaul transport delay. As shown in Fig 4b, the processing latency is below the deadline for all the cell deployment scenarios in the isolated case. However, the tail latency significantly increases with the introduction of other workloads, violating the required 99.999% reliability.

The tail latency increase occurs for two main reasons:

Scheduling latency The Linux kernel can introduce latencies that, depending on the kernel configuration, can vary from tens of microseconds to tens of milliseconds [66, 88]. The main reason is that parts of the kernel are non-preemptible (even with real time patches) [88]. Therefore, the high priority vRAN worker threads can be delayed from reclaiming a CPU core once they yield if the kernel has taken control (e.g. due to interrupts, RCU operations or system calls from a workload sharing the core). We quantify these effects in Section 6.2.

Cache interference Multiple studies have shown that the performance of collocated workloads can be severely affected by uncontrolled cache interference [26, 27, 41, 56, 92]. In the case of the Last Level Cache (LLC), which is shared among cores, workloads do not even have to be collocated on the same core for performance degradation to occur. We measure and discuss these effects in Section 6.2.

For these reasons, a standard operational practice is to isolate the vRAN from other workloads as much as possible by dedicating cores and LLC cache (e.g. as recommended by the OpenNESS project [74] for FlexRAN [21]) though most of today's deployments run on completely isolated servers.

3 SYSTEM DESIGN

An overview of Concordia is shown in Fig 5. It is composed of the *Concordia WCET predictor* and the *Concordia scheduler*.

Concordia WCET predictor: The key component of Concordia is a novel predictor that provides an accurate WCET prediction of each RAN task in a DAG. At the beginning of each TTI slot, the predictor takes as input a set of features X describing the state of the base station (e.g. number of scheduled UEs and their transport block sizes, number of layers, etc.). For each signal processing task that has to execute in the slot, the predictor evaluates its individual prediction model and sends a WCET prediction to the Concordia scheduler based on the input features X. Once the tasks are executed, the predictor uses the observed runtimes to improve the model for subsequent slots by adjusting predictions depending on the impact of collocated workloads, effectively dealing with the problem of cache interference. All this is discussed in detail in Section 4.

Concordia scheduler: Concordia leverages ideas from the mixed-criticality literature to perform its scheduling(see [17] for a comprehensive survey). It treats the vRAN as the high-priority workload, and all other workloads as best-effort. Such workloads are allowed to use the CPU cores which have not been allocated to the vRAN DAGs, or the ones released by the vRAN DAGs due to early completion. Concordia uses the state-of-the-art federated scheduling algorithm for parallel tasks (DAGs) from [61] to decide the number of cores to allocate to the vRAN tasks. In a nutshell, the algorithm uses the longest path of the DAG and the predicted WCET of each DAG task to estimate the predicted execution time and whether to increase or decrease the number of cores allocated to the RAN, based on the deadline of the DAG. If the remaining time until the DAG deadline is too small, the algorithm gets into a critical stage where it allocates all cores to the RAN, evicting all best-effort workloads. We defer the reader to Table 3 of [61] for details on the Concordia core allocation strategy. The predicted WCETs are not always sufficiently accurate to ensure that a RAN DAG will meet its deadline. Also, some CPU cores might take longer to wake up when scheduled (e.g., due to the scheduling latency issue discussed in Section 2.3). To improve on these mispredictions, the Concordia scheduler updates its decisions every 20 μs.

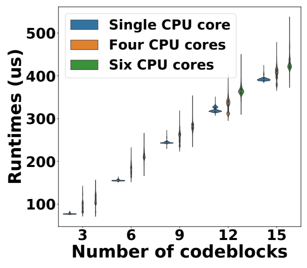
4 CONCORDIA WCET PREDICTOR

We begin by discussing the challenges of parameterized prediction for the vRAN tasks' WCET. We then present the novel parameterized WCET predictor we designed for Concordia.

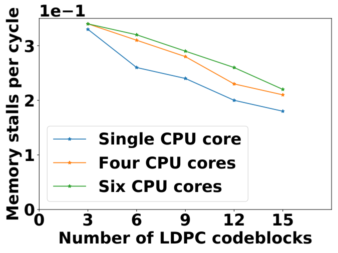
4.1 vRAN tasks WCET prediction challenges

We illustrate and quantify the WCET prediction challenges on the example of 5G (LDPC) decoding, since according to our measurements (see Appendix A.1) it is the most expensive task and can consume more than 60% of the total uplink processing time and more than 50% of the total processing time (both uplink and downlink). However, we have verified that the same observations and conclusions apply to other significant tasks, like encoding (>40% of the downlink processing), channel estimation (>8% of the uplink processing), equalization (>5% of the uplink processing) and modulation/precoding (>25% of the downlink processing). A holistic view of the system that includes all the signal processing tasks is studied in more depth in Section 6.

1. Parameterized task runtime prediction is non-linear: Both the average times and WCETs of signal processing tasks often linearly increase with the input size [40, 103]. However, other parameters, such as the number of CPU cores or the SNR and link adaptation of the mobile users, may have a non-linear impact on the execution times. This is illustrated in Fig. 6a for the case of 120K LDPC decoding operations on a group of codeblocks (8448 bits per codeblock). All operations are generated on a single CPU core. While the runtime depends linearly on the number of LDPC codeblocks, the dependence on the number of CPU cores is not linear. When the data is decoded across multiple cores (cases of 4 and 6 CPU cores in Fig. 6a), the decoding core needs to fetch the required data, causing CPU memory stalls (Fig 6b). This can increase the WCET by up to 25% from the single core case. The exact overhead depends on multiple factors, including the number of UEs transmitting/receiving data, the transport block size, the level of parallelization supported by the vRAN implementation, the number of CPU cores etc. Similar observations



(a) Runtime violin plot



(b) Memory stalls

Figure 6: Runtime characteristics for LDPC decoding for different codeblock assignments.

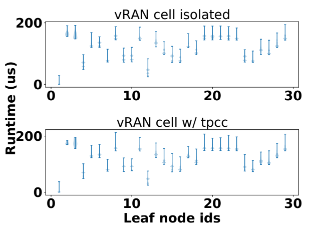
have been made in the literature for the piecewise-linear effect of the mobile user SNR and link adaptation to decoding (e.g. [5, 12, 89]).

2. vRAN task runtimes are affected by cache interference: The cache interference caused by collocated workloads (Section 2.3) has a direct impact on the runtime distribution of vRAN tasks. The change of the distribution means that any model used for the prediction of the WCETs must be retrained frequently (every few ms) using online samples, to adapt to the various (and possibly unknown) collocated workloads. To show this in practice, we repeat the previous experiment over 4 CPU cores when the vRAN is running in isolation, as well as with collocated workloads (Redis, SQL server). By running the KS test [69] on our collected runtimes for the three cases (isolated, Redis, SQL server), we obtain p-values $<< 0.001$. This verifies that the runtimes in the case of interfering workloads are not drawn from the same distribution as in the isolated case, meaning that any model trained against the isolated RAN samples will need to be retrained online for improved accuracy.

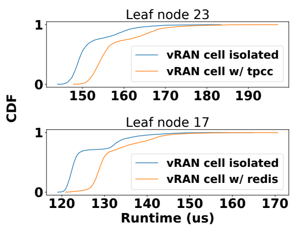
4.2 Concordia WCET prediction model

We now present the detailed design of the WCET prediction model of Concordia that, (i) makes parameterized WCET predictions considering the effect of the tasks' inputs to their runtime, and (ii) takes into account the challenges of Section 4.1.

High-level description of prediction mechanism: At a high level, the Concordia predictor maintains a separate quantile decision tree [70, 93] for each vRAN task, with training runtime samples stored in its leaf nodes. It provides a WCET prediction for the task with a given set of input parameters (or features) X using the maximum of the runtimes stored in the corresponding leaf node. The predictor builds the decision trees in an *offline phase*, using runtime samples measured for test vRAN workloads running without other collocated workloads. Then in an *online phase*, during regular operation, the predictor updates the runtime samples in each leaf *without changing the tree structure*. The intuition is that the tree splits the input feature space for the training set so that each leaf node ends up having a set of similar runtime samples. This is because it uses the CART algorithm [57] to minimize the variance among the samples that end up in the same leaf. We can then build and maintain separate simple predictors for each leaf online without having to retrain the trees, which is both computationally more expensive, and we also observe that is not needed.



(a) Violin plots of runtime samples used for each leaf node of the quantile decision tree.



(b) CDFs of runtimes of most dissimilar leaf nodes between isolated case and TPCC/redis

Figure 7: Mapping of runtime samples to decision tree leaves and effect of interference to their distribution (LDPC decoding).

To verify this intuition we first plot violin plots mapping the collected FlexRAN runtime samples of 120K runs to leaf nodes of a quantile decision tree used for the LDPC decoding task in the offline case (in isolation). For each run, we vary the size of the inputs (number of UEs, packet sizes etc). As shown in the top part of Fig. 7a, the variance of each violin plot is small compared to the overall variance of the input samples. We next plot violin plots (bottom part of Fig. 7a) representing the mapping of collected FlexRAN runtimes for the same workload, but in the presence of a collocated TPCC workload [80], and while still using the offline-trained decision tree. The distributions are visually similar to the isolated case, showing that the grouping of the online runtime samples remains similar when using the offline trained tree. We further verify this observation by zooming in and comparing the most distorted leaf node CDFs in the presence of collocated workloads compared to the isolated case (identified using the Wasserstein distance [105]). As shown in Fig. 7b for TPCC and redis, the runtime samples in the presence of interference result in heavier-tailed distributions, but the runtimes within the leaf node are still located in the same region. We verify the same for all other tasks and workloads we tried (e.g. redis, nginx, MLPerf).

Offline construction of quantile decision trees: The decision trees are trained offline, using a dataset with samples collected by profiling the vRAN in the absence of collocated workloads.Samples are collected at a TTI granularity and each sample contains the state of the vRAN and the runtime of the vRAN tasks. The state of the vRAN contains a set of features X, including data like the number of active UEs, their transport block sizes, the transmission configurations etc. To create a dataset with maximum coverage of the input space, the profiling is performed using a set of transmission parameters that vary for each TTI (e.g. 0 to 16 transmitting UEs, varying transport block sizes, modulation and coding schemes etc). Using this dataset and for each task t, we perform feature selection to identify a subset of features $X_t \subseteq X$ with the most significant impact to the task runtime. For the feature selection, we combine hand-picked features based on domain expertise and automated feature selection methods (correlation with the task runtime using the distance correlation metric [98, 99], backwards elimination). All this is summarized in Algorithm 1.

Online training and prediction: We construct the online prediction by simply replacing the offline samples in each leaf with online ones. The online prediction runs every TTI (every 0.5-1ms depending on

Algorithm 1: Construction of quantile decision tree for a signal processing task t

```
Input  : vRAN state $X$ of current TTI, set of handpicked features $X_t^h$ for
         task $t$, runtime $R_t$ of task in current TTI
Output : Feature vector $X_t$ of task $t$, quantile decision tree $T_t$
Tree Training ($X$, $R_t$, $X_t^h$)
    /* Pick N most highly correlated features using distance
       correlation metric [98, 99]                                  */
    $X_d \leftarrow dcor(X, R_t, N)$;
    /* Pick M features using backwards elimination feature
       selection                                                    */
    $X_d \leftarrow back\_elim(X_d, M)$;
    /* Combine with hand-picked features                            */
    $X_t \leftarrow X_t^h \cup X_d$ ;
    /* Train quantile decision tree                                 */
    $T_t \leftarrow train(X_t)$;
```

Algorithm 2: Quantile decision tree prediction model of a signal processing task t

```
Input  : Quantile decision tree $T_t$ with ringbuffer $B_i$ for leaf node $i$,
         features $X_t$ and runtime $R_t$ for task in current TTI
Output : WCET prediction $WCET_p$
Training Step ($T_t$, $X_t$, $R$)
    /* Traverse $T_t$ to find the appropriate leaf node            */
    $i \leftarrow T_t(X_t)$;
    Store $R$ in $B_i$;
Prediction Step ($B$, $X_t$)
    /* Traverse $T_t$ to find the appropriate leaf node            */
    $i \leftarrow T_t(X_t)$;
    $WCET_p \leftarrow max(B_i)$
```

the cell configuration) and has to be fast. For each leaf node i in the decision tree, we maintain a ring buffer B_i of the most recently observed execution times, which is updated at runtime in every TTI. Consider a task t assigned during a given TTI slot with parameters X_t, and whose observed runtime is R_t. We first traverse the decision tree T_t for that task and find the leaf node i that maps the task parameters X_t. We add the observation R_t to the buffer B_i. To predict the runtime of a task with parameters X_t in a given execution slot, we first find the decision tree node i that corresponds to the parameters X_t. We then use the maximum of all the samples found in the ring buffer B_i as an estimated WCET for the task. All this is formally shown in Algorithm 2.

Comparison with other approaches: Due to its parameterized prediction, Concordia offers more accurate WCET prediction than state-of-the-art real-time systems predictors [18, 111] that do not consider input parameters (see Section 6.3 for comparison). We also experimented with such methods (e.g. [23]) to replace our online predictor on each leaf node, but they provided similar accuracy while being more computationally expensive. We further tried different parameterized prediction models (linear and non-linear regression) instead of the decision tree, but they either provided lower prediction accuracy or reclaimed less CPU cores (results presented in Section 6).

5 IMPLEMENTATION

Here we describe the implementation of Concordia. We build our prototype on standard Linux, on top of Intel's FlexRAN v20.02 [48, 62], the most mature 4G and 5G vRAN implementation on the Intel architecture. We note that the OpenAirInterface [43, 52] and Agora [28] projects could also be used.

Following the FlexRAN recommendations [49], we use a number of CPU cores in a vRAN pool to execute signal processing tasks (the exact number of cores depends on the workload and is specified in Section 6). As already mentioned, the Concordia design assumes that the vRAN is the high priority workload and other workloads are best-effort. As such, we set the threads of the vRAN pool to a real-time scheduling policy (SCHED_FIFO, priority 94), meaning that they can only be preempted by a few critical kernel threads (e.g. watchdog thread). The only way that other workloads can run on the same cores is if the Concordia scheduler decides that the vRAN pool threads must yield. Moreover, we set the kernel parameter *sched_rt_runtime_us* to -1 to prevent non real-time tasks from running on the same cores as the vRAN pool while the vRAN worker threads have not yielded. We use the Linux *isolcpus* boot kernel parameter to dedicate 1 CPU core for the thread that maintains the time of the vRAN and periodically runs the Concordia scheduler and 3 CPU cores for the tasks of the MAC layer. Finally, we use a single core for OS management tasks. We offload the RCU callbacks of the system to the OS management core and migrate all the interrupts and kernel threads out of the used by the MAC layer and the Concordia scheduler (also banning those cores from irqbalance). However, unlike the isolated FlexRAN case, we allow interrupts and kernel threads to be served by the vRAN pool cores plus the OS management CPU core.

WCET Predictor implementation: The WCET predictor component of Concordia is auto-generated using a collection of Python scripts. During the offline tree construction phase we obtain 500K training samples from synthetic workloads in the way described in Section 4.2 and we automatically extract all relevant system parameters and the task runtimes for each TTI slot. We then run Algorithm 1 in Python, built on the pandas framework [76] with R bindings[1] for the use of the distance correlation algorithm [98, 99] and the scikit-learn library for the backwards elimination feature selection and the training of the decision trees [79]. Next, another Python script takes the decision tree from the previous phase and generates an optimized C code (about 6K lines of code) for traversing the tree and storing/fetching runtime samples (Algorithm 2), with the ring buffers of the leaf nodes having 5K entries. The predictor runs as a task on a CPU of the RAN pool in the beginning of each TTI. The predictor runtime is evaluated in Section 6.5.

Scheduler implementation: The Concordia scheduler is implemented in C (about 2K lines of code) and is integrated in the vRAN pool framework of FlexRAN as part of the timer thread, running on a dedicated core that is never preempted. The scheduler uses a bitmap with the ids of CPU cores in order to signal BBU worker threads that they must yield their cores to the OS. Semaphores are used to wake up the worker threads when scheduled. The scheduler changes the order of cores that are used for vRAN pools every 2ms to avoid constantly using the same cores. This allows other workloads that cannot be migrated to get some CPU time (e.g. some kernel threads and interrupts). The scheduler runtime is evaluated in Section 6.5.

[1]https://rdrr.io/cran/Rfast/man/dcor.html

Bandwidth	# cells	Avg DL cell throughput	Avg UL cell throughput	TTI processing deadline
100MHz	2	750Mbps	80Mbps	1.5ms
20MHz	7	270Mbps	120Mbps	2ms

Table 1: Cell configuration for evaluation of Concordia

6 EXPERIMENTAL EVALUATION

Here we evaluate the performance of Concordia. We start by describing the setup for the evaluations. We focus our evaluation on 5G vRAN cell deployments, i.e., we use the 5G signal processing chains of FlexRAN. As real-world 5G deployments are still at an early stage, we do not have access to realistic traffic patterns from real 5G cells and therefore we rely on emulated traces. The traces are based on actual 5G radio samples, encoding a varying number of 5G users, modulation and coding schemes, transport block sizes, data rates, MIMO antenna layers etc. We implement a traffic generator that combines these samples to create uplink traffic benchmark traces that are unique to each cell. The traces are based on the traffic fluctuation patterns of the LTE traces presented in Section 2.2, but with a volume of traffic that is scaled up to match that expected from 5G deployments (i.e., $> \times 10$ increase in the aggregate traffic of each cell). We create downlink traffic benchmarks in a similar way. While we acknowledge that the traffic patterns from real 5G cells might not fully match those of the traces used in this work, we believe that our evaluation can still provide deep insights into the effectiveness of Concordia, due to the randomness in the fluctuation of traffic for each cell and the uniqueness of each cell's trace.

For the experiments presented throughout this section and unless stated otherwise, we deploy our vRAN on the first NUMA node of a 48-core server (Intel Xeon Platinum 8168 @ 2.7GHz) running Ubuntu Linux 18.04, with hyper-threading disabled and configured for high performance as advised for FlexRAN [49]. This includes the use of a low latency kernel, disabled power states/frequency scaling and use of huge pages. Some experiments require servers with fewer cores. In order to compare on the same CPU architecture, we deactivate unneeded cores through the `sysfs` Linux virtual filesystem.

We consider the 2 cell configurations of Table 1 and 5 types of workloads collocated on the vRAN pool cores, that stress various parts of the server (CPU, memory, network, disk):

Redis We deploy 8 containers, each with a single Redis server. We saturate the servers using 8 remote instances of the Redis benchmark tool [87], connected over a 40G link and performing GET/SET requests over a set of 100K keys.

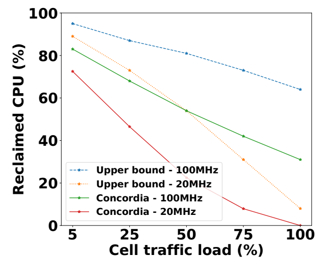
Nginx We deploy 5 Nginx containers and an external client, connected over a 40G link that fully saturates Nginx, fetching 612B-large HTTP files.

TPCC We deploy 1 container of a MySQL server and run a TPCC benchmark [80] using a remote client (1000 warehouses and 32 simultaneous connections).

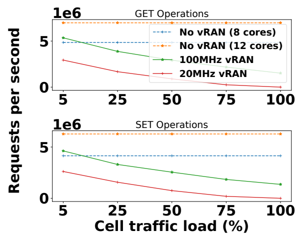
MLPerf We deploy 1 container running MLPerf [86] to train ResNet50-v1.5 [45, 46] for image classification using the ImageNet 1K dataset[2].

Mix We deploy a mix of the above workloads at the same time. The workloads are turned on and off at random time intervals ranging from 10 to 70 seconds.

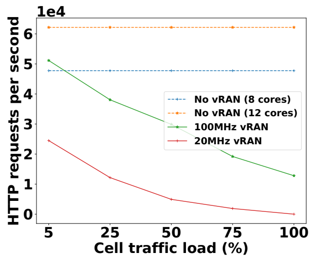
[2]http://www.image-net.org/



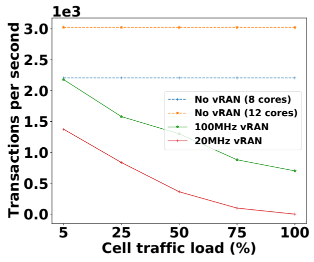
(a) CPU cores reclaimed by Concordia vs ideal case



(b) Redis benchmark performance (8 Docker containers)



(c) Nginx benchmark performance (5 Docker containers)



(d) TPCC benchmark performance (1 Docker container)

Figure 8: Reclaimed vRAN pool CPU cores for vRAN with 20MHz (7 cells) and 100MHz (2 cells) configurations and performance of collocated workloads for various cell traffic loads.

Unless stated otherwise, we allocate 8 CPU cores for the vRAN pool and run 15 minutes experiments, following the Intel recommendation for quick performance validation [49]. This corresponds to a number of scheduling events ranging between 3.6×10^6 and 6.3×10^6 depending on the cell configuration. We further validate the 99.999% reliability of Concordia by running 8 hours tests with the mixed workload (between 1.152×10^8 and 2.016×10^8 scheduling events). No performance or reliability differences were observed between the long and the short tests. We compare Concordia with the baseline FlexRAN scheduler, as it is the most intuitive queue-based design. It acquires more cores when there are tasks waiting in the queues and relinquishes them when the queues are empty. In Section 6.3 we compare against other scheduler designs.

6.1 High-level benefits of Concordia

One of the main goals of Concordia is to improve CPU utilization on vRAN servers running BBU tasks. To this end, we start by evaluating the benefits of Concordia for the vRAN collocated workloads. We vary the traffic load up to the peak traffic listed in Table 2 for different cell configurations, and for each load we generate random traffic as described above. To make the comparison fair, we use the minimum number of cores required to meet the vRAN processing deadline.

We begin by measuring the percentage of CPU cores that are made available by Concordia to other workloads and we compare this to the ideal case where every idle CPU cycle is reclaimed. As it can be seen in Fig. 8a, Concordia can reclaim more than 70% of the CPU cores for low cell traffic loads both for 20MHz and 100MHz cell configurations. The percentage drops to 0% and 38% correspondingly for cells operating at the max allowed average load. We observe that Concordia is slightly more efficient for low cell workloads, because there are many idle TTI slots whose duration is easy to predict.

We next study the performance of different workloads collocated with the vRAN. As a reference, we measure the maximum achievable performance of those workloads in the ideal case, when running on the same cores without the vRAN workload. As shown in Fig. 8b-8d, the achieved performance varies depending on the workload. For example, in the case of the 100MHz cell configuration and for low cell traffic load (83.3% of the cores reclaimed), TPCC achieves 72% of the ideal performance (without the vRAN), Redis achieves 76.6% and Nginx achieves 82.2%. The MLPerf workload figure is omitted due to lack of space, but similar results were obtained (78% of the ideal performance achieved for low cell traffic load in the 100MHz case). The reason for the lower yield compared to the theoretical max expected performance is related to the effects that the collocated workloads have on cache pollution, preemption, scheduling latencies, etc (as also observed in [56]). We next study these effects in detail. It should be noted that throughout these experiments, Concordia provided 99.999% reliability to the TTI processing latency of the vRAN pool.

Overall, we conclude that Concordia is able to recover a large fraction of CPU cycles unused by vRAN. This is in contrast with the current operators' best practice which does not attempt any load sharing on servers with vRAN pools.

Bandwidth	# cells	Peak DL cell throughput	Peak UL cell throughput	# of CPU cores
100MHz	2	1.5Gbps	160Mbps	12
20MHz	7	380Mbps	160Mbps	8

Table 2: Cell configuration and minimum number of CPU cores required for evaluation of Section 6.1 .

6.2 Effects of collocation on the vRAN

One of the key benefits of Concordia is its ability to predict task execution times. Because of this, it can minimize the number of cores it uses at any time. This increases cache locality, reduces cache pollution and reduces OS scheduler calls, making the system more efficient while leaving unneeded cores to the collocated workloads. The vanilla FlexRAN scheduler does not have an estimate of the traffic and has to be more conservative in allocating more cores than necessary. It also has to acquire and release the cores back to the OS more frequently in order to be able to share. This reduces the locality and increases cache trashing on the cores used by the vRAN pools. We next quantify these effects with experiments.

Cache efficiency: To measure the cache efficiency, we use the Linux `perf` tool [25] to profile the vRAN pool worker threads. We measure the change in the cache efficiency observed by the worker threads with a collocated workload compared to the baseline isolated vRAN case. Here, we present results for 100MHz cell configuration and the Redis workload (the other results are similar and we omit them due to lack of space). As it can be seen in Fig. 9, vanilla FlexRAN has a 25% increase in the stall cycles per instruction due to L1 cache misses compared to the baseline isolated vRAN case. This leads to an increase in the runtime of the signal processing tasks of the vRAN pool and thus directly affects the tail TTI processing latency.

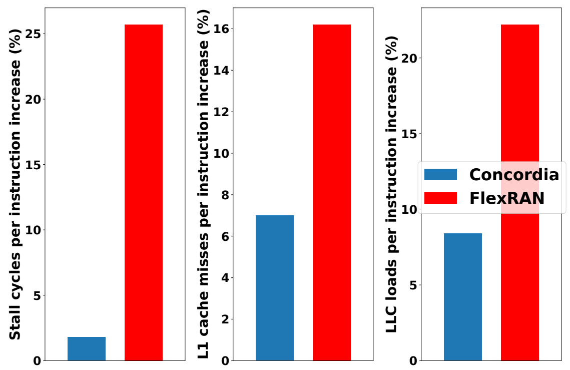


Figure 9: Latency effects of cache from collocated workload (Redis) interference for 2 100MHz cells

In contrast, Concordia is able to predict well the number of required cores and thus limits the increase in the stall cycles caused by cache misses due to collocation to less than 2%.

OS scheduling latency: We use the `runqlat` tool that is part of the BCC toolkit [63] to measure the OS scheduling latency of the vRAN pool worker threads once they have yielded and have been signaled to wake up (from the Concordia scheduler or from the generation of more signal processing tasks in the case of vanilla FlexRAN). We collect scheduling latency measurements for 1 minute. As shown in Fig. 10 (log scale), the total number of scheduling events of vanilla FlexRAN is significantly higher compared to that of Concordia (about 230% higher in both the isolated and interfering case), leading to higher scheduling latency per slot and thus more deadline violations. The reduced number of OS scheduling calls in the case of Concordia is due to the proactive allocation of cores, which does not allow worker threads to yield while more signal processing tasks are expected during a TTI slot. A side-effect of this is that Concordia has a higher number of scheduling events with high tail latency (>63 μs) in the presence of other workloads compared to FlexRAN. We believe this is because the CPU cores of the vRAN pool are retained by the worker threads longer, leading to the queuing of OS tasks that cannot be migrated (e.g. interrupts) and increasing the chances of some kernel thread entering a non-preemptible section when the worker threads yield. This effect is mitigated by the fine-grained scheduling of cores by Concordia every 20 μs, since more cores can be allocated to the vRAN if a scheduled core fails to wake up in time.

Tail latency: We next compare the effects of collocated workload interference to the tail processing latency of Concordia vs vanilla FlexRAN. For each cell configuration considered in Table 1 and different workloads we run experiments measuring the 99.99% and 99.999% TTI processing latency of the vRAN pool. The results are illustrated in Fig. 11. Without other workloads, both FlexRAN and Concordia can meet the processing deadline with 99.999% reliability for both configurations. Once we introduce any other workload, the tail latency of vanilla FlexRAN increases significantly and it is no longer possible to provide 99.999% of reliability or even 99.99%, with the exception of MLPerf. However, Concordia is not affected and maintains 99.999% of reliability in all cases. The same observations apply for the mixed workload test (figure omitted due to lack of space).

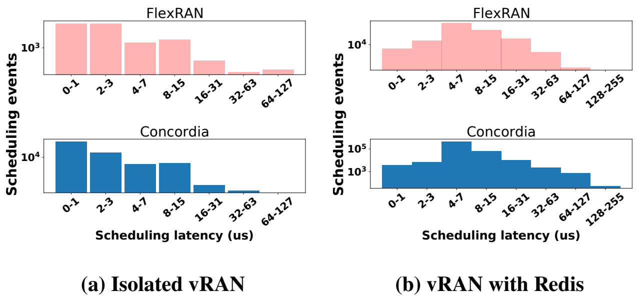


Figure 10: Scheduling latency of vRAN pool worker threads (8 CPU cores) for 2 100MHz cells with and without workload interference (Redis). Y-axis in log scale.

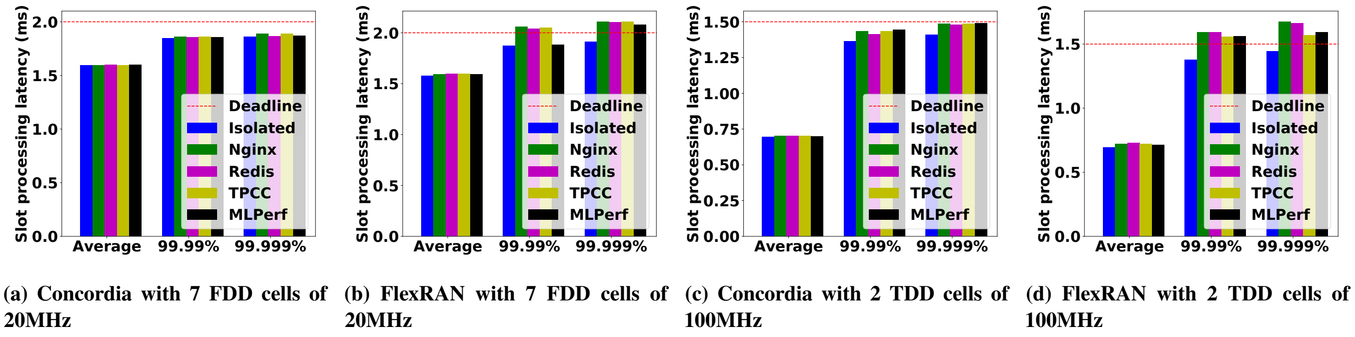
Number of vRAN pool cores: Adding more CPU cores to the vRAN pool helps Concordia to meet deadlines. We see that in Fig 12 for a test using a constantly running mixed workload. The 20MHz cell configuration achieves 99.999% of reliability with 8 cores (Fig. 12a), but the 100MHz case achieves only 99.99% (Fig. 12b). However, by adding one more CPU core to the vRAN pool, the reliability goes back to 99.999%. This is because the more CPU cores we assign to the vRAN pool, the more chances Concordia will have to schedule an extra core if the vRAN is on track of missing a deadline and an already scheduled core takes a long time to wake up (e.g. due to a non-preemptive kernel task occupying it), as described in Section 3.

6.3 Comparison with alternative schedulers

Conventional WCET prediction method: We compare the effectiveness of the Concordia predictor against a well-known method [23] that is representative of the probabilistic WCET prediction literature [18]. The method in [23] uses Extreme Value Theory and predicts a single WCET per signal processing task regardless of its input with a confidence of 0.99999. As shown in Fig 13 for the 20MHz cell configuration, Concordia outperforms the conventional model (up to 20% difference in reclaimed cycles). This is because the conventional WCET model makes more pessimistic predictions compared to Concordia. At the same time, the tail latency reduction achieved by the conventional model is marginal (about 5 μs in all cases), further incentivizing our use of a parametrized WCET prediction model. We make similar observations for the 100MHz case, with the results omitted due to lack of space.

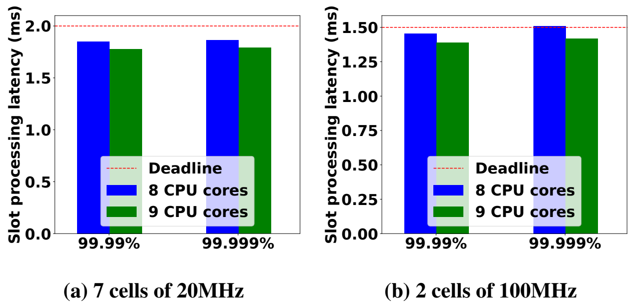
Schedulers not considering the WCET: Next, we compare the reliability of Concordia to two schedulers that do not take into account the WCET of tasks: (i) a variant of Shenango [75] (also used in Snap [68]) and (ii) a utilization-based scheduler. Our Shenango-variant increases the number of cores allocated to the vRAN by one every time that a signal processing task remains in the priority queue for more than a predefined amount of time, and we vary this threshold. The utilization-based scheduler adjusts the number of cores based on the utilization of the vRAN in the past few TTIs. Once the utilization surpasses a threshold (60% and 30% for the 20MHz and 100MHz cell configurations), an additional worker thread is woken up.

In the case of the Shenango-based scheduler, it was very challenging to identify the queuing time threshold that would both satisfy the vRAN deadlines and would allow other workloads to share the



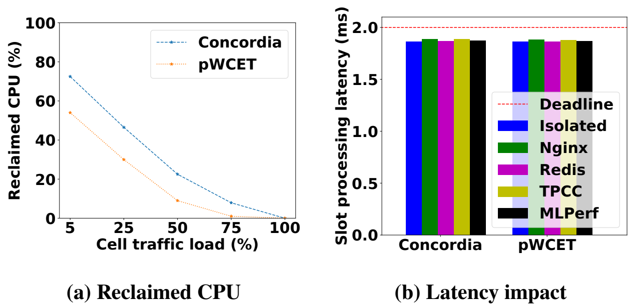
(a) Concordia with 7 FDD cells of 20MHz (b) FlexRAN with 7 FDD cells of 20MHz (c) Concordia with 2 TDD cells of 100MHz (d) FlexRAN with 2 TDD cells of 100MHz

Figure 11: Tail TTI processing latency (99.99% and 99.999%) of Concordia vs vanilla FlexRAN in the presence of various workloads. All experiments are performed on with a vRAN pool of 8 CPU cores.



(a) 7 cells of 20MHz (b) 2 cells of 100MHz

Figure 12: Concordia tail TTI processing latency for mixture of Nginx, Redis, TPCC workloads

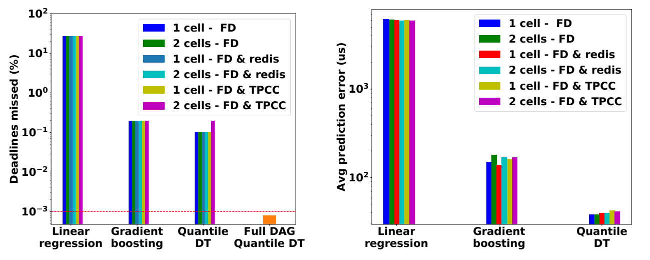


(a) Reclaimed CPU (b) Latency impact

Figure 13: WCET prediction accuracy effect of various prediction methods for LDPC decoding task.

vRAN pool cores. Setting to a high value (200 μs) made the scheduler react slowly to delays in the processing of vRAN tasks, with less than 99.99% deadlines met. Setting to a low value (5 μs), similar to Shenango, led to the vRAN utilizing all of the CPU resources, never allowing other workloads to run. Different values ranging from 5 μs to 200 μs provided better results for different vRAN traffic loads, but no single value always met deadlines with ≥ 99.99% reliability.

In the case of the utilization-based scheduler, the vRAN traffic burstiness could not be captured by observing the past utilization of the vRAN pool cores. This led to less than 99.99% reliability in the presence of collocated workloads, since the scheduler often underestimated the amount of CPU resources required for processing the upcoming TTI slot. These results reinforce our finding that having



(a) Percentage of slots where the processing deadline was violated. Y-axis in log scale. (b) Average WCET prediction error for successfully met deadline. Y-axis in log scale.

Figure 14: WCET prediction accuracy effect of various prediction methods for LDPC decoding task.

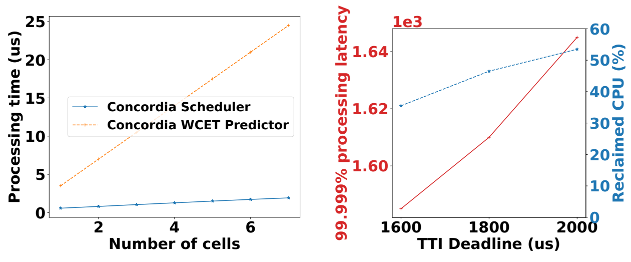
predictions of task execution times is instrumental for efficient CPU sharing in the vRAN.

6.4 Accuracy of other prediction models

Here, we compare the accuracy of Concordia's quantile decision tree against other prediction models we explored. We consider a linear regression and a (non-linear) gradient boosting model. For the training of the models, we collected the vRAN state and runtimes offline in the same way as described in Section 4.2 for the quantile decision tree and selected training features according to Algorithm 1. We also adapted the models to take into account the online runtime samples, like in the quantile decision tree case (we omit the details due to lack of space).

We perform probabilistic WCET predictions using a prediction interval of 0.99999. To evaluate the prediction accuracy we use two metrics; (i) the percentage of missed deadlines (i.e. times that the runtime of the task exceeded the predicted WCET) and (ii) the average WCET prediction error for successfully met deadlines. The intuition behind the second metric is that the closer a successful WCET prediction is to the actual runtime, the less cores would be dedicated to the vRAN by Concordia, freeing up cycles for other workloads.

We generate randomly fluctuating traffic for the 20MHz cell configuration of Table 1, varying the number of UEs (0 to 8). We consider deployment scenarios with 1 or 2 cells and different types of collocated workloads (none, Redis, TPCC benchmark) on 4 CPU cores. For each scenario we ran a 5 minutes test. As it can be seen in Fig. 14a for the LDPC decoding task, the non-linear gradient boosting model



(a) Processing overhead of Concordia WCET predictor and scheduler for a varying number of cells

(b) Effect of TTI deadline parameter on tail latency and reclaimed cores (20MHz cell configuration)

Figure 15: Characteristics of Concordia scheduler

is almost equally effective to the quantile decision tree in predicting deadlines, and much better than the linear model. However, as shown in Fig. 14a, the quantile decision tree has the smallest average WCET prediction error when deadlines are met (43us), making it the most efficient of all studied algorithms. Given the lack of space, please see Appendix A.2 for additional prediction accuracy results of other computationally intensive signal processing tasks. It should be noted that while the prediction accuracy for individual tasks is not 0.99999, the Concordia scheduler compensates for any misprediction by updating its scheduling decision every 20 μs. This results in 99.999 reliability for the full DAG execution, as shown if Fig. 14a and in Section 6.2.

6.5 Concordia scheduler characteristics

Execution times: The Concordia scheduler runs once every 20 μs and the WCET predictor once every TTI slot, so they have to be very fast. We evaluate their execution time while varying the number of cells from 1 to 7. As shown in Fig. 15a, their overhead increases linearly with the number of cells, since the number of tasks being processed scales the same way. The scheduler runs on the timer thread, but as its overhead for up to 7 cells remains always below 2 μs, it can run every 20 μs without any issues (it is also possible to multiplex the scheduler with other tasks on the timer core, such as processing the incoming fronthaul packets). The total overhead of the WCET predictions grows from 4 μs for 1 cell to 24 μs for 7 cells. This is less than 0.2% of the overall vRAN pool processing time per TTI slot (and as it runs on the vRAN pool cores as discussed in Section 5, it doesn't block the timer thread).

Effect of TTI deadline: Next, we study the effect of the signal processing DAG deadline to the performance of the vRAN and the number of reclaimed cores. We consider as an example the 20MHz 7 cell configuration of Table 2, with a cell traffic load of 25% of the max designated capacity. As it can be seen in Fig. 15b, the shorter the deadline is, the lower the tail TTI processing latency gets at the expense of a lower number of reclaimed CPU cores. Similar observations can be made for other cell configurations (omitted due to lack of space). The DAG deadline can therefore be used to tune the performance of the vRAN with different values providing a tradeoff between the vRAN reliability (e.g. 99.99% or 99.999%) and the percentage of reclaimed vRAN pool CPU cycles.

7 CONCORDIA EXTENSIONS

Offloading vRAN tasks to hardware accelerators: The focus of this paper has been on vRAN deployments that fully rely on CPUs for signal processing. However, in many practical scenarios hardware accelerators (e.g. FPGAs, GPUs) could be used to offload heavy tasks like LDPC encoding/decoding [48] to reduce processing latency and to improve energy efficiency.

# cells	Minimum # CPU cores	Average CPU utilization
1	1	58.2%
2	3	46.6%
3	4	58.7%

Table 3: vRAN pool CPU requirements for 100MHz TDD cell configuration (1.6Gbps DL, 150Mbps UL per cell) and FPGA LDPC acceleration

To understand the impact of accelerators to the benefits of Concordia, we extended our FlexRAN testbed with a server (Intel Xeon W-2295 @ 3GHz, Ubuntu 18.04 lowlatency kernel) equipped with an FPGA (Terasic DE5-Net) for offloading LDPC encoding/decoding tasks. We profile the vRAN performance for peak traffic and measure the minimum number of vRAN pool cores required to support the vRAN, as well as their utilization for a varying number of 100MHz TDD cells. As shown in Table 3, the FPGA use enables support for more cells with higher traffic loads on the same number of CPU cores compared to the scenarios studied in Section 6. However, the CPU utilization still remains below 60% in all cases. This underutilization of the cores even at peak capacity happens for two main reasons:

	Average processing time of non-offloaded tasks (μs)	Average total processing time of single slot (μs)
Uplink	515	1414
Downlink	196	366

Table 4: Average processing times for uplink/downlink slot of single cell (including FPGA acceleration) and for non-offloaded tasks (excluding FPGA acceleration) on 1 CPU core.

- **Time division multiplexing** As in the case of non-accelerated configurations, the time division multiplexing of cells creates idle periods for the vRAN pool cores, since the downlink processing time is significantly lower compared to the uplink for the non-offloaded tasks. This can be seen in Table 4 for the single cell case of the scenario under study, where the average total pool CPU core time spent on the non-offloaded uplink processing tasks is more than 2.5 times higher than the downlink, even though the downlink traffic volume is an order of magnitude higher.
- **Offload processing wait times** Due to the dependencies of the tasks in the DAG structures of the signal processing chains, the worker threads running on the vRAN pool cores have periods when they cannot make any progress and therefore have to block, waiting for the completion of FPGA offloaded tasks. As shown in Table 4, the average total processing time of a single uplink slot (including the FPGA processing time) is ~2.5 times higher than the average processing time for the non-offloaded tasks executed on the allocated vRAN pool core. Their difference matches the time that the vRAN pool worker thread had to block, waiting for the offloaded tasks to be completed. Similar observations can be made for the downlink, where the total slot processing time is ~1.9 times higher than the processing time of the non-offloaded tasks.

The aforementioned observations demonstrate that there are significant opportunities for reclaiming vRAN CPU cores even in the presence of hardware accelerators. We plan to extend Concordia to accommodate such scenarios. This can be achieved by extending Concordia's WCET predictor to also predict the WCET of the offloaded tasks, as well as by adapting Concordia's scheduler to factor in the idle periods arising from the offloading of the tasks, e.g., by creating separate DAGs for the tasks running before/after the offloaded tasks and adjusting the deadlines of those DAGs appropriately.

Extending Concordia for other workloads: Throughout this work, we assumed that the vRAN is the high priority workload, with a maximum scheduling priority. All other workloads are considered as best-effort and as such can be pre-empted by the vRAN at any point in time. Based on this, the focus of this work has been on providing predictions specifically targeting the physical layer signal processing vRAN tasks. However, the techniques used by Concordia could be generalized to also apply to other task-based deadline-constrained workloads across the protocol stack of the vRAN, as well as to applications running as part of a (near) real-time RAN intelligent controller for the optimization of the RAN radio resources [2, 36, 38, 73].

One characteristic example is the MAC layer of the vRAN, which is responsible for the scheduling of radio resources to mobile devices. The schedulers of the MAC layer (e.g. uplink, downlink, broadcast etc.) can be viewed as deadline tasks that can be processed by a vRAN pool, similar to the signal processing tasks of the physical layer. In fact, this is the approach proposed by Intel as a best practice for the L2 of FlexRAN [62]. Moving towards 5G networks and beyond, the processing requirements of the MAC layer increase. For example, the introduction of Massive MIMO increases the scheduling complexity, which can greatly fluctuate depending on the number of scheduled users and their mapping to antennas [14]. The WCET prediction capabilities of Concordia could allow the vRAN MAC to be multiplexed with other workloads.

The proposed schemes of Concordia could be extended and applied to other domains with latency sensitive characteristics, like AR/VR workloads [55, 64] and video analytics [3], where the processing time of frames needs to be minimized to provide the optimal experience and/or to actuate some other system (e.g. traffic lights).

8 RELATED WORK

vRAN resource management: A number of works have focused on the problem of vRAN resource pooling to optimize the allocation of compute resources across BBUs [15, 40, 115]. Going one step beyond, vrAIn [5] proposes a joint compute and radio resource allocation framework for the vRAN based on reinforcement learning. In contrast to Concordia, the aforementioned works assume an isolated vRAN and do not consider the effects of scheduling latency and cache interference to the WCET of signal processing tasks. The problem of controlled tail latency for signal processing tasks has been studied both in the context of general purpose processors (e.g. [101]) and DSPs (e.g. [7]). However, such works do not consider the presence of collocated workloads as in the case of Concordia. Finally, a number of ML-based techniques have been proposed for the intelligent allocation of resources to the RAN (e.g., [10, 37, 97]). However, the focus of such works has been on the radio resources and not on the optimization of compute.

Real-time scheduling: Real-time scheduling has been studied extensively in the literature [24]. Relevant to Concordia, a plethora of works focus on mixed-criticality systems[17] and on the scheduling of parallel task DAG models similar to that of the vRAN (e.g., [8, 9, 50, 59–61, 82, 91]). Concordia builds on the work in [61], which proposes the most relevant state-of-the art mixed-criticality deadline scheduler.

An integral requirement of real-time schedulers is the knowledge of task WCETs. As such, there exists a large volume of work on WCET prediction for hard real-time systems [111]. More recent approaches have focused on providing probabilistic WCET bounds (e.g. with 4 or 5 nines) through distributions obtained using static analysis, measurements or a combination of both [18]. However, in all such works, the WCET prediction does not adjust dynamically at runtime based on the input, leading to underutilization of the compute resources. Moreover, most such works target embedded systems and therefore assume that real-time tasks operate without the presence of other interfering non real-time workloads. Concordia overcomes this limitation through the introduction of its novel parameterized WCET predictor and its offline and constant online training phases.

Low-latency scheduling frameworks: Workload interference is a well-known problem. As such, various resource allocation and scheduling optimization frameworks have been proposed (e.g. [13, 65, 81, 83]). While the goal of such frameworks is to mitigate the effects of interference, they operate at a coarse time granularity, which is not suitable to deal with the sub-millisecond requirements of the vRAN. Shinjuku [51] enables scheduling for microsecond-scale tail latencies. However, its design as a single-address space OS does not allow the deployment of conventional applications. Shenango [75] and Snap [68] bear the most similarities to Concordia. However, neither provides mechanisms to predict the (varying) WCETs of tasks, nor is able to provide 99.999 reliability. Moreover, Shenango requires from applications to implement a specific API and to avoid the use of system calls. In contrast, Concordia allows the collocation of the vRAN with conventional applications (e.g. running in containers).

9 CONCLUSIONS

In this work we presented Concordia, a userspace deadline-aware scheduling framework for the sharing of compute resources between the vRAN and best-effort workloads. Concordia allocates CPU resources among the vRAN physical layer and other workloads at a granularity of 20 μs, ensuring that the vRAN meets its real-time signal processing deadlines. The scheduling decisions of Concordia are powered by a prediction mechanism based on quantile decision trees that predicts the WCET of signal processing tasks in the presence of interference from other workloads. Experimental results on a prototype based on the commercial Intel FlexRAN vRAN solution demonstrate the ability of Concordia to reclaim more than 70% of the vRAN's compute resources, while providing 99.999% reliability in meeting vRAN signal processing deadlines.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers and our shepherd Muhammad Shahbaz for their valuable feedback that helped us improve this work.

REFERENCES

[1] 3GPP. 2019. 5G NR Physical Channels and Modulation, document 38.211.

[2] ORAN Alliance. 2020. O-RAN Use Cases and Deployment Scenarios. *White Paper, Feb* (2020).

[3] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. 2017. Real-time video analytics: The killer app for edge computing. *computer* 50, 10 (2017), 58–67.

[4] Erdal Arikan. 2009. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on information Theory* 55, 7 (2009), 3051–3073.

[5] Jose A Ayala-Romero, Andres Garcia-Saavedra, Marco Gramaglia, Xavier Costa-Perez, Albert Banchs, and Juan J Alcaraz. 2019. vrAIn: A Deep Learning Approach Tailoring Computing and Radio Resources in Virtualized RANs. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[6] Arjun Balasingam, Manu Bansal, Rakesh Misra, Kanthi Nagaraj, Rahul Tandra, Sachin Katti, and Aaron Schulman. 2019. Detecting if LTE is the Bottleneck with BurstTracker. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–15.

[7] Manu Bansal, Aaron Schulman, and Sachin Katti. 2015. Atomix: A framework for deploying signal processing applications on wireless infrastructure. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. 173–188.

[8] Sanjoy Baruah. 2016. The federated scheduling of systems of mixed-criticality sporadic DAG tasks. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 227–236.

[9] Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. 2012. A generalized parallel task model for recurrent real-time processes. In *2012 IEEE 33rd Real-Time Systems Symposium*. IEEE, 63–72.

[10] Ali Kashif Bashir, Rajakumar Arul, Shakila Basheer, Gunasekaran Raja, Ramkumar Jayaraman, and Nawab Muhammad Faseeh Qureshi. 2019. An optimal multitier resource allocation of cloud RAN in 5G using machine learning. *Transactions on emerging telecommunications technologies* 30, 8 (2019), e3627.

[11] Ejder Bastug, Mehdi Bennis, and Mérouane Debbah. 2014. Living on the edge: The role of proactive caching in 5G wireless networks. *IEEE Communications Magazine* 52, 8 (2014), 82–89.

[12] Dario Bega, Albert Banchs, Marco Gramaglia, Xavier Costa-Pérez, and Peter Rost. 2018. CARES: Computation-aware scheduling in virtualized radio access networks. *IEEE Transactions on Wireless Communications* 17, 12 (2018), 7993–8006.

[13] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. 2014. {IX}: A Protected Dataplane Operating System for High Throughput and Low Latency. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 49–65.

[14] Mouncef Benmimoune, Elmahdi Driouch, Wessam Ajib, and Daniel Massicotte. 2015. Joint transmit antenna selection and user scheduling for massive MIMO systems. In *2015 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 381–386.

[15] Sourjya Bhaumik, Shoban Preeth Chandrabose, Manjunath Kashyap Jataprolu, Gautam Kumar, Anand Muralidhar, Paul Polakos, Vikram Srinivasan, and Thomas Woo. 2012. CloudIQ: A framework for processing base stations in a data center. In *Proceedings of the 18th annual international conference on Mobile computing and networking*. 125–136.

[16] Nicola Bui and Joerg Widmer. 2016. OWL: A reliable online watcher for LTE control channel measurements. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*. 25–30.

[17] Alan Burns and Robert I Davis. 2017. A survey of research into mixed criticality systems. *ACM Computing Surveys (CSUR)* 50, 6 (2017), 1–37.

[18] Francisco J Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. 2019. Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–35.

[19] SDX Central. 2020. Rakuten Mobile Delivers Its Virtualized Reality. Retrieved 2021-06-21 from https://www.sdxcentral.com/articles/news/rakuten-mobile-delivers-its-virtualized-reality/2020/04/

[20] Aleksandra Checko, Henrik L Christiansen, Ying Yan, Lara Scolari, Georgios Kardaras, Michael S Berger, and Lars Dittmann. 2014. Cloud RAN for mobile networksA technology overview. *IEEE Communications surveys & tutorials* 17, 1 (2014), 405–426.

[21] Intel Corporation. 2020. OpenNESS Radio Access Network configuration. https://github.com/open-ness/specs/blob/master/doc/ran/openness$_r$an.md Accessed: 2020-06-02.

[22] Max Costa. 1983. Writing on dirty paper (corresp.). *IEEE transactions on information theory* 29, 3 (1983), 439–441.

[23] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J Cazorla. 2012. Measurement-based probabilistic timing analysis for multi-path programs. In *2012 24th euromicro conference on real-time systems*. IEEE, 91–101.

[24] Robert I Davis and Alan Burns. 2011. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)* 43, 4 (2011), 1–44.

[25] Arnaldo Carvalho De Melo. 2010. The new Linux 'perf' tools. In *Slides from Linux Kongress*, Vol. 18. 1–42.

[26] Christina Delimitrou and Christos Kozyrakis. 2013. QoS-aware scheduling in heterogeneous datacenters with paragon. *ACM Transactions on Computer Systems (TOCS)* 31, 4 (2013), 1–34.

[27] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: resource-efficient and QoS-aware cluster management. *ACM SIGPLAN Notices* 49, 4 (2014), 127–144.

[28] Jian Ding, Rahman Doost-Mohammady, Anuj Kalia, and Lin Zhong. 2020. Agora: Real-time massive MIMO baseband processing in software. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 232–244.

[29] Ericsson. 2016. How cloud and networks achieve 99.999in different ways. Retrieved 2021-01-21 from https://www.ericsson.com/en/blog/2016/9/how-cloud-and-networks-achieve-99.999-availability-in-different-ways

[30] Ericsson. 2019. Critical capabilities for private 5G networks. Retrieved 2021-06-4 from https://www.ericsson.com/en/reports-and-papers/white-papers/private-5g-networks

[31] Ericsson. 2020. 5G private network operations: What do you need to know? Retrieved 2021-06-4 from https://www.ericsson.com/en/blog/2020/7/5g-private-network-operations

[32] TR ETSI. 2018. 138 913 V15. 0.0 (2018-09) 5G," Study on scenarios and requirements for next generation access technologies (3GPP TR 38.913 version 15.0. 0 Release 15)".

[33] Robert Falkenberg and Christian Wietfeld. 2019. FALCON: An Accurate Real-time Monitor for Client-based Mobile Network Data Analytics. In *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Waikoloa, Hawaii, USA. https://doi.org/10.1109/GLOBECOM38437.2019.9014096 arXiv:1907.10110

[34] Fierce Wireless. 2020. Dish names Intel as vRAN network supplier. https://www.fiercewireless.com/operators/dish-names-intel-as-vran-network-supplier Accessed: 2021-06-4.

[35] Alexander Fish, Shamgar Gurevich, Ronny Hadani, Akbar M Sayeed, and Oded Schwartz. 2013. Delay-Doppler channel estimation in almost linear complexity. *IEEE Transactions on Information Theory* 59, 11 (2013), 7632–7644.

[36] Xenofon Foukas, Mahesh K Marina, and Kimon Kontovasilis. 2017. Orion: RAN slicing for a flexible and cost-effective multi-service mobile network architecture. In *Proceedings of the 23rd annual international conference on mobile computing and networking*. 127–140.

[37] Xenofon Foukas, Mahesh K Marina, and Kimon Kontovasilis. 2019. Iris: Deep reinforcement learning driven shared spectrum access architecture for indoor neutral-host small cells. *IEEE Journal on Selected Areas in Communications* 37, 8 (2019), 1820–1837.

[38] Xenofon Foukas, Navid Nikaein, Mohamed M Kassem, Mahesh K Marina, and Kimon Kontovasilis. 2016. FlexRAN: A flexible and programmable platform for software-defined radio access networks. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. 427–441.

[39] Andres Garcia-Saavedra, Xavier Costa-Perez, Douglas J Leith, and George Iosifidis. 2018. Fluidran: Optimized vran/mec orchestration. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2366–2374.

[40] Krishna C Garikipati, Kassem Fawaz, and Kang G Shin. 2016. RT-OPEX: Flexible scheduling for cloud-ran processing. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*. 267–280.

[41] Sriram Govindan, Jie Liu, Aman Kansal, and Anand Sivasubramaniam. 2011. Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 1–14.

[42] GSMA. 2019. Vodafone starts trials of OpenRAN in Europe and Africa. Retrieved 2020-07-14 from https://www.gsma.com/futurenetworks/digest/vodafone-starts-trials-of-openran-in-europe-and-africa

[43] Wang Tsu Han and Raymond Knopp. 2018. OpenAirInterface: A pipeline structure for 5G. In *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*. IEEE, 1–4.

[44] Lajos Hanzo, Tong Hooi Liew, and Bee Leong Yeap. 2002. *Turbo coding, turbo equalisation, and space-time coding*. Wiley Online Library.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[46] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In *European conference on computer vision*. Springer, 630–645.

[47] China Mobile Research Institute. 2011. C-RAN the road towards green ran.

[48] Intel. 2018. An Overview of FlexRAN Software Wireless Access Solutions. Retrieved 2020-07-15 from https://software.intel.com/content/www/us/en/devel

op/videos/an-overview-of-flexran-sw-wireless-access-solutions.html

[49] Intel. 2020. FlexRAN. https://github.com/intel/FlexRAN Accessed: 2020-09-03.

[50] Xu Jiang, Nan Guan, Di Liu, and Weichen Liu. 2019. Analyzing GEDF Scheduling for Parallel Real-Time Tasks with Arbitrary Deadlines. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1537–1542.

[51] Kostis Kaffes, Timothy Chong, Jack Tigar Humphries, Adam Belay, David Mazières, and Christos Kozyrakis. 2019. Shinjuku: Preemptive scheduling for μsecond-scale tail latency. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 345–360.

[52] Florian Kaltenberger, Aloizio P Silva, Abhimanyu Gosain, Luhan Wang, and Tien-Thinh Nguyen. 2020. OpenAirInterface: Democratizing Innovation in the 5G Era. *Computer Networks* (2020), 107284.

[53] Swarun Kumar, Ezzeldin Hamed, Dina Katabi, and Li Erran Li. 2014. LTE radio analytics made easy and accessible. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 211–222.

[54] Altice labs. 2021. Towards autonomous private 5G networks. Altice labs. White Paper.

[55] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, Ningwei Dai, and Hung-Sheng Lee. 2019. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. *IEEE Transactions on Mobile Computing* 19, 7 (2019), 1586–1602.

[56] Jacob Leverich and Christos Kozyrakis. 2014. Reconciling high server utilization and sub-millisecond quality-of-service. In *Proceedings of the Ninth European Conference on Computer Systems*. 1–14.

[57] Roger J Lewis. 2000. An introduction to classification and regression tree (CART) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, Vol. 14.

[58] He Li, Kaoru Ota, and Mianxiong Dong. 2018. Learning IoT in edge: Deep learning for the Internet of Things with edge computing. *IEEE network* 32, 1 (2018), 96–101.

[59] Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. 2013. Outstanding paper award: Analysis of global edf for parallel tasks. In *2013 25th Euromicro Conference on Real-Time Systems*. IEEE, 3–13.

[60] Jing Li, Jian Jia Chen, Kunal Agrawal, Chenyang Lu, Chris Gill, and Abusayeed Saifullah. 2014. Analysis of federated and global scheduling for parallel real-time tasks. In *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 85–96.

[61] Jing Li, David Ferry, Shaurya Ahuja, Kunal Agrawal, Christopher Gill, and Chenyang Lu. 2017. Mixed-criticality federated scheduling for parallel real-time tasks. *Real-time systems* 53, 5 (2017), 760–811.

[62] Ziyi Li, Fan He, Peng Huang, Minjun Li, Leifeng Ruan, and Yao Dong. 2018. 5G L2 SW Architecture Best Practice on IA. Intel Corporation. White Paper.

[63] Linux. 2015. BCC: Dynamic Tracing Tools for Linux. https://iovisor.github.io/bcc/ Accessed: 2020-08-05.

[64] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge assisted real-time object detection for mobile augmented reality. In *The 25th Annual International Conference on Mobile Computing and Networking*. 1–16.

[65] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*. 450–462.

[66] Michael M Madden. 2019. Challenges Using Linux as a Real-Time Operating System. In *AIAA Scitech 2019 Forum*. 0502.

[67] Antonis Manousis, Rahul Anand Sharma, Vyas Sekar, and Justine Sherry. 2020. Contention-Aware Performance Prediction For Virtualized Network Functions. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 270–282.

[68] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C Evans, Steve Gribble, et al. 2019. Snap: a microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 399–413.

[69] Frank J Massey Jr. 1951. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American statistical Association* 46, 253 (1951), 68–78.

[70] Nicolai Meinshausen. 2006. Quantile regression forests. *Journal of Machine Learning Research* 7, Jun (2006), 983–999.

[71] RRCWireless News. 2019. Rakuten to deploy 4,000 edge servers for virtualized mobile network. Retrieved 2020-07-14 from https://www.rcrwireless.com/20190806/5g/rakuten-deploy-4000-edge-servers-virtualized-mobile-network-report

[72] Navid Nikaein. 2015. Processing radio access network functions in the cloud: Critical issues and modeling. In *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*. 36–43.

[73] ONF. 2020. SD-RAN: ONF's Software-Defined RAN Platform Consistent with the O-RAN Architecture. ONF. White Paper.

[74] OpenNESS. 2020. Open Network Edge Services Software. https://www.openness.org/ Accessed: 2020-06-02.

[75] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High {CPU} Efficiency for Latency-sensitive Datacenter Workloads. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*. 361–378.

[76] Pandas. 2019. pandas - Python Data Analysis Library. Retrieved 2020-07-27 from https://pandas.pydata.org/

[77] STL Partners. 2020. *Building Telco Edge Infrastructure: MEC, Private LTE & vRAN*. Technical Report. STL Partners, Executive Briefing.

[78] Georgios Patounas, Xenofon Foukas, Ahmed Elmokashfi, and Mahesh K Marina. 2020. Characterization and Identification of Cloudified Mobile Network Performance Bottlenecks. *IEEE Transactions on Network and Service Management* 17, 4 (2020), 2567–2583.

[79] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[80] Percona. 2008. TPCC benchmark. https://github.com/Percona-Lab/tpcc-mysql Accessed: 2020-08-05.

[81] George Prekas, Marios Kogias, and Edouard Bugnion. 2017. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 325–341.

[82] Manar Qamhieh, Frédéric Fauberteau, Laurent George, and Serge Midonnet. 2013. Global EDF scheduling of directed acyclic graphs on multiprocessor systems. In *Proceedings of the 21st International conference on Real-Time Networks and Systems*. 287–296.

[83] Henry Qin, Qian Li, Jacqueline Speiser, Peter Kraft, and John Ousterhout. 2018. Arachne: core-aware thread management. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 145–160.

[84] Rakuten. 2019. Rakuten Mobile and NEC to Build Open vRAN Architecture in Japan. Retrieved 2020-07-14 from https://global.rakuten.com/corp/news/press/2019/0605$_0$1.html

[85] Heavy Reading. 2019. New Transport Network Architectures for 5G RAN. Fujitsu. White Paper.

[86] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 446–459.

[87] Redis. 2019. How fast is Redis? Retrieved 2020-08-05 from https://redis.io/topics/benchmarks

[88] Federico Reghenzani, Giuseppe Massari, and William Fornaciari. 2019. The real-time linux kernel: A survey on preempt_rt. *ACM Computing Surveys (CSUR)* 52, 1 (2019), 1–36.

[89] Peter Rost, Salvatore Talarico, and Matthew C Valenti. 2015. The complexity–rate tradeoff of centralized radio access networks. *IEEE Transactions on Wireless Communications* 14, 11 (2015), 6164–6176.

[90] William E Ryan et al. 2004. An introduction to LDPC codes. , 23 pages.

[91] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. 2013. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems* 49, 4 (2013), 404–435.

[92] Daniel Sanchez and Christos Kozyrakis. 2011. Vantage: scalable and efficient fine-grain cache partitioning. In *Proceedings of the 38th annual international symposium on Computer architecture*. 57–68.

[93] scikit-garden. 2017. Quantile Decision Trees. https://scikit-garden.github.io/examples/QuantileRegressionForests/#quantile-decision-trees Accessed: 2020-07-27.

[94] SDKI. 2021. Virtualized RAN (vRAN) Market Size, Share & Forecast. https://www.marketwatch.com/press-release/virtualized-ran-vran-market-size-share-forecast-2025-2021-04-20 Accessed: 2021-06-4.

[95] SDXCentral. 2021. VMware Tilts Into vRAN Telco Cloud, Dish Gets First Dibs. https://www.sdxcentral.com/articles/news/vmware-tilts-into-vran-telco-cloud-dish-gets-first-dibs/2021/04/ Accessed: 2021-06-4.

[96] Xun Shao, Jinhong Yuan, and Yubin Shao. 2007. Error performance analysis of linear zero forcing and MMSE precoders for MIMO broadcast channels. *IET communications* 1, 5 (2007), 1067–1074.

[97] Xuemin Shen, Jie Gao, Wen Wu, Kangjia Lyu, Mushu Li, Weihua Zhuang, Xu Li, and Jaya Rao. 2020. AI-assisted network-slicing based next-generation wireless networks. *IEEE Open Journal of Vehicular Technology* 1 (2020), 45–66.

[98] Gábor J Székely and Maria L Rizzo. 2009. Brownian distance covariance. *The annals of applied statistics* (2009), 1236–1265.

[99] Gábor J Székely, Maria L Rizzo, Nail K Bakirov, et al. 2007. Measuring and testing dependence by correlation of distances. *The annals of statistics* 35, 6 (2007), 2769–2794.

[100] Tarik Taleb, Rui Luis Aguiar, I Grida Ben Yahia, Bruno Chatras, Gerry Christensen, Uma Chunduri, Alexander Clemm, Xavier Costa, Lijun Dong, Jaafar Elmirghani, et al. 2020. White paper on 6G networking. (2020).

[101] Kun Tan, He Liu, Jiansong Zhang, Yongguang Zhang, Ji Fang, and Geoffrey M Voelker. 2011. Sora: high-performance software radio using general-purpose

multi-core processors. *Commun. ACM* 54, 1 (2011), 99–107.
[102] Tuyen X Tran, Abolfazl Hajisami, Parul Pandey, and Dario Pompili. 2017. Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine* 55, 4 (2017), 54–61.
[103] Tuyen X Tran, Ayman Younis, and Dario Pompili. 2017. Understanding the computational requirements of virtualized baseband units using a programmable cloud radio access network testbed. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 221–226.
[104] Hoang Duy Trinh, Nicola Bui, Joerg Widmer, Lorenza Giupponi, and Paolo Dini. 2017. Analysis and modeling of mobile traffic using real traces. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, 1–6.
[105] SS Vallender. 1974. Calculation of the Wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications* 18, 4 (1974), 784–786.
[106] J-J Van De Beek, Ove Edfors, Magnus Sandell, Sarah Kate Wilson, and P Ola Borjesson. 1995. On channel estimation in OFDM systems. In *1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty-First Century*, Vol. 2. IEEE, 815–819.
[107] Jianda Wang and Yang Hu. 2019. Characterizing and Understanding the Architectural Implications of Cloudnative Edge NFV Workloads. In *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 1–7.
[108] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2018. When edge meets learning: Adaptive control for resource-constrained distributed machine learning. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 63–71.
[109] Xiaofei Wang, Min Chen, Tarik Taleb, Adlen Ksentini, and Victor CM Leung. 2014. Cache in the air: Exploiting content caching and delivery techniques for 5G systems. *IEEE Communications Magazine* 52, 2 (2014), 131–139.
[110] Ami Wiesel, Yonina C Eldar, and Shlomo Shamai. 2008. Zero-forcing precoding and generalized inverses. *IEEE Transactions on Signal Processing* 56, 9 (2008), 4409–4418.
[111] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al. 2008. The worst-case execution-time problemoverview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)* 7, 3 (2008), 1–53.
[112] Wind. 2017. *Carrier Grade Performance and Reliability in Network Virtualization*. Technical Report. Wind, Whitepaper.
[113] Fierce Wireless. 2019. Telefonica invests in vRAN vendor Altiostar. Retrieved 2020-07-14 from https://www.fiercewireless.com/tech/telefonica-invests-vran-vendor-altiostar
[114] Fierce Wireless. 2020. Dish selects Fujitsu, Altiostar for 5G radios, Open vRAN. Retrieved 2020-08-24 from https://www.fiercewireless.com/operators/dish-selects-fujitsu-altiostar-for-5g-radios-open-vran
[115] Wenfei Wu, Li Erran Li, Aurojit Panda, and Scott Shenker. 2014. PRAN: Programmable radio access networks. In *Proceedings of the 13th ACM Workshop on Hot topics in Networks*. 1–7.
[116] Dirk Wubben, Peter Rost, Jens Steven Bartelt, Massinissa Lalam, Valentin Savin, Matteo Gorgoglione, Armin Dekorsy, and Gerhard Fettweis. 2014. Benefits and impact of cloud computing on 5G signal processing: Flexible centralization through cloud-RAN. *IEEE signal processing magazine* 31, 6 (2014), 35–44.
[117] Fengli Xu, Yong Li, Huandong Wang, Pengyu Zhang, and Depeng Jin. 2016. Understanding mobile traffic patterns of large scale cellular towers in urban environment. *IEEE/ACM transactions on networking* 25, 2 (2016), 1147–1161.
[118] Chun Yeow Yeoh, Mohammad Harris Mokhtar, Abdul Aziz Abdul Rahman, and Ahmad Kamsani Samingan. 2016. Performance study of LTE experimental testbed using OpenAirInterface. In *2016 18th International Conference on Advanced Communication Technology (ICACT)*. IEEE, 617–622.
[119] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE* 107, 8 (2019), 1738–1762.
[120] Guangxu Zhu, Dongzhu Liu, Yuqing Du, Changsheng You, Jun Zhang, and Kaibin Huang. 2020. Toward an intelligent edge: Wireless communication meets machine learning. *IEEE communications magazine* 58, 1 (2020), 19–25.

A APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A.1 List of most significant vRAN signal processing tasks

For the completeness of this work and to complement the uplink DAG that was presented in Fig. 1, this appendix provides an example of a (simplified) downlink signal processing tasks DAG for 5G NR in Fig. 16. To ease the understanding of the reader, we also provide a brief description of the most significant tasks composing the uplink and downlink DAGs of the 4G and 5G vRAN. Given the large number of involved tasks (>40 in total), we only focus on describing the most computationally demanding tasks according to our own measurements and other related works [28, 72, 107]. A breakdown of the costs of the described tasks is listed in Table 5. It should be noted that the percentages of the total processing time of the tasks listed on the table are approximate. The exact compute requirements of each task depends on a number of parameters, including the cell configuration (e.g., bandwidth, number of antennas), the number of users, their transmission modes (MIMO vs SISO) etc.

Uplink tasks	
Task	**Uplink processing time %**
Decoding	More than 60%
Channel estimation	More than 8%
Equalization	More than 5%
Demodulation	More than 6%

Downlink tasks	
Task	**Downlink processing time %**
Encoding	More than 40%
Precoding	More than 15%
Modulation	More than 10%

Table 5: Breakdown of the percentage of processing time spent on the most expensive tasks for the uplink/downlink signal processing DAGs. The values show an approximate percentage for each task and in practice could differ based on the scenario.

Encoding & decoding: The encoding task is used to improve the reliability of the downlink vRAN transmissions. At a high level, this is achieved by appending redundancy bits to the original transmitted data. The proportion of the data-stream that is non-redundant designates the code rate of a transmission. For higher code rates, the utilization of the spectrum is more efficient and the computational load for generating the redundancy bits is lower. However, higher code rates also make it more difficult to recover from errors when the signal quality is low.

The decoding task is used in the uplink vRAN transmissions and it performs the exact opposite operation to the encoding task for the downlink, i.e., it uses the redundancy bits to verify the correctness of the transmitted data and, if possible, to correct any discovered errors. The decoding process in 4G and 5G is iterative and typically it stops when there is a verification of the correctness of the received data or when a certain threshold (number of iterations) is reached. Similar to the encoding, higher code rates mean lower computational cost for decoding the received data. Moreover, low signal quality can lead to a higher rate of errors on the received data and therefore more decoding iterations might be required, increasing the computational demands of the decoding task.

The encoding and decoding algorithms differ between the 4G and the 5G RAN. In the case of 4G, the algorithm used is Turbo coding [44], while in the case of 5G, LDPC coding [90] is used

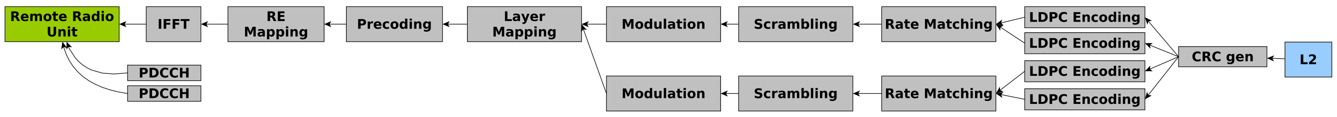


Figure 16: Example of downlink signal processing tasks DAG for 5G NR

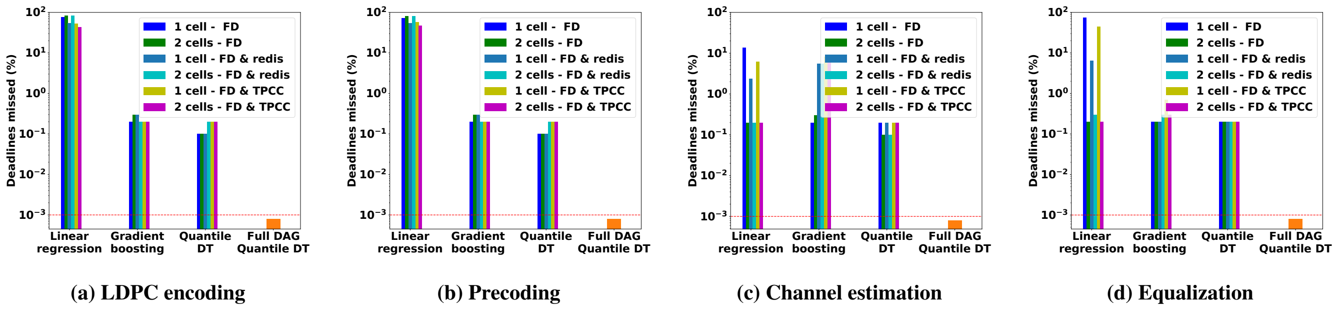
for the user data transmissions and Polar coding [4] is used for the control data.

Channel estimation: The channel estimation task is used to estimate the properties of the channel used for the transmissions/receptions. At a high level, channel estimation is performed by inserting known reference pilot symbols into a transmission and then interpolating the rest of the channel response through those pilot symbols. The computational complexity of the channel estimation task can vary depending on the algorithm used [35, 106] as well as the transmission scheme (e.g., Massive MIMO vs SISO).

Precoding & Equalization: Precoding is a downlink task that is used to realize spatial multiplexing, i.e. multi-user MIMO. The idea behind precoding is that the data streams of users are combined through a mathematical transformation and are sent out simultaneously through multiple antenna ports. This transformation relies a matrix called the *precoder* that is computed using channel state information obtained through the channel estimation task. Precoding algorithms can be divided to non-linear (e.g., DPC [22]) and linear (e.g. MMSE and Zero-Forcing [96, 110]). Linear precoding algorithms are the ones that are currently used in practice, since they provide a reasonable performance, while having a much lower complexity. Generally, the computational load of the precoding task depends on the number of transmitting antenna ports, the number of data streams and the bandwidth of the vRAN cell.

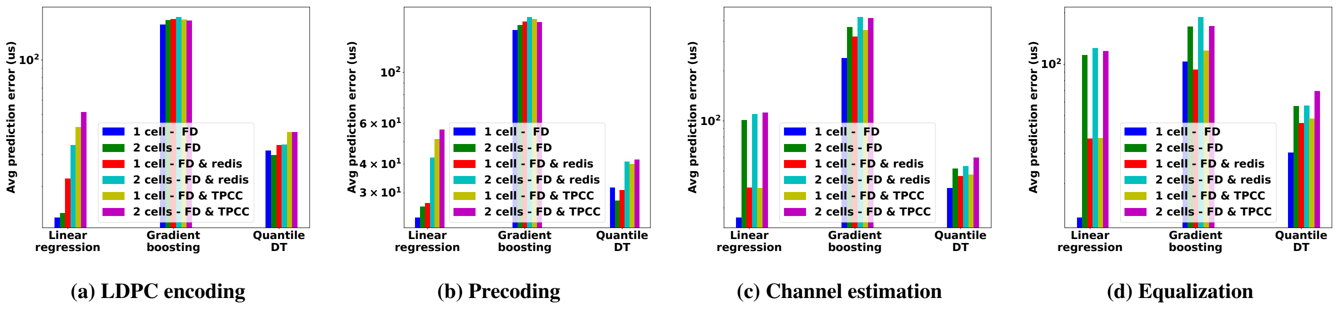
On the other hand, equalization is an uplink task that is used to demultiplex user data received from multiple antenna ports, i.e., undo the effects of the channel at the receiver side. Similar to the case of the precoding task, many different algorithms can be used for the equalization (e.g. MMSE and Zero-Forcing), each with a different complexity and performance. Also, as in the case of the precoding task, the computational load of the equalization task depends on the number of the receiving antennas, the number of data streams and the bandwidth of the vRAN cell.

Modulation & Demodulation: Modulation is a downlink signal processing task that is used to encode a sequence of bits onto the carrier signal, by adjusting the signal's amplitude and/or initial phase. Different modulation schemes exist, depending on the number of bits that can be encoded on the carrier signal at a time. For example, the binary phase shift keying (BSK) modulation scheme takes one bit at a time and transmits it using a carrier wave that can have two different states (0 or 1), while the quadrature phase shift keying (QPSK) scheme allows the transmission of two bits on the carrier wave. The



(a) LDPC encoding (b) Precoding (c) Channel estimation (d) Equalization

Figure 17: Percentage of slots where the processing deadline was violated. Y-axis in log scale.



(a) LDPC encoding (b) Precoding (c) Channel estimation (d) Equalization

Figure 18: Average WCET prediction error for successfully met deadline. Y-axis in log scale.

higher the modulation scheme, the more bits are transmitted on a radio resource block (leading to a higher user throughput), but also the higher the computational cost for performing the modulation becomes.

Demodulation is an uplink task, that performs the opposite operation to the downlink modulation, i.e., it extracts the bits that are modulated on the received radio wave. Similar to the modulation, the computational complexity of the demodulation depends on the modulation scheme, with higher schemes being more computationally demanding.

A.2 Further results on prediction model accuracy

Here, we expand the evaluation of Section 6.4 and we present additional prediction accuracy results for other computationally intensive 5G vRAN signal processing tasks; namely LDPC encoding, precoding, channel estimation and equalization. As shown in Fig. 17 and similarly to the case of LDPC decoding presented in Section 6.4, Concordia's quantile decision tree provides consistently much higher deadline prediction accuracy compared to the simpler linear regression model for all the tasks. The gradient boosting model has comparable prediction accuracy to the quantile decision tree, except for the case of the channel estimation task shown in Fig. 17c. However, as in the case of LDPC decoding, the quantile decision tree has a consistently small average WCET prediction error for all tasks, as shown in Fig 17, making it the most efficient of all studied algorithms. Similar observations are also made for all the remaining signal processing tasks.