IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 10 August 2026, accepted 26 August 2026, date of publication 31 August 2026, date of current version 4 September 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3728816

RESEARCH ARTICLE

CoDR: A Conflict Detection and Resolution Framework for Multi-rApp Deployments on the O-RAN Non-RT RIC

ROBERTO RIGGIO, **(Senior Member, IEEE)**
Universitá Politecnica delle Marche, 60131 Ancona, Italy
e-mail: r.riggio@univpm.it

ABSTRACT The O-RAN Non-Real-Time RAN Intelligent Controller enables operators to deploy multiple third-party rApps that automate radio access network (RAN) management. When rApps with competing objectives operate concurrently, their actions can conflict and silently degrade network performance, yet no framework for detecting and resolving such conflicts has been experimentally validated. We propose a four-category conflict taxonomy and design CoDR (Conflict Detection and Resolution), which intercepts rApp intents, detects conflicts via a layered pipeline, and applies configurable resolution strategies. CoDR is implemented on the O-RAN Software Community (OSC) platform and evaluated on a multi-cell simulated testbed with five concurrent rApps. Results show that unmanaged conflicts degrade cell throughput by 21% and increase handover failures by nearly 5×. CoDR detects conflicts with 92.6% precision and 86.7% recall at 12 ms median latency, and a hybrid resolution strategy recovers up to 87% of the lost throughput while keeping the mean resolution overhead at 38 ms.

INDEX TERMS O-RAN, non-RT RIC, rApp, conflict detection, conflict resolution, network automation.

I. INTRODUCTION

The Open Radio Access Network (O-RAN) architecture introduces a disaggregated, programmable framework for managing radio access networks (RANs) through open interfaces and intelligent controllers [1]. A central element of this architecture is the Non-Real-Time RAN Intelligent Controller (Non-RT RIC), a platform hosted within the Service Management and Orchestration (SMO) framework that enables the deployment of modular applications known as rApps. Each rApp encapsulates a specific network management function (e.g., energy saving, coverage optimization, traffic steering, load balancing, or QoS enforcement) and interacts with the RAN through standardized interfaces for policy guidance and configuration management [2]. By lowering the barrier for third-party innovation, the rApp model promises a rich ecosystem in which operators can compose tailored automation strategies from independently developed components.

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandro Floris.

This promise, however, carries an inherent risk. In any realistic deployment, multiple rApps will operate concurrently over overlapping sets of managed elements, each pursuing its own objective with limited or no awareness of the actions taken by its peers. An energy-saving rApp may decide to switch off lightly loaded cells at the same time that a coverage-optimization rApp attempts to increase the footprint of those very cells. A traffic-steering rApp may redirect users toward a cell that a load-balancing rApp is actively trying to offload. When such conflicts go undetected, the resulting configuration oscillations, contradictory policies, and compounding side effects can silently erode the network performance that the rApps were deployed to improve in the first place.

The problem of conflicting autonomous functions in cellular networks is not new. The Third Generation Partnership Project (3GPP) identified coordination challenges among Self-Organizing Network (SON) functions over a decade ago and introduced the SON Coordination Function as a mitigation mechanism [3]. The O-RAN setting, however, differs in important ways. rApps are developed by third

© 2026 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

parties, may arrive as opaque containers with undisclosed internal logic, and interact with the RAN through a richer set of interfaces than traditional SON functions. Moreover, the scale of the envisioned rApp ecosystem, tens of rApps from multiple vendors running on a shared platform, amplifies both the likelihood and the complexity of conflicts. The O-RAN Alliance Working Group 2 has acknowledged this challenge in its Non-RT RIC architecture specification [2], yet no concrete detection or resolution mechanism has been standardized at this tier. Recent research addressed conflict management for xApps on the Near-RT RIC [4], [5], [6]. Nevertheless, a systematic framework that classifies the conflict types specific to the rApp tier, provides detection mechanisms adapted to the Non-RT RIC's interface model, and experimentally compares resolution strategies at this layer is still lacking.

This paper addresses the gap with three contributions. First, we propose a taxonomy that classifies rApp conflicts into four categories and characterizes each category in terms of its root cause, detection difficulty, and potential impact on network performance. Second, we design CoDR, a framework that operates as a supervisory component on the Non-RT RIC, intercepts rApp intents through a centralized registry, and applies configurable resolution strategies including priority-based blocking, intent negotiation, and temporal multiplexing. Third, we implement CoDR on the O-RAN Software Community (OSC) Non-RT RIC platform and evaluate it experimentally on a multi-cell testbed driven by ns-3 and the 5G-LENA simulator [7]. Using a portfolio of five rApps with deliberately conflicting objectives, we show that unmanaged conflicts degrade average cell throughput by up to 21% and increase handover failure rates by nearly 5×, while CoDR detects conflicts with 92.6% precision and 86.7% recall at a median latency of 12 ms and recovers up to 87% of the lost throughput. A hybrid resolution strategy that maps each conflict category to its most effective mechanism achieves near-optimal performance with a mean resolution overhead of 38 ms. We further compare resolution strategies, analyze detection sensitivity to parameter choices, and derive practical design guidelines for operators managing multi-rApp deployments.

The remainder of this paper is organized as follows. Section II reviews the O-RAN Non-RT RIC architecture, illustrates sources of rApp conflict, and surveys related work. Section III presents the proposed conflict taxonomy. Section IV describes the CoDR framework architecture, detection mechanisms, and resolution strategies. Section V details the testbed implementation. Section VI presents the experimental evaluation, results, and design guidelines. Section VII concludes the paper and outlines directions for future work.

II. BACKGROUND AND RELATED WORK

A. O-RAN NON-RT RIC AND THE rApp ECOSYSTEM

The O-RAN reference architecture decomposes the RAN management stack into disaggregated components interconnected by open, standardized interfaces [1]. At the highest layer of the control hierarchy sits the SMO, which hosts the Non-RT RIC, a platform responsible for RAN optimization tasks that operate on time scales longer than one second [2]. The Non-RT RIC fulfills this role by hosting rApps: self-contained, independently deployable software modules that consume RAN data, execute management logic, and produce configuration actions or policy guidance directed at the RAN.

An rApp interacts with the Non-RT RIC platform and with the rest of the O-RAN architecture through two principal interfaces. The R1 interface connects the rApp to the services exposed by the Non-RT RIC framework, including data management and storage, AI/ML model management, and rApp lifecycle operations [8]. Through R1, an rApp can retrieve performance measurement (PM) data, access enrichment information, and register its management intents. The A1 interface, in turn, connects the Non-RT RIC to one or more Near-RT RIC instances and carries policy directives that guide the behavior of xApps operating at shorter time scales [9]. In addition, the Non-RT RIC can modify RAN configuration parameters via the O1 interface, which provides access to the configuration and performance management functions of the O-RAN managed elements. The combination of A1 policy guidance and O1 configuration changes gives each rApp a broad action space over the managed RAN.

An important architectural distinction concerns the boundary between rApp and xApp action spaces. Several RAN configuration parameters, such as cell individual offset (CIO), antenna electrical tilt, and handover hysteresis, can in principle be modified by both rApps (via O1 CM operations) and xApps (via E2 control messages). The difference is one of time scale and persistence. An rApp adjusts these parameters as persistent configuration changes based on aggregated performance data observed over multiple reporting periods; the new value remains in effect until the rApp's next decision cycle. An xApp, by contrast, may adjust the same parameter on a per-UE or per-slot basis in response to instantaneous channel measurements, with the adjustment potentially overwritten within milliseconds. The conflict scenarios studied in this paper involve rApp-level O1 configuration actions operating on cycles of 10–30 s, consistent with the Non-RT RIC's >1 s control loop. Conflicts arising from concurrent xApp-level E2 actions on the same parameters fall within the Near-RT RIC's scope and are not addressed here.

From an operational perspective, rApps are packaged as containerized workloads and deployed on the Non-RT RIC platform, which is typically realized as a Kubernetes cluster [10]. The O-RAN Alliance envisions a marketplace model in which operators select rApps from multiple vendors and onboard them through a standardized lifecycle that includes registration, instantiation, configuration, and eventual decommissioning. A key implication of this model is that the operator may have limited visibility into the internal

decision logic of a given rApp, particularly when the rApp is provided as a proprietary component by a third-party vendor.

B. SOURCES OF CONFLICT IN MULTI-rApp ENVIRONMENTS

When a single rApp operates in isolation, its actions can be validated against the operator's intent and the current RAN state in a relatively straightforward manner. The situation changes fundamentally when multiple rApps share the same Non-RT RIC instance and target overlapping sets of managed elements. We illustrate the problem through three representative scenarios that recur throughout this paper.

Scenario 1: Energy saving versus coverage optimization. Consider an energy-saving rApp (ES-rApp) that monitors cell-level traffic load and, when the load falls below a threshold, deactivates underutilized cells or reduces their transmit power to lower energy consumption [11]. Concurrently, a coverage-optimization rApp (CO-rApp) monitors reference signal received power (RSRP) reports and adjusts antenna electrical tilt and cell individual offset (CIO) parameters to eliminate coverage holes. When the ES-rApp switches off a cell, the CO-rApp may detect a new coverage gap in the area previously served by that cell and respond by increasing the tilt or CIO of neighbouring cells to compensate. If the ES-rApp subsequently reactivates the cell in response to rising load, the CO-rApp's compensatory adjustments become counterproductive, leading to overlapping coverage and pilot pollution. The two rApps enter a cycle of action and reaction, each perceiving the other's changes as disturbances that require correction.

Scenario 2: Traffic steering versus load balancing. A traffic-steering rApp (TS-rApp) aims to improve per-user throughput by directing users toward cells with favourable radio conditions, typically by adjusting handover hysteresis or CIO parameters. A load-balancing rApp (LB-rApp) pursues a complementary but potentially conflicting objective: equalizing the physical resource block (PRB) utilization across cells to prevent congestion. If the TS-rApp steers a group of users to cell A because of its superior signal quality, cell A's load increases, prompting the LB-rApp to offload users away from cell A, possibly the very users that the TS-rApp just moved. The result is a handover ping-pong that degrades user experience and wastes radio resources.

Scenario 3: QoS enforcement versus resource efficiency. A QoS-enforcement rApp (QoS-rApp) issues A1 policies that reserve resources for high-priority network slices, requesting the Near-RT RIC to guarantee minimum throughput levels. Meanwhile, a resource-efficiency rApp seeks to maximize spectral efficiency by dynamically reallocating unused PRBs across slices. The reservation imposed by the QoS-rApp constrains the flexibility available to the resource-efficiency rApp, and aggressive reallocation by the latter may violate the guarantees requested by the former. The conflict is indirect: neither rApp modifies the same parameter, but their combined effect undermines at least one of their objectives.

These scenarios share a common characteristic: each rApp behaves rationally in isolation, yet the aggregate outcome of their concurrent operation is suboptimal or even harmful. The root cause is the absence of a coordination layer that can detect when rApp actions interact and intervene before the conflicting intents reach the RAN.

C. RELATED WORK

1) SON CONFLICT MANAGEMENT

The earliest systematic treatment of conflict among autonomous RAN functions emerged in the context of 3GPP SON. The 3GPP specifications define a SON Coordination Function responsible for detecting and resolving conflicts among SON use cases such as Mobility Robustness Optimization (MRO), Mobility Load Balancing (MLB), and Coverage and Capacity Optimization (CCO) [3], [12]. The EU-funded SOCRATES project [13] provided one of the first concrete architectures for SON coordination, proposing a policy-based framework in which operator-defined rules govern the interaction among SON functions. Subsequent work by Litjens et al. [14] extended this framework with a conflict classification scheme and a coordination protocol based on function priority and temporal separation. Lateef et al. [15] propose a formal detection framework based on configuration parameter dependency analysis. While these contributions establish foundational concepts, conflict taxonomies, priority-based resolution, and temporal separation, they were designed for a closed ecosystem in which the set of SON functions is known at design time, and their internal logic is transparent to the coordination entity. Neither assumption holds in the O-RAN rApp setting.

2) POLICY CONFLICT MANAGEMENT IN SDN AND NFV

In software-defined networking, policy conflicts arise when multiple applications installed on an SDN controller issue contradictory flow rules. Hu et al. [16] formalize the problem as anomaly detection over firewall rule sets, identifying shadowing, redundancy, and correlation anomalies. Porras et al. [17] introduce FortNOX, a security enforcement kernel for OpenFlow that mediates among applications via role-based priority and real-time rule conflict detection. More recent work has extended these ideas to intent-based networking, where high-level operator intents are compiled into low-level configurations and conflicts must be detected at the intent level before compilation [18]. In the NFV domain, Boutaba et al. [19] survey intelligent network management including policy coordination among virtualised network functions. While the general principles, intent interception, conflict classification, and resolution through priority or negotiation, are transferable, the action spaces differ significantly: rApps act on RAN-specific parameters (tilt angles, CIO offsets, A1 policies) whose physical-layer semantics introduce coupling effects that have no direct analogue in packet-level flow management.

3) xApp CONFLICT MANAGEMENT ON THE NEAR-RT RIC

We identify five complementary lines of work.

Framework-based approaches. Adamczyk and Kliks [4] define a Conflict Mitigation Framework (CMF) for the Near-RT RIC that detects and resolves all conflict types specified by the O-RAN Alliance Working Group 3, including direct, indirect, and implicit conflicts. The framework defines message flows between Near-RT RIC components and validates its design through simulation. A subsequent experimental assessment on a real-world OTIC testbed with over-the-air RF transmission [20] confirms that the CMF reduces downlink throughput variability by 78%, providing the first hardware-validated evidence of conflict mitigation in O-RAN. Adamczyk [21] further discusses the practical challenges of integrating conflict mitigation into the Near-RT RIC architecture.

Game-theoretic approaches. Wadud et al. [22] propose a Conflict Mitigation Controller for the Near-RT RIC based on cooperative bargaining theory. Their subsequent work, QACM [23], extends this approach with QoS awareness, maximising the number of xApps that meet their QoS requirements during conflict resolution. QACM represents one of the most mature conflict mitigation methods for the Near-RT RIC.

Learning-based approaches. Zhang et al. [24] propose team learning based on deep Q-learning to increase cooperation between xApps for resource allocation, demonstrating improved throughput and packet drop rates compared to independently trained xApps. Iturria-Rivera et al. [25] extend this to multi-agent team learning. Erdol et al. [26] propose xApp distillation, which trains a single model that retains the capabilities of multiple xApps, eliminating conflicts by construction at the cost of requiring access to each xApp's training data. Zolghadr et al. [27] use graph neural networks to learn and reconstruct conflict patterns from observed RAN behavior.

Conflict characterisation and evaluation. del Prever et al. [5] propose PACIFISTA, a framework for characterising and evaluating conflict severity among xApps, noting that certain conflicts manifest only under specific operational conditions and that conflict severity is not binary but varies with the network state. Sharma et al. [28] combine explainable ML (SHAP) with causal inference to build directed acyclic graphs of parameter–KPI relationships, quantifying the causal impact of conflicting control actions via average and conditional treatment effects. Wadud et al. [29] provide a detailed theoretical framework of xApp-level conflicts including direct, indirect, and implicit types, validated on a mobility-driven energy saving use case.

Scheduler-based and architectural approaches. Cinemre et al. [30] propose a scheduler-based conflict mitigation framework that selects among pre-trained xApps without requiring retraining, using an Advantage Actor-Critic approach to dynamically choose which xApp to activate under given network conditions. Corici et al. [31] discuss conflict mitigation in the converged 6G Open RAN control plane, considering the interactions among xApps, rApps, and the emerging dApp tier.

4) rApp-LEVEL COORDINATION ON THE NON-RT RIC

Despite the extensive xApp-focused literature, rApp-level conflict management on the Non-RT RIC has received comparatively limited attention. Polese et al. [32] identify multi-rApp coordination as an open challenge but offer no solution. Abdalla et al. [33] note that the boundary between Non-RT RIC and Near-RT RIC responsibilities for conflict management remains underspecified. More recently, multi-agentic LLM-based approaches have been explored for conflict-aware rApp policy orchestration [34], and CollabORAN [35] proposes a hierarchical rApp–xApp–dApp framework for collaborative resource sharing, though neither provides an experimental evaluation of conflict detection and resolution at the rApp tier.

The rApp tier differs from the xApp tier in three respects that affect both detection and resolution design. First, rApps act through two distinct interfaces (A1 policies and O1 configuration), creating a broader action space with cross-interface conflict possibilities that do not arise in the xApp setting where all actions flow through E2. Second, rApp decision cycles (seconds to minutes) mean that conflicts unfold over longer horizons, requiring observation-window-based detection rather than per-slot policy inspection. Third, rApps may be delivered as opaque, third-party containers, preventing the platform from inspecting their internal objectives or predicting their next action, a constraint that rules out the glass-box conflict prediction strategies applicable when xApp logic is known. the rApp tier specifically: it provides a taxonomy of the conflict types that arise at this layer, designs detection mechanisms adapted to the Non-RT RIC's interface model and time scales, and experimentally compares resolution strategies, aspects that the existing xApp-focused literature does not cover.

5) NOVELTY OF THE PRESENT WORK

Table 1 positions CoDR against the three bodies of prior work along five dimensions. Three aspects of novelty are worth highlighting. First, CoDR is, to the best of our knowledge, the first conflict management framework designed specifically for the rApp tier of the Non-RT RIC, operating across the A1 and O1 interfaces with decision cycles of seconds to minutes, a scope not addressed by the existing xApp-focused literature, which targets per-slot E2 control on the Near-RT RIC. Second, CoDR introduces a principled conflict taxonomy derived from two orthogonal dimensions (observability and mediation mechanism) that produces four categories, each mapped to a detection mechanism of increasing computational cost. Prior O-RAN work classifies conflicts as direct, indirect, and implicit based on parameter overlap [4], [29]; our taxonomy refines this by distinguishing resource contention from objective interference (both ''indirect'' in the prior classification) and by adding temporal conflicts, which arise from the interaction of multiple rApp decision

cycles and have no counterpart in the sub-second xApp setting. Third, CoDR is the first framework to experimentally compare multiple resolution strategies (priority, negotiation, temporal multiplexing, hybrid) within a unified evaluation, whereas prior work evaluates a single method, game-theoretic bargaining [23], knowledge distillation [26], or team learning [24], against an uncoordinated baseline.

III. TAXONOMY OF rApp CONFLICTS

This section presents a taxonomy that classifies rApp conflicts into four categories based on the mechanism through which the conflict manifests. For each category we provide a definition, identify the interface and parameter scope at which the conflict occurs, illustrate the category with a concrete O-RAN example, and assess the difficulty of automated detection. Table 2 summarizes the taxonomy and serves as a reference for the detection mechanisms introduced in Section IV.

The taxonomy is motivated by a practical engineering requirement: different conflict types require fundamentally different detection mechanisms, and a layered detection pipeline that matches each mechanism to the conflicts it targets is more efficient than a monolithic detector. The four categories emerge from two orthogonal dimensions. The first is the *observability* of the conflict: whether it can be detected from the rApp action payloads alone (pre-execution) or only from the KPI outcomes after the action has taken effect (post-execution). The second is the *mediation mechanism*: whether the conflict is mediated by a shared parameter or resource that appears explicitly in the intent registry, or by the implicit physical-layer coupling and timing dynamics of the RAN. Crossing these two dimensions yields four categories: direct parameter conflicts (pre-execution, shared parameter), resource contention conflicts (pre-execution, shared resource), temporal conflicts (pre-execution, shared reconfiguration timeline), and objective interference conflicts (post-execution, physical-layer coupling). Each category maps to a detection mechanism of increasing computational cost, syntactic matching, threshold monitoring, rate monitoring, and KPI-correlation analysis, respectively, which motivates the layered pipeline in Section IV-C where lightweight checks are applied first and expensive analyses are reserved for intents that pass the earlier stages.

A. DIRECT PARAMETER CONFLICTS

A direct parameter conflict occurs when two or more rApps issue actions that target the same configuration parameter of the same managed element with contradictory values within a time window shorter than the parameter's convergence period. Formally, let r_i and r_j denote two rApps that each produce an action setting parameter p of managed element m to values v_i and v_j, respectively, where $v_i \neq v_j$. If both actions are submitted within a window Δt that is shorter than the time required for the RAN to stabilize after applying either action, the two intents are in direct parameter conflict.

This is the most straightforward category of conflict. It arises when rApps share an overlapping action space and operate on similar time scales. A typical example involves two rApps that both adjust the CIO of the same neighbour cell relation: a traffic-steering rApp increases the CIO to attract users toward a cell with favourable radio conditions, while a load-balancing rApp decreases the CIO of the same relation to push users away from that cell and relieve congestion. The conflicting parameter updates may arrive at the O1 interface in rapid succession, causing the managed element to oscillate between the two values or, depending on the implementation, to silently apply whichever update arrives last.

Direct parameter conflicts are detectable with high reliability through syntactic analysis of the rApp action streams. Because the conflict signature (i.e., same managed element, same parameter, different value) can be evaluated by inspecting the action payloads alone, detection requires no knowledge of the rApps' internal logic or objectives. The principal requirement is a centralized point of observation through which all rApp actions pass before reaching the RAN, a role that the CoDR intent registry fulfills as described in Section IV.

B. RESOURCE CONTENTION CONFLICTS

A resource contention conflict occurs when the concurrent operation of multiple rApps places competing demands on a shared resource (whether a RAN resource, a platform resource, or a data-plane resource) such that the aggregate demand exceeds the available capacity and degrades the performance of at least one rApp. Unlike direct parameter conflicts, the rApps involved in a resource contention conflict may target entirely different parameters and managed elements; the conflict materializes through the shared resource that their actions implicitly consume.

Two sub-types are worth distinguishing. The first concerns RAN-side resources. For example, an energy-saving rApp that deactivates cells effectively redistributes the traffic of those cells to the remaining active cells, increasing their physical resource block (PRB) utilization. If a QoS-enforcement rApp simultaneously reserves PRBs for a high-priority slice on the same set of active cells, the combined PRB demand may exceed the cell capacity, causing both rApps to underperform. Neither rApp modifies the other's parameters, yet the shared PRB pool mediates the conflict.

The second sub-type concerns platform-side resources on the Non-RT RIC itself. rApps that perform intensive data collection (e.g., requesting fine-grained PM counters at high reporting rates via the O1 interface) can saturate the data management services of the Non-RT RIC or the transport capacity of the O1 link. When multiple data-hungry rApps operate concurrently, the resulting contention increases data retrieval latency for all rApps, potentially causing stale inputs and degraded decision quality across the board. This platform-level contention is particularly insidious because

TABLE 1. Positioning of CoDR relative to prior conflict management approaches. ✓ = addressed, (✓) = partially addressed, — = not addressed.

	SON coord. [13]	SDN/NFV [16], [18]	O-RAN xApp [4], [23], [5]	O-RAN rApp [34], [35]	CoDR (this work)
Targets Non-RT RIC / rApp tier	—	—	—	(✓)	✓
Handles opaque third-party components	—	(✓)	(✓)	—	✓
Dual-interface action space (A1 + O1)	—	—	—	—	✓
Taxonomy with ≥4 categories	✓	—	✓	—	✓
Layered detection (cost-ordered)	—	—	—	—	✓
Observation-window KPI correlation	—	—	(✓)	—	✓
Multiple resolution strategies compared	(✓)	—	—	—	✓
Experimental eval. with ≥5 rApps	—	—	—	—	✓

it affects rApp performance without producing any visible conflict at the RAN configuration level.

Detection of resource contention conflicts requires monitoring resource utilization (PRB occupancy on the RAN side, CPU, memory, and interface throughput on the platform side) and correlating spikes with the activation or reconfiguration of specific rApps. The detection is therefore more complex than for direct parameter conflicts, as it depends on continuous telemetry rather than simple payload inspection, but it remains tractable with standard monitoring infrastructure.

C. OBJECTIVE INTERFERENCE CONFLICTS

An objective interference conflict occurs when the actions of one rApp, although targeting different parameters and consuming no shared resource, produce side effects that degrade a KPI another rApp is tasked with optimising. Because this description alone would admit any temporally coincident degradation, we adopt an operational criterion. A conflict is registered between r_j and r_i only when all of the following hold: (i) r_j executed an action at t on a managed element within the radio neighbourhood of an element monitored by r_i; (ii) some $k \in \mathcal{K}_i$ deviates from its per-element baseline by more than σ_k within $[t, t+\delta]$; (iii) the variance-normalised deviation $\rho = \Delta k/\sigma_{k_i}$ exceeds γ; and (iv) no concurrent action by r_i accounts for the deviation. Conditions (i) and (iv) exclude spatially unrelated and self-induced degradation respectively; the per-element baseline in (ii) absorbs diurnal load variation and slow traffic drift; and the conjunction of (ii) and (iii) requires both an absolute and a relative excursion, so intrinsically noisy KPIs demand proportionally larger deviations. What the criterion cannot exclude is a genuinely exogenous event, a traffic surge, a mobility-driven handover cluster, or external interference, occurring inside the observation window and coinciding with an unrelated action. Section VI-K quantifies this residual and describes two low-cost refinements that would reduce it.

The energy-saving versus coverage-optimization scenario from Section II-B is the canonical example. The ES-rApp reduces transmit power or deactivates a cell, inadvertently creating a coverage gap that the CO-rApp detects and compensates for by adjusting antenna tilt or CIO values of neighboring cells. These compensatory actions alter the interference landscape, potentially triggering further ES-rApp adjustments in subsequent cycles. The conflict is thus indirect, emergent, and mediated by the radio environment rather than by a shared parameter or resource.

Objective interference conflicts are the hardest to detect, as no syntactic signature exists in the action streams. Detection relies on temporal correlation: if rApp r_j acts at time t and a KPI monitored by rApp r_i deteriorates within a window $[t, t+\delta]$, a conflict is suspected. However, this approach is inherently probabilistic, i.e., exogenous factors such as user mobility or traffic fluctuations can produce false positives. Section IV describes how CoDR balances detection sensitivity against false-positive rate through configurable correlation thresholds and window lengths.

D. TEMPORAL CONFLICTS

A temporal conflict occurs when two or more rApp actions are individually valid and mutually compatible in steady state but interfere with each other during the transient phase of their execution. The defining characteristic is that the conflict would not exist if the actions were applied sequentially with sufficient time separation, but their near-simultaneous application creates an undesirable transient behavior.

Consider the traffic-steering and load-balancing scenario from Section II-B. Suppose that, at a given moment, both the TS-rApp and the LB-rApp independently determine that cell A should receive additional handover inflow, the TS-rApp because cell A offers the best signal quality for a cluster of users, and the LB-rApp because cell A is currently underloaded. If both rApps adjust the CIO simultaneously, the combined magnitude of the CIO change may be larger than either rApp intended, triggering a sudden influx of

handovers that temporarily overloads cell *A* and causes handover failures. Had the two adjustments been applied minutes apart, each would have taken effect against an updated RAN state and the overload would not have occurred. The conflict is therefore not in the direction of the actions but in their timing.

Temporal conflicts can also arise from reconfiguration frequency rather than magnitude. If two rApps reconfigure the same cell on alternating decision cycles with short periodicity, the cell may never reach a stable operating point, even if each individual reconfiguration is reasonable in isolation. This oscillatory behavior wastes radio resources and degrades user-perceived quality of service.

Detection of temporal conflicts relies on monitoring the reconfiguration rate per managed element, i.e., the number of configuration changes applied within a sliding time window, and flagging elements whose reconfiguration rate exceeds a stability threshold. When a spike in reconfiguration rate coincides with actions from multiple rApps, a temporal conflict is identified. The detection mechanism is straightforward to implement but requires careful calibration of the stability threshold: too low a threshold produces false positives in dynamic environments, while too high a threshold misses genuine oscillation.

E. SUMMARY AND DISCUSSION

Table 2 summarizes the four conflict categories along five dimensions: mechanism, interface, representative rApp pair, detection approach, and detection difficulty. The four-category decomposition is not merely descriptive: it is prescriptive in that each category dictates a specific detection mechanism and constrains the set of applicable resolution strategies. Collapsing any two categories would either force an unnecessarily expensive detection mechanism onto conflicts that can be caught more cheaply, or lose detection capability for conflicts whose signatures differ structurally from those of the merged category. For instance, the ES-rApp and CO-rApp can produce both an objective interference conflict (energy saving degrades coverage KPIs) and a temporal conflict (rapid cell activation–deactivation alternation). The CoDR framework presented in Section IV is designed to handle this multiplicity by running independent detection mechanisms for each category and consolidating their outputs before invoking a resolution strategy.

Two observations are worth highlighting before proceeding to the framework design. First, detection difficulty increases as the conflict mechanism becomes more indirect: direct parameter conflicts can be caught by inspecting action payloads alone, whereas objective interference conflicts require correlating actions with KPI observations over time. This gradient informs the layered detection architecture of CoDR, which applies lightweight syntactic checks first and reserves more expensive correlation-based analysis for cases that pass the initial filters. Second, the taxonomy is grounded in the current O-RAN interface model; as new interfaces or service exposure capabilities are standardized (e.g., extensions to the R1 service catalogue), additional conflict types may emerge and the taxonomy may need to be extended accordingly.

Table 3 maps our categories onto the classifications used in the O-RAN WG3 specifications and the SON literature. Three relationships are worth noting. Our taxonomy *splits* the WG3 ''indirect'' category into resource contention and objective interference, because the two admit structurally different detectors: the former is detectable pre-execution from utilisation telemetry, the latter only post-execution from KPI correlation. It *merges* the WG3 ''indirect'' and ''implicit'' categories where both concern KPI-mediated coupling, since the distinguishing criterion, whether the parameter–KPI relationship is known a priori, collapses when rApps are opaque third-party components whose internal models are never available to the platform. And it *adds* the temporal category, which has no WG3 counterpart because the steady-state/transient distinction that defines it does not arise at sub-second xApp time scales but does arise at rApp decision cycles of 10–30 s.

We claim exhaustiveness only with respect to the two dimensions from which the taxonomy is derived. All four cells of the observability × mediation grid are populated by conflict types observed experimentally, and no cell is empty or redundant. We do not claim that these two dimensions capture every possible conflict mechanism: conflicts mediated by shared data pipelines, or by semantic equivalence between nominally distinct parameters, are not represented as first-class categories and would be detected, if at all, by the objective interference stage with its associated latency and precision penalty. Extending the taxonomy would require identifying a third dimension that distinguishes detector requirements as cleanly as the two adopted here.

IV. CoDR: CONFLICT DETECTION AND RESOLUTION FRAMEWORK

This section presents the design of CoDR, a framework for detecting and resolving conflicts among concurrently operating rApps on the Non-RT RIC. We first describe the overall architecture and the role of the intent registry, then detail the detection mechanisms for each conflict category identified in Section III, and finally discuss the resolution strategies that CoDR can apply once a conflict has been identified.

A. ARCHITECTURE OVERVIEW

CoDR is positioned as a supervisory component that mediates between the rApp tier and the southbound interfaces of the Non-RT RIC. Fig. 1 illustrates the high-level architecture. Rather than allowing rApp actions to flow directly to the A1 and O1 interfaces, CoDR interposes an intent interception layer that captures every action before it reaches the RAN. Each intercepted action is recorded in the *intent registry*, a structured data store that maintains a live view of all pending and recently executed rApp intents. The detection engine continuously analyses the contents of the intent

TABLE 2. Summary of the proposed rApp conflict taxonomy. *Preventive* detection evaluates an intent while its status is `PENDING` and can stop it before it reaches the RAN; *retrospective* detection identifies the conflict only after the action has been applied, so resolution is necessarily corrective (rollback, suppression of subsequent actions, or operator escalation).

Category	Mechanism	Interface	Example rApp pair	Detection approach	Detection point	Detection difficulty
Direct parameter	Same parameter, same element, contradictory values	O1, A1	TS-rApp vs. LB-rApp (CIO)	Syntactic matching on intent registry	Preventive	Low
Resource contention	Aggregate demand on shared resource exceeds capacity	O1, platform	ES-rApp vs. QoS-rApp (PRBs)	Utilisation projection against threshold	Preventive	Medium
Objective interference	Side effects of one rApp degrade another rApp's KPI	O1, A1	ES-rApp vs. CO-rApp (coverage vs. energy)	Windowed KPI deviation scoring	**Retrospective**	High
Temporal	Actions compatible in steady state but harmful during transient	O1, A1	TS-rApp + LB-rApp (simultaneous CIO)	Reconfiguration rate monitoring	Preventive	Medium

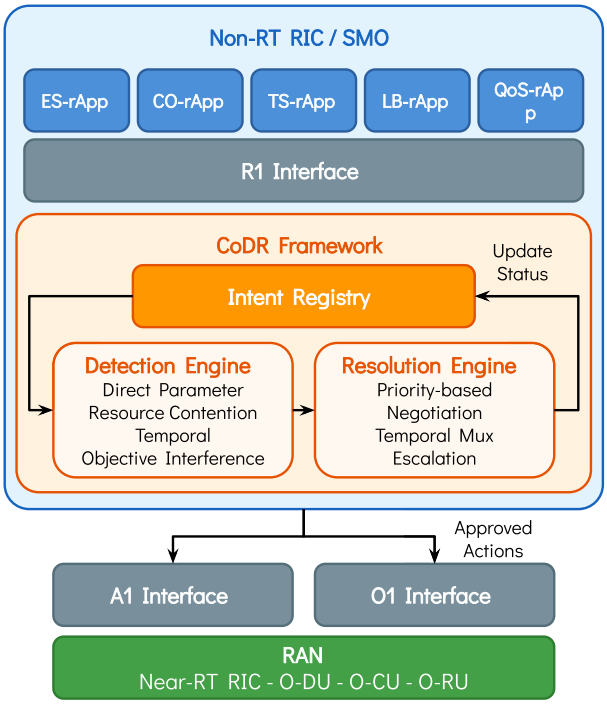

FIGURE 1. High-level architecture of the CoDR framework. rApp actions are intercepted by the intent registry before reaching the A1 and O1 interfaces. The detection engine analyses the registry contents and RAN telemetry to identify conflicts, and the resolution engine applies a configurable strategy before forwarding cleared actions to the RAN.

registry, together with RAN telemetry and platform metrics, to identify potential conflicts. When a conflict is detected, the resolution engine selects and applies a resolution strategy according to operator-defined policies. Only actions that have cleared the detection and resolution stages are forwarded to the RAN through the standard A1 and O1 paths.

From a deployment perspective, CoDR can be realized in three ways, which differ in the strength of the interception guarantee they provide. The first is *cooperative routing*: rApps are configured at onboarding to direct their A1 and O1 actions to CoDR's endpoint rather than to the A1 Policy Management Service or the O1 mediator. This requires no platform modification and is the approach used in our prototype, but it provides no enforcement, i.e., an rApp that ignores the configuration and calls the southbound services directly bypasses the registry entirely. The second is *substrate-level enforcement*: Kubernetes NetworkPolicy rules deny egress from rApp pods to the A1-PMS and O1 mediator, and a service mesh sidecar transparently redirects those calls to CoDR. This enforces interception for arbitrary rApps without modifying the OSC source, at the cost of requiring service-mesh infrastructure and elevating CoDR to a privileged position in the cluster network policy. The third is *platform integration*: adding a gating hook to the A1-PMS and O1 mediator so that policy and configuration requests are validated against the registry before being applied. This provides the strongest guarantee but requires upstream changes to the platform. The R1 interface as currently specified does not provide an action-mediation capability, so none of the three options is achievable through R1 alone.

Our prototype uses cooperative routing: the five rApps are configured to submit their intents to CoDR's REST endpoint, which forwards approved actions to the A1-PMS and O1 mediator. We did not implement substrate-level enforcement, so the evaluation assumes rApp compliance with the routing configuration. Validating CoDR against non-compliant rApps would require the NetworkPolicy and service-mesh arrangement described in Section IV-A.

B. INTENT REGISTRY

The intent registry is the central data structure of CoDR. It stores a record for every rApp action that has been intercepted, including actions that are pending evaluation, actions that have been approved and forwarded, and actions

TABLE 3. Mapping of the proposed taxonomy onto the O-RAN WG3 and 3GPP SON conflict classifications. Split indicates that one prior category is subdivided; merge indicates that two prior categories collapse; new indicates a category with no prior counterpart.

This work	O-RAN WG3 [4], [29]	3GPP SON [3]	Relation	Detection point	Rationale for the mapping
Direct parameter	Direct	Configuration conflict	1:1	Pre-exec.	Identical semantics: two actions assign contradictory values to the same parameter of the same managed element. Detectable by syntactic matching in all three schemes.
Resource contention	Indirect	Characteristic conflict (subset)	**Split**	Pre-exec.	Aggregate demand on a shared resource exceeds capacity. Separated from objective interference because it is detectable *before* execution by projecting demand against a utilisation threshold using O1 PM data.
Objective interference		Characteristic / measurement conflict		Post-exec.	Side effects of one rApp degrade another's KPI via physical-layer coupling. Detectable only *after* execution through KPI correlation over an observation window, requiring a structurally different detector.
	Implicit	—	**Merge**	Post-exec.	WG3 separates implicit from indirect by whether the parameter–KPI relationship is known a priori. This distinction collapses for opaque third-party rApps, whose internal models are never available to the platform; both present identically to the detector.
Temporal	—	(partially, undeveloped)	**New**	Pre-exec.	Actions that are individually valid and compatible in steady state interfere during transients, via magnitude superposition or reconfiguration oscillation. No WG3 counterpart: at sub-second xApp time scales, actions are instantaneous relative to RAN convergence, so the steady-state/transient distinction does not arise. It does arise at rApp cycles of 10–30 s.

that have been blocked or deferred by the resolution engine. Each record contains the following fields:

- **rApp identifier**: the unique identifier of the rApp that issued the action.
- **Action type**: whether the action is an A1 policy directive or an O1 configuration change.
- **Target scope**: the set of managed elements affected by the action, identified by cell, node, or slice identifiers as applicable.
- **Parameter set**: the specific parameters modified by the action together with their intended values.
- **Timestamp**: the time at which the action was intercepted.
- **Status**: one of `PENDING`, `APPROVED`, `BLOCKED`, or `DEFERRED`.
- **Declared objective** (optional): a high-level label indicating the rApp's optimization goal, if provided by the rApp developer at onboarding time.
- **Conflict annotations**: references to any conflicts detected against this action, populated by the detection engine.

The registry exposes query interfaces that allow the detection engine to retrieve actions by target scope, by parameter name, by rApp identifier, or by time window. A configurable retention policy governs how long completed records are kept; retaining a history of recent actions is necessary for temporal conflict detection and for objective interference correlation, as both mechanisms require access to past actions.

The registry data model is intentionally kept simple. We represent each record as a JSON document, which aligns with the data formats used by the OSC Non-RT RIC and facilitates integration with existing R1 data management services.

C. CONFLICT DETECTION ENGINE

The detection engine implements four detection mechanisms, one for each conflict category in the taxonomy. The mechanisms are organized in a layered pipeline that applies lightweight checks first and progressively engages more expensive analyses, as illustrated in Fig. 2. When a new action enters the intent registry with status `PENDING`, the detection engine evaluates it through the following stages.

The first three stages are *preventive*: they evaluate an intent while its status is `PENDING` and can prevent it from reaching the RAN. The fourth is *retrospective*: it evaluates the observable consequences of an intent that has already been approved and applied, and can therefore only trigger corrective action. This asymmetry is inherent to the conflict types rather than an implementation choice, an objective interference conflict has no signature until the action has perturbed the network, and it propagates into the resolution semantics described in Section IV-D.

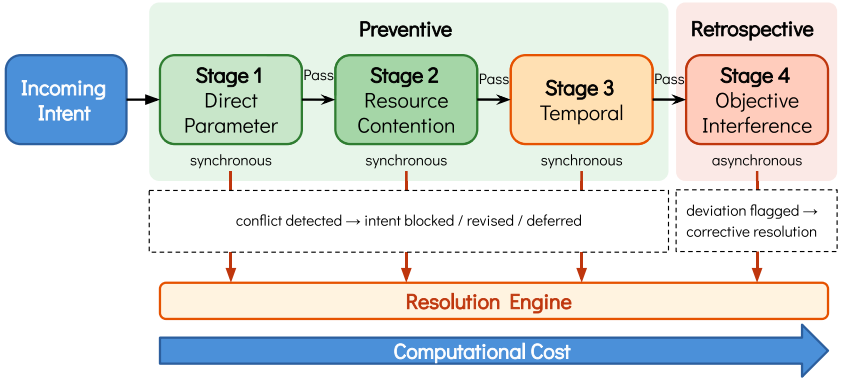

FIGURE 2. Layered detection pipeline. Each incoming intent is evaluated by the four detection mechanisms in order of increasing computational cost. A conflict detected at any stage triggers the resolution engine.

1) RATIONALE FOR RULE-BASED DETECTION

The four detection mechanisms are deterministic and rule-based rather than learned. This choice is motivated by four properties of the Non-RT RIC deployment context.

First, the *cost asymmetry between conflicting and non-conflicting intents* favours cheap deterministic checks. In our experiments, approximately 87% of intents are non-conflicting, yet every intent must traverse the pipeline before reaching the RAN. An indexed registry lookup on the (m, p) key completes in single-digit milliseconds, whereas a learned classifier would incur feature extraction and inference cost on every intent, including the large majority for which no conflict exists.

Second, *labelled training data is unavailable at deployment time*. A supervised conflict detector requires examples of conflicting and non-conflicting intent pairs, but conflicts must be observed, and therefore permitted to occur, before they can be labelled. The offline oracle used in Section VI constructs labels from isolated-execution counterfactuals, information that is not available to a runtime detector.

Third, *auditability is a requirement, not a preference*. When CoDR escalates a conflict to the operator (Section IV-D), the escalation report must justify the flag. Rule-based detection produces an inherently interpretable justification, the matched parameter, the conflicting values, and the threshold exceeded, whereas a learned detector would require post-hoc explanation whose fidelity is itself uncertain.

Fourth, *two of the four conflict categories admit exact definitions*. A direct parameter conflict is, by definition, two contradictory values assigned to the same (m, p) pair within a convergence window; a temporal conflict is, by definition, a reconfiguration rate exceeding a stability threshold with contributions from multiple rApps. These are deterministic predicates, and applying a probabilistic classifier to them would introduce classification error where none is intrinsic to the problem.

2) DIRECT PARAMETER CONFLICT DETECTION

The first stage performs syntactic matching against the intent registry. For each incoming action targeting managed element m and parameter p, the detector queries the registry for any other action with status `PENDING` or recently `APPROVED` that targets the same (m, p) pair. If a match is found and the intended parameter values differ, a direct parameter conflict is flagged. The comparison accounts for parameter semantics where applicable: for numerical parameters such as CIO offsets or transmit power levels, a configurable tolerance threshold ϵ determines whether two values are considered contradictory, avoiding false positives from negligible differences.

This stage is computationally inexpensive, it reduces to an indexed lookup on the (m, p) key, and is expected to resolve the majority of direct parameter conflicts with high precision. Its limitation is that it cannot detect conflicts involving different parameters that have equivalent effects (e.g., two distinct parameters that both influence handover behavior); such cases fall under the objective interference category and are handled by a later stage.

3) RESOURCE CONTENTION DETECTION

The second stage evaluates whether the incoming action, combined with currently active actions, would cause aggregate resource demand to exceed a capacity threshold. CoDR monitors two classes of resources. On the RAN side, per-cell PRB utilization is tracked through periodic PM reports. When a new action is expected to increase the load on a cell, for instance, a traffic-steering action that redirects users toward it, the detector estimates the resulting PRB utilization by combining the current utilization with a load-impact estimate derived from historical data. If the projected utilization exceeds a configurable threshold θ_{RAN}, a resource contention conflict is flagged with respect to any other rApp that has recently placed load-increasing actions on the same cell or that holds active resource reservations (e.g., A1 QoS policies) on that cell.

On the platform side, CoDR monitors the CPU, memory, and O1 interface throughput consumed by each rApp's data collection activities. If the aggregate platform resource consumption approaches a threshold θ_{platform}, newly arriving data-intensive actions are flagged for resource contention.

The platform-side monitoring reuses standard Kubernetes metrics (CPU and memory usage per pod) and O1 interface counters exposed by the Non-RT RIC framework.

The load-impact estimation introduces a degree of approximation. Rather than building a detailed traffic model, CoDR uses a simple empirical approach: for each action type (e.g., CIO increase of $+x$ dB on a given neighbour relation), it maintains a running average of the PRB utilization change observed in past instances of the same action type on similar cells. This historical lookup is lightweight and sufficient for flagging likely contention; precise capacity planning remains the responsibility of the operator's network design processes.

4) TEMPORAL CONFLICT DETECTION

The third stage monitors the reconfiguration rate of each managed element. The detector maintains, for every managed element m, a counter $c_m(w)$ that records the number of configuration changes applied to m within a sliding time window of length w. When $c_m(w)$ exceeds a stability threshold τ, the detector examines the contributing actions. If the actions originate from two or more distinct rApps, a temporal conflict is flagged. If all actions originate from a single rApp, the situation is logged as potential instability within that rApp but is not classified as an inter-rApp conflict.

The choice of window length w and threshold τ involves a trade-off explored experimentally in Section VI. A short window with a low threshold maximizes sensitivity to rapid oscillations but may generate false positives in legitimately dynamic scenarios (e.g., a sporting event causing rapid load fluctuations). A longer window with a higher threshold reduces false positives at the cost of slower detection.

5) OBJECTIVE INTERFERENCE DETECTION

The fourth and most computationally demanding stage evaluates whether recent actions by one rApp have degraded the KPIs associated with another rApp. This stage operates on a longer time horizon than the previous three, as the chain from action to observable KPI impact may span multiple PM reporting periods.

The mechanism proceeds as follows. CoDR maintains, for each active rApp r_i, a set of monitored KPIs $\mathcal{K}_i$ together with baseline values computed over a recent stable period. When rApp r_j ($j \neq i$) executes an action at time t, the detector opens an observation window $[t, t+\delta]$ and monitors the KPIs in $\mathcal{K}_i$. If any KPI $k \in \mathcal{K}_i$ deviates from its baseline by more than a configurable sensitivity threshold σ_k during the observation window, CoDR computes the deviation score

$$\rho = \frac{|k_{\text{obs}} - \bar{k}_i|}{\sigma_{k_i}},$$

where k_{obs} is the KPI mean over the observation window and $(\bar{k}_i, \sigma_{k_i})$ are the baseline mean and standard deviation for that KPI on that managed element. If ρ exceeds a threshold γ, a candidate objective interference conflict is flagged between r_j and r_i.

This approach is deliberately kept simpler than a full causal inference methodology. While techniques such as Granger causality testing or structural causal models could provide stronger attribution guarantees, they require substantially more data and computational resources than are practical for a real-time detection component on the Non-RT RIC. The sliding-window correlation approach offers a pragmatic compromise: it is implementable with the PM data already available through the O1 interface and provides actionable conflict signals at the cost of occasional false positives, whose rate is governed by the σ_k and γ parameters. Section VI quantifies this trade-off experimentally.

6) CAUSAL JUSTIFICATION AND ITS LIMITS

The detector addresses a counterfactual question (would rApp r_i's KPI have deviated had rApp r_j not acted?) using a correlational estimate. Three structural features of the O-RAN setting make this estimate stronger than a naive association measure. First, the *temporal ordering is unambiguous*: the rApp action is a discrete, timestamped intervention recorded in the intent registry before the observation window opens, so reverse causation is excluded by construction. Second, the *baseline provides an explicit counterfactual*: the deviation Δk is measured against a mean and standard deviation computed over a recent stable period for the same managed element, not against a global average. Third, the *correlation score normalises by intrinsic variability*: since $\rho = \Delta k/\sigma_{k_i}$ is expressed in units of the KPI's own baseline standard deviation, a noisier KPI requires a proportionally larger deviation to trigger a flag, and the separate σ_k threshold imposes an additional absolute-magnitude condition. A conflict is flagged only when both conditions hold.

What the detector cannot address is *confounding*. An exogenous event, a traffic surge, a mobility-driven handover cluster, or interference from a neighbouring network, occurring within the observation window produces the same signature as an rApp-induced degradation. The detector has no mechanism to distinguish the two, and this is the dominant source of the false positives reported in Section VI. We deliberately accept this limitation rather than adopting a full causal inference procedure such as Granger causality testing or a structural causal model, both of which require substantially more historical data per rApp pair and computational resources than are practical for a component that must operate continuously on the Non-RT RIC. The parameters σ_k and γ expose the resulting precision–recall trade-off to the operator, and Experiment 5 characterises it empirically.

D. RESOLUTION ENGINE

Once the detection engine flags a conflict, the resolution engine must decide how to handle the conflicting actions. CoDR supports four resolution strategies, each suited to different operational contexts. The operator selects the active

strategy, or a combination of strategies, through a CoDR configuration policy at deployment time.

The resolution semantics differ according to the detection point. For preventively detected conflicts, resolution acts on a pending intent: priority-based blocking sets its status to `BLOCKED` and it is never applied; negotiation returns a constrained intent before any RAN exposure; temporal multiplexing defers it to a later window. For retrospectively detected conflicts the offending action has already been applied, and resolution is necessarily corrective: CoDR can roll back the applied configuration to its prior value, suppress subsequent actions from the implicated rApp on the affected elements, or escalate to the operator. Blocking in the preventive sense is unavailable, and the RAN is exposed to the conflict for at least the duration of the observation window δ.

1) PRIORITY-BASED BLOCKING

In this strategy, each rApp is assigned a static priority level at onboarding time, reflecting the operator's assessment of the relative importance of its optimization objective. When a conflict is detected, the action of the lower-priority rApp is blocked (its status in the intent registry is set to `BLOCKED`), while the higher-priority action proceeds. If both rApps share the same priority level, the action that arrived first is given precedence.

Priority-based blocking is the simplest resolution strategy and introduces minimal additional latency, as the resolution decision is immediate once priorities are compared. Its limitation is rigidity: the priority ordering is static and does not adapt to changing network conditions. An rApp that is low-priority in general may become critical during specific periods (e.g., an energy-saving rApp during off-peak hours), and a fixed priority scheme cannot capture such context dependence without manual reconfiguration by the operator.

2) INTENT NEGOTIATION

Rather than unilaterally blocking the lower-priority action, the negotiation strategy attempts to find a compromise. When a conflict is detected, CoDR sends a *constraint notification* to the conflicting rApp via the R1 interface, informing it of the conflict and requesting a revised action that satisfies specified bounds. For example, if a traffic-steering rApp's CIO adjustment conflicts with a load-balancing rApp's action on the same cell, CoDR may inform the traffic-steering rApp that the CIO increase must not exceed a certain value to keep the projected cell load below the contention threshold. The rApp then has the opportunity to recompute its action within the communicated constraints and resubmit it.

Negotiation preserves a degree of autonomy for both rApps and can yield outcomes that satisfy both objectives partially, rather than sacrificing one entirely. The trade-off is increased resolution latency: the round-trip time for the constraint notification, rApp recomputation, and resubmission adds delay before the action reaches the RAN. Moreover, negotiation requires that rApps implement a constraint-handling interface, which imposes a design requirement on rApp developers. In our prototype, we realize this interface as an optional R1 service endpoint; rApps that do not implement it fall back to priority-based blocking when they are involved in a conflict.

3) TEMPORAL MULTIPLEXING

Temporal multiplexing resolves conflicts by separating the execution of conflicting actions in time rather than suppressing one of them. CoDR assigns non-overlapping execution windows to the conflicting rApps, ensuring that only one rApp's actions are applied during any given period. The execution windows can be defined as fixed time slots (e.g., ES-rApp active during 00:00–06:00, CO-rApp active during 06:00–24:00) or as alternating decision cycles of configurable duration.

This strategy is particularly well-suited for conflicts between rApps whose objectives are naturally time-dependent, such as energy saving (most valuable during off-peak hours) and coverage optimization (most critical during peak hours). It avoids the all-or-nothing outcome of priority-based blocking and requires no cooperation from the rApps, as the multiplexing is enforced externally by CoDR. The limitation is that temporal separation reduces the effective operating time of each rApp, which may not be acceptable for rApps that must respond to events in real time regardless of the time slot.

4) OPERATOR ESCALATION

For conflicts that cannot be resolved automatically (because the conflicting rApps have equal priority and do not support negotiation, because the operator has not defined a temporal multiplexing schedule, or because the detection engine has flagged a potential objective interference conflict with low confidence) CoDR escalates the conflict to the operator through the SMO dashboard. The escalation report includes the identities of the conflicting rApps, the conflict category, the affected managed elements, the detection confidence score, and a recommended resolution action. The conflicting intents are held in `DEFERRED` status until the operator provides a resolution directive.

Operator escalation is the strategy of last resort. Its purpose is to ensure that CoDR never silently drops a conflict signal, even when it lacks sufficient information or authority to resolve the conflict autonomously. In practice, frequent escalations would indicate that the operator's conflict management policies are insufficiently specified and need refinement.

E. CONFLICT RESOLUTION WORKFLOW

Fig. 3 summarizes the end-to-end conflict resolution workflow. When a new action arrives, it is recorded in the intent registry and passed through the detection pipeline. If no conflict is detected, the action is approved and forwarded immediately. If a conflict is detected, the resolution engine evaluates the applicable strategy based on the conflict category and the operator's policy configuration. The operator

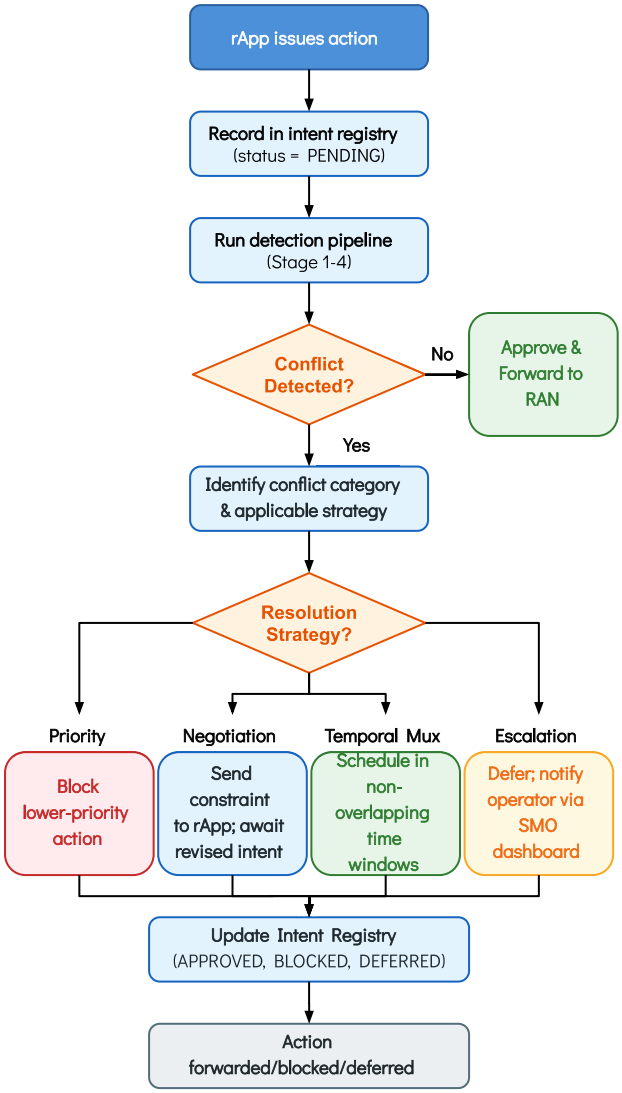

FIGURE 3. End-to-end conflict resolution workflow. The operator configures per-category resolution policies. The detection pipeline and resolution engine process each incoming action accordingly.

can define per-category resolution policies, for instance, priority-based blocking for direct parameter conflicts, negotiation for resource contention, temporal multiplexing for known objective interference pairs, and escalation as the default fallback. The resolution engine applies the configured strategy, updates the intent registry with the outcome, and either forwards the approved action, rejects the blocked action, or defers the action pending negotiation or operator input.

The mapping from conflict category to resolution strategy is motivated by the structural properties of each category. Direct parameter and temporal conflicts involve clear-cut situations, contradictory values for the same parameter or excessive reconfiguration rate, where one action must yield entirely; priority-based blocking resolves these instantly with negligible overhead, and no intermediate compromise exists that would satisfy both intents. Resource contention conflicts involve aggregate demand exceeding a capacity threshold, which admits quantitative compromise: both rApps can partially achieve their objectives if each reduces its resource claim, making negotiation effective. Objective interference conflicts are detected asynchronously and arise from physical-layer coupling between rApp objectives that cannot be resolved by parameter-level adjustment; temporal separation eliminates the coupling by ensuring that the conflicting rApps do not act on the RAN simultaneously. The hybrid policy encodes this reasoning as a static per-category mapping with four entries, independent of the number of deployed rApps. Operators can optionally specify per-pair exceptions (e.g., always use negotiation for the TS–LB pair regardless of conflict category), but such overrides are not required and the default per-category mapping applies to all pairs not explicitly listed.

A key design principle is that CoDR introduces latency only for actions that are involved in a conflict. Actions that pass all four detection stages without triggering a conflict flag are forwarded to the RAN with negligible additional delay (the syntactic matching and threshold checks in the first three stages execute in the order of milliseconds). The objective interference stage, which is the most expensive, operates asynchronously: it does not block the forwarding of actions at submission time but may retroactively flag a conflict and trigger a corrective resolution (e.g., rolling back a recently approved action or adjusting the parameters of a subsequent action) if a KPI correlation is detected after the fact. This asynchronous design ensures that CoDR does not become a bottleneck in the action path for the common case in which no conflict exists.

F. COMPUTATIONAL COMPLEXITY ANALYSIS

This section analyses the asymptotic cost of the four detection stages and the resolution engine. Table 4 summarises the results. We use the following notation: n is the number of deployed rApps; m is the number of managed elements (cells); p is the number of distinct configuration parameters per element; λ is the aggregate intent arrival rate (intents per second); R is the number of records retained in the intent registry; $|\mathcal{K}|$ is the average number of KPIs monitored per rApp; and δ, w are the observation and temporal windows defined in Section IV-C.

1) REGISTRY SIZE

The intent registry retains records for a configurable duration $T_{\text{ret}} \geq \max(w, \delta)$. With intent arrival rate λ, the steady-state registry size is

$$R = \lambda \cdot T_{\text{ret}}. \tag{1}$$

The arrival rate is determined by the rApp portfolio: each rApp r_i with decision cycle c_i targeting an average of s_i managed elements per cycle contributes s_i/c_i intents per

second, giving

$$\lambda = \sum_{i=1}^{n} \frac{s_i}{c_i} = O(n). \tag{2}$$

Registry size therefore grows linearly in n, and memory consumption is $O(n \cdot T_{\text{ret}})$. This matches the linear memory growth observed in Experiment 4 (Section VI-G).

2) STAGE 1: DIRECT PARAMETER DETECTION

The detector performs a lookup on the composite key (m, p) using a secondary index maintained by the registry. With hash-based indexing, the expected lookup cost is $O(1)$, returning a bucket of size

$$b_{m,p} = \frac{R}{m \cdot p}, \tag{3}$$

assuming intents are distributed uniformly across managed elements and parameters. The detector compares the incoming value against each record in the bucket, giving per-intent cost

$$C_1 = O\left(\frac{R}{m \cdot p}\right) = O\left(\frac{n \cdot T_{\text{ret}}}{m \cdot p}\right). \tag{4}$$

For realistic deployments where $m \cdot p \gg n \cdot T_{\text{ret}}$ (e.g., $m = 21$ cells, $p \approx 8$ parameters, $n = 5$ rApps, $T_{\text{ret}} = 60$ s), the bucket contains a small constant number of records and C_1 is effectively $O(1)$. Importantly, this stage does *not* enumerate rApp pairs: it inspects only intents that collide on the same key, which is why its cost is independent of n^2.

3) STAGE 2: RESOURCE CONTENTION DETECTION

The RAN-side check retrieves the current PRB utilisation for the target element ($O(1)$ lookup) and queries the registry for concurrently active load-increasing intents on that element. The latter is again a bucketed lookup, giving $O(R/m)$. The load-impact estimate is a historical average lookup, $O(1)$. The platform-side check queries per-pod metrics for all n rApps, giving $O(n)$. Total per-intent cost is

$$C_2 = O\left(\frac{R}{m} + n\right) = O(n), \tag{5}$$

since $R = O(n)$ by (1). The platform-side check can be amortised by caching aggregate metrics across intents arriving within the same metrics sampling interval, reducing the effective cost to $O(R/m)$.

4) STAGE 3: TEMPORAL DETECTION

The reconfiguration counter $c_m(w)$ is maintained incrementally with TTL-based expiry, so incrementing and reading it is $O(1)$. When the threshold τ is exceeded, the detector enumerates the contributing rApps within the window, which is bounded by τ (the number of reconfigurations that triggered the check) rather than by n. Total per-intent cost is

$$C_3 = O(1) \text{ amortised}, \quad O(\tau) \text{ worst case}. \tag{6}$$

Since τ is a small operator-configured constant (5 in our experiments), this stage is effectively constant-time.

TABLE 4. Per-intent computational cost of the CoDR detection stages and aggregate cost per unit time.

Stage	Per intent	Per unit time	Path
1. Direct parameter	$O(R/mp) \approx O(1)$	$O(n)$	sync
2. Resource contention	$O(n)$	$O(n^2)$	sync
3. Temporal	$O(1)$ amortised	$O(n)$	sync
4. Objective interf.	$O(n\lvert\mathcal{K}\rvert)$	$O(n^2\lvert\mathcal{K}\rvert)$	async
Resolution engine	$O(1)$	$O(n)$	sync
Registry (memory)	—	$O(n \cdot T_{\text{ret}})$	—

5) STAGE 4: OBJECTIVE INTERFERENCE DETECTION

This is the only stage whose cost depends on the number of rApp pairs. For each approved intent from rApp r_j, the detector opens an observation window and evaluates the KPIs of every other active rApp, giving

$$C_4 = O(n \cdot |\mathcal{K}|) \tag{7}$$

per intent. Since the intent arrival rate is itself $O(n)$ by (2), the aggregate cost per unit time is

$$\Lambda_4 = \lambda \cdot C_4 = O\left(n^2 \cdot |\mathcal{K}|\right), \tag{8}$$

confirming the quadratic growth in the number of rApps that the taxonomy anticipates. Critically, however, this cost is incurred *asynchronously*: the observation callbacks execute outside the intent forwarding path and therefore do not contribute to the end-to-end action latency. Their impact is on CPU utilisation and on the memory required to hold pending observation contexts, the latter being $O(\lambda \cdot \delta) = O(n \cdot \delta)$.

6) RESOLUTION ENGINE

Priority-based resolution compares two priority values, $O(1)$. Temporal multiplexing evaluates a time-window predicate, $O(1)$. Negotiation incurs a network round-trip whose cost is dominated by latency rather than computation, contributing $O(1)$ CPU cost but adding $O(\text{RTT})$ to the resolution latency, as measured in Experiment 3. Operator escalation constructs a report, $O(1)$. The resolution engine is therefore not a scalability bottleneck.

7) SUMMARY AND BOTTLENECK IDENTIFICATION

Three conclusions follow. First, the *synchronous path*, the sequence of operations that must complete before an intent is forwarded to the RAN, has per-intent cost $O(n)$, dominated by Stage 2's platform-side metrics query. With metrics caching, this reduces to effectively $O(1)$. This explains the near-constant median detection latency (8–12 ms) observed across 2–5 rApps in Experiment 4: the synchronous path does not degrade with portfolio size.

Second, the *quadratic term arises exclusively in Stage 4* and manifests as CPU load rather than as action latency, because the objective interference detector operates asynchronously. This is a deliberate architectural choice: placing the only super-linear component off the critical path bounds

the latency impact of scaling while accepting a growing background CPU cost.

Third, the aggregate CPU cost is asymptotically dominated by $\Lambda_4 = O(n^2|\mathcal{K}|)$, yet this is *not* visible in the Experiment 4 measurements. Fitting both a linear and a quadratic model to the four measured points (3.2%, 4.8%, 6.5%, 8.1% at $n = 2, \ldots, 5$) yields a quadratic coefficient $\alpha \approx 0$, and the linear model CPU(n) $\approx 1.64n$ explains the data equally well. We interpret this as evidence that the tested range is too narrow for the quadratic term to dominate: at $n = 5$ the constant and linear components, registry maintenance, metrics polling, and the three synchronous stages, still account for the bulk of the CPU cost, masking the $O(n^2)$ contribution from Stage 4. This observation is consistent with the analysis rather than contradicting it: a quadratic function with a small leading coefficient is indistinguishable from a linear one over a short interval. It also means the measurements cannot be used to extrapolate to larger portfolios, since a linear extrapolation would underestimate the true cost precisely where the quadratic term begins to dominate. Determining the crossover point at which Stage 4 becomes the dominant CPU consumer requires measurements at $n \geq 10$, which we identify as necessary future work in Section VII.

8) MITIGATING THE QUADRATIC TERM

Three architectural refinements would bound the Stage 4 cost. First, *scope-based pruning*: the detector need only evaluate rApps whose monitored KPIs pertain to managed elements within the radio neighbourhood of the acting rApp's target. Since RAN coupling decays with distance, restricting evaluation to k-hop neighbours reduces the effective pair count from $O(n^2)$ to $O(n \cdot \bar{d})$, where $\bar{d}$ is the average neighbourhood degree. Second, *declared objective filtering*: if rApps declare their optimisation objectives at onboarding (as recommended in Section VI-K), the detector can skip pairs whose objectives are known to be non-interacting. Third, *sampling*: rather than opening an observation window for every approved intent, the detector could sample a fraction of intents, trading detection latency for CPU cost in a manner analogous to statistical process control. We consider the first of these the most promising, since RAN spatial locality is a robust property that requires no additional rApp cooperation.

V. IMPLEMENTATION

This section describes the testbed, the rApp portfolio, and the CoDR prototype used in the experimental evaluation.

A. TESTBED ARCHITECTURE

The testbed comprises three layers: a simulated RAN, a Non-RT RIC platform, and the CoDR framework itself. Fig. 4 provides an overview.

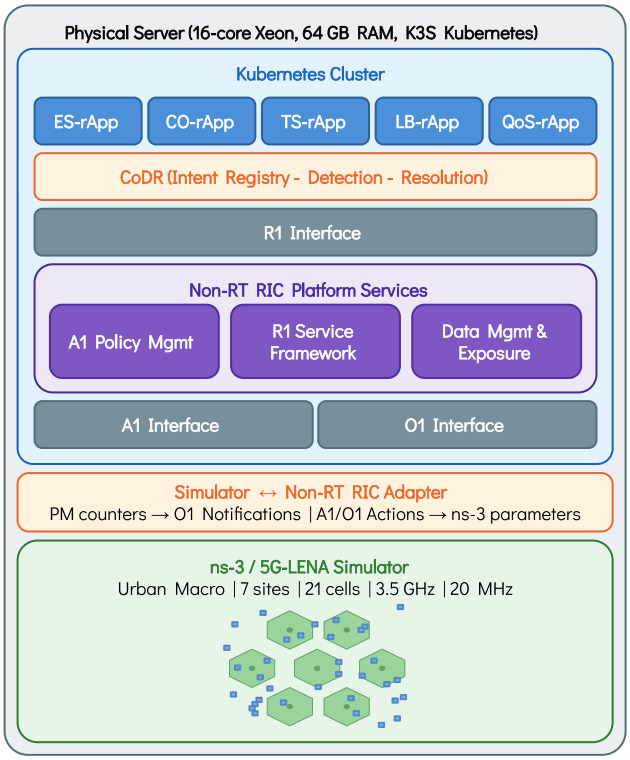

FIGURE 4. Testbed architecture. The ns-3/5G-LENA simulator provides a multi-cell RAN environment. An adapter module bridges the simulator to the OSC Non-RT RIC, translating PM data and configuration actions. The rApps and CoDR run as containerized workloads on the Kubernetes cluster.

1) SIMULATED RAN

The radio access network is simulated using ns-3 (version 3.44) with the 5G-LENA NR module [7], [36]. The simulated scenario models an urban macro deployment consisting of 7 tri-sectorized sites arranged in a hexagonal layout, yielding 21 cells. The inter-site distance is 500 m, consistent with the 3GPP Urban Macro calibration parameters [37]. Each cell operates on a 20 MHz carrier at 3.5 GHz with a numerology of $\mu = 1$ (30 kHz subcarrier spacing), providing 51 physical resource blocks per slot. User equipment (UE) instances are distributed uniformly across the coverage area, with the total number of active UEs varied between 210 and 630 (10–30 UEs per cell on average) depending on the experiment. UE mobility follows the random waypoint model with speeds between 1 and 5 m/s.

The traffic model combines three profiles: full-buffer downlink for throughput-centric evaluations, constant bit rate (CBR) at 2 Mbps for QoS-sensitive flows, and Poisson-arrival bursty traffic for background load. The simulator exposes performance measurement (PM) counters (per-cell PRB utilization, per-UE throughput, RSRP and RSRQ measurements, handover attempt and failure counts, and energy consumption estimates) at a configurable reporting interval. Unless otherwise stated, the PM reporting period is 1 s for RAN-level counters and 5 s for per-UE measurements.

2) NON-RT RIC PLATFORM

The Non-RT RIC is realized using the O-RAN Software Community (OSC) platform L release [10]. The platform runs

on a Kubernetes cluster deployed on a single physical server equipped with a 16-core Intel Xeon processor, 64 GB of RAM, and a 1 TB NVMe SSD. The Kubernetes distribution is K3s [38], chosen for its lightweight footprint and suitability for research testbeds. The OSC Non-RT RIC components deployed include the A1 Policy Management Service, the R1 service framework, and the data management and exposure services. An adapter module bridges the ns-3 simulation and the Non-RT RIC: it translates PM counters from the simulator into O1-compliant notifications consumed by the rApps, and converts A1 policies and O1 configuration changes issued by the rApps into simulation parameter updates applied at the next PM reporting boundary. This closed-loop adapter enables the rApps to operate as if they were connected to a live RAN, preserving the realistic timing and data flow of the O-RAN architecture while using a fully reproducible simulated environment.

B. rApp PORTFOLIO

We implement five rApps, each encapsulating a distinct RAN management function. The rApps are intentionally kept simple, rule-based or threshold-driven, so that the experimental focus remains on the conflicts among them rather than on the sophistication of any individual rApp's decision logic. Each rApp is packaged as a Docker container and deployed on the Kubernetes cluster alongside the Non-RT RIC services. Table 5 summarizes the portfolio.

1) ES-rApp (ENERGY SAVING)

The ES-rApp monitors per-cell PRB utilization through the O1 PM interface. When the average PRB utilization of a cell falls below a low-load threshold λ_{low} (default: 15%) over three consecutive reporting periods, the rApp issues an O1 CM operation to modify the cell's administrative state, transitioning it to a deactivated mode (or, in the ns-3 model, reducing its transmit power to a minimum value that effectively removes it from the coverage map). This is a heavyweight configuration action that persists until the ES-rApp's next decision cycle (30 s), in contrast to xApp-level power control via E2 that operates on a per-slot basis. When the traffic in the surrounding cells increases above a reactivation threshold λ_{high} (default: 70%), the ES-rApp reactivates the dormant cell. The optimization objective is to minimize total network energy consumption subject to maintaining acceptable coverage.

2) CO-rApp (COVERAGE OPTIMIZATION)

The CO-rApp consumes per-UE RSRP and RSRQ reports aggregated at the cell level. It maintains a running estimate of the coverage probability, defined as the fraction of UEs whose serving-cell RSRP exceeds a minimum threshold ρ_{min} (default: -110 dBm). When the coverage probability of a cell drops below a target P_{cov} (default: 95%), the rApp issues O1 CM operations to adjust the electrical tilt of the ManagedElement and the CIO of the affected cell's neighbour relations in the CellRelation managed objects. These adjustments are computed from coverage statistics aggregated over the 10 s reporting period and persist as static configuration until the next decision cycle, unlike xApp-level beam management via E2 that adapts on per-measurement-report time scales. Tilt adjustments are bounded to $\pm 2°$ per decision cycle, and CIO adjustments to ± 3 dB, to prevent large transient disturbances.

3) TS-rApp (TRAFFIC STEERING)

The TS-rApp aims to improve 5th-percentile user throughput by steering cell-edge users toward cells offering better radio conditions. It periodically evaluates, for each UE with throughput below a target T_{min} (default: 5 Mbps), whether a neighbouring cell would offer a higher estimated throughput based on the latest RSRP reports. If a beneficial target cell is identified, the rApp issues an O1 CM operation to adjust the CIO of the corresponding neighbour relation in the CellRelation managed object, biasing the handover decision toward that cell. Unlike an xApp performing real-time traffic steering via E2, the TS-rApp operates on aggregated RSRP statistics over a 10 s window and produces a persistent configuration change that remains in effect until the next decision cycle. The CIO adjustment magnitude is proportional to the RSRP difference between the serving and target cells, capped at ± 6 dB.

4) LB-rApp (LOAD BALANCING)

The LB-rApp monitors per-cell PRB utilization and aims to equalize the load distribution across the network. When the PRB utilization of a cell exceeds a high-load threshold μ_{high} (default: 80%), the rApp identifies the least-loaded neighbouring cell and issues an O1 CM update to the CIO of the corresponding CellRelation managed object, encouraging handovers away from the congested cell. The LB-rApp operates on PRB utilisation statistics averaged over its 10 s decision cycle, producing a persistent configuration change rather than the per-UE, per-slot handover control that an xApp would exercise via E2. Conversely, when a cell is underloaded (below μ_{low}, default: 30%), the rApp adjusts CIOs to attract users from more heavily loaded neighbours. CIO adjustments are bounded to ± 4 dB per cycle.

5) QoS-rApp (QoS ENFORCEMENT)

The QoS-rApp issues A1 policy directives to enforce minimum throughput guarantees for a designated high-priority traffic class. It monitors the per-UE throughput of users belonging to the high-priority slice and, when the aggregate satisfaction ratio (fraction of users meeting the throughput target) drops below a threshold S_{min} (default: 90%), issues an A1 policy intent specifying a minimum throughput target for the designated slice, delegating the enforcement mechanism to the Near-RT RIC (simulated in our testbed as a policy-to-configuration translator that maps the intent to a PRB reservation increase). The QoS-rApp operates strictly through A1 policy guidance on a 15 s cycle; it does not perform direct resource allocation, which remains the

TABLE 5. Summary of the rApp portfolio.

rApp	Objective	Action space	Interface	Cycle (s)
ES-rApp	Energy saving	Cell on/off, Tx power	O1	30
CO-rApp	Coverage opt.	Tilt, CIO	O1	10
TS-rApp	Traffic steering	CIO	O1	10
LB-rApp	Load balancing	CIO	O1	10
QoS-rApp	QoS enforcement	PRB reservation	A1	15

TABLE 6. Conflict-prone rApp pairs and associated taxonomy categories. D = Direct parameter, R = Resource contention, O = Objective interference, T = Temporal.

rApp pair	D	R	O	T
TS-rApp vs. LB-rApp	✓		✓	✓
ES-rApp vs. CO-rApp			✓	✓
ES-rApp vs. QoS-rApp		✓	✓	
CO-rApp vs. TS-rApp	✓			✓
LB-rApp vs. QoS-rApp		✓	✓	
CO-rApp vs. LB-rApp	✓			

responsibility of the Near-RT RIC and its xApps acting on sub-second time scales via E2. The reservation increment is 5% of the cell's total PRBs per decision cycle, up to a maximum reservation of 40%.

C. CONFLICT PAIRS

Table 6 identifies the conflict-prone rApp pairs within the portfolio and maps each pair to the taxonomy categories introduced in Section III. Several pairs exhibit conflicts in more than one category simultaneously. For instance, the TS-rApp and LB-rApp pair produces both direct parameter conflicts (both adjust the CIO of the same neighbour relation) and temporal conflicts (near-simultaneous CIO changes cause handover oscillation). The ES-rApp and CO-rApp pair produces objective interference (coverage degradation from cell deactivation) and, under rapid toggling of cell state, temporal conflicts as well. These multi-category conflict pairs serve as particularly informative test cases in the experimental evaluation.

D. CoDR PROTOTYPE

The CoDR framework is implemented as a Python-based microservice running in a dedicated container on the Kubernetes cluster. It consists of three main components corresponding to the design presented in Section IV.

1) INTENT REGISTRY

The intent registry is backed by a Redis instance [39] that provides fast key-value storage with support for time-to-live (TTL) expiration, which we use to implement the configurable retention policy. Each intent record is stored as a JSON document keyed by a composite identifier comprising the rApp ID, the target managed element, and a monotonically increasing sequence number. Secondary indices on the managed-element ID and the parameter name enable the indexed lookups required by the direct parameter conflict detector. The registry exposes a REST API through which the detection engine queries recent and pending intents.

TABLE 7. Default CoDR detection parameters.

Parameter	Description	Default
ϵ	Direct parameter tolerance	0.5 dB
θ_{RAN}	PRB contention threshold	85%
θ_{platform}	Platform resource threshold	80%
w	Temporal window length	60 s
τ	Reconfiguration rate threshold	8 changes
δ	Objective interference obs. window	30 s
σ_k	KPI deviation sensitivity	2σ
γ	Correlation score threshold	1.5

2) DETECTION ENGINE

The detection engine is implemented as a set of four Python modules, one per detection stage, orchestrated by a pipeline controller. The pipeline controller is invoked each time a new intent arrives in the registry and executes the four stages sequentially, short-circuiting to the resolution engine as soon as a conflict is detected.

The direct parameter detector performs a Redis query for any recent intent matching the same (managed element, parameter) pair and compares values using the configurable tolerance ϵ. The resource contention detector maintains a sliding window of per-cell PRB utilization received from the PM adapter and compares the projected utilization against the threshold θ_{RAN}. Platform-side resource monitoring is implemented via the Kubernetes Metrics API, querying per-pod CPU and memory usage. The temporal conflict detector maintains per-cell reconfiguration counters in Redis with TTL-based expiration, implementing the sliding window of length w without requiring explicit cleanup. The objective interference detector subscribes to a KPI stream from the PM adapter and, upon each new rApp action, opens an observation window of length δ during which it monitors deviations from baseline KPI values. Baseline statistics (mean and standard deviation) are computed over a configurable warm-up period at the start of each experiment.

3) RESOLUTION ENGINE

The resolution engine receives conflict reports from the detection engine and applies the operator-configured strategy. The configuration is specified in a YAML policy file that maps each conflict category to a resolution strategy and, for the priority-based strategy, defines the rApp priority ordering. The engine supports all four strategies described in Section IV-D. The negotiation strategy is realized through a lightweight callback mechanism: CoDR publishes a constraint message to a per-rApp Redis channel, and rApps that

implement the negotiation interface subscribe to this channel and resubmit revised intents within a configurable timeout (default: 5 s). In our portfolio, the TS-rApp and LB-rApp implement the negotiation interface; the remaining rApps fall back to priority-based resolution when negotiation is configured.

E. DEFAULT CONFIGURATION PARAMETERS

Table 7 lists the default values of the CoDR detection parameters used across all experiments unless otherwise stated. Section VI includes a dedicated sensitivity analysis (Experiment 5) in which the objective interference parameters σ_k, γ, and δ are varied systematically.

Each experiment runs for a simulated duration of 600 s (10 minutes) after a 120 s warm-up period during which baseline KPI statistics are collected and rApps are allowed to reach initial steady-state behavior. Results are averaged over 5 independent simulation runs with different random seeds to account for stochastic variability in UE placement and mobility.

The choice of a 600 s simulation (480 s post-warmup) yields between 16 and 48 decision cycles per rApp, depending on the cycle period (30 s for ES-rApp, 10 s for TS/LB/CO-rApps). To verify that this duration is sufficient for the system to reach steady state, we ran extended 3600 s simulations for the two most critical configurations, all five rApps without CoDR and CoDR with the hybrid strategy, and tracked the cumulative running mean of each KPI over time. For both configurations, the running means of all five KPIs enter and remain within $\pm 5\%$ of their final (3600 s) values within at most 140 s after the warm-up period, well within the 480 s observation window used in the experiments. At $t = 600$ s, the running means deviate from the 3600 s reference by less than 1.4% across all KPIs and configurations, confirming that the 600 s duration captures the steady-state behavior of the system.

The rapid convergence is attributable to two factors. First, the rApps are rule-based and react to threshold crossings, so their decision dynamics converge within a small number of decision cycles once the PM data stream stabilises. Second, the conflict patterns are quasi-periodic (driven by the rApp cycle periods and the oscillatory load dynamics), and the CoDR detection engine accumulates sufficient statistics within its observation window $\delta = 30$ s. A system employing slower-adapting ML-based rApps with longer learning horizons would require a correspondingly longer simulation, but this is outside the scope of the present evaluation.

F. THRESHOLD SPECIFICATION AND CALIBRATION

This section states the decision predicate in which each threshold appears, its default value, and the procedure by which that default is obtained. Thresholds fall into two classes: those *derived* from a property of the system that requires no tuning, and those *calibrated* from measurements taken during the warm-up period. Table 8 summarises both.

1) ϵ—DIRECT PARAMETER TOLERANCE

Two intents assigning values v_i, v_j to parameter p of element m are contradictory when

$$|v_i - v_j| > \epsilon_p. \tag{9}$$

ϵ_p is *derived*, not tuned: it is set to the quantisation step of p in the O-RAN information model, since two values differing by less than one quantisation step map to the same applied configuration and cannot be in conflict. For CIO, whose `q-OffsetRange` is specified in 0.5 dB increments over the range of interest, $\epsilon_{\text{CIO}} = 0.5$ dB. For electrical tilt, $\epsilon_{\text{tilt}} = 0.1°$. For PRB reservation, $\epsilon_{\text{PRB}} = 1$ PRB. No calibration is required, and the value transfers unchanged across deployments.

2) θ_{RAN}—PRB CONTENTION THRESHOLD

An incoming load-increasing intent on cell m is flagged when

$$u_m + \widehat{\Delta u}(a) > \theta_{\text{RAN}}, \tag{10}$$

where u_m is current PRB utilisation and $\widehat{\Delta u}(a)$ is the historical mean utilisation change for actions of the same type. θ_{RAN} is *calibrated* to the knee of the utilisation–delay curve: during warm-up, mean packet delay is recorded against binned PRB utilisation, and θ_{RAN} is set to the utilisation at which the fitted delay curve reaches twice its low-utilisation asymptote. In our deployment this yields $\theta_{\text{RAN}} = 85\%$. Operators with different traffic mixes should repeat this fit; the value is scenario-dependent.

3) θ_{PLATFORM}—PLATFORM RESOURCE THRESHOLD

Aggregate platform contention is flagged when $\max(c_{\text{agg}}/c_{\text{lim}},\ \mu_{\text{agg}}/\mu_{\text{lim}}) > \theta_{\text{platform}}$, where $c_{\text{lim}}, \mu_{\text{lim}}$ are the Kubernetes CPU and memory limits for the rApp namespace. $\theta_{\text{platform}} = 80\%$ follows standard Kubernetes practice of reserving 20% headroom for burst absorption before the limit triggers throttling or eviction. It is *derived* from the cluster configuration rather than tuned.

4) W, τ—TEMPORAL WINDOW AND RECONFIGURATION RATE

A temporal conflict is flagged on element m when

$$c_m(w) > \tau \quad \wedge \quad |\mathcal{S}_m(w)| \geq 2, \tag{11}$$

where $c_m(w)$ counts configuration changes applied to m in the trailing window of length w and $\mathcal{S}_m(w)$ is the set of distinct rApps contributing them.

w is *derived*: it is set to twice the longest rApp decision cycle, $w = 2 \max_i c_i$, so that at least two decision cycles of every rApp fall within the window and oscillation between any pair is observable. With $\max_i c_i = 30$ s (ES-rApp), $w = 60$ s.

τ is *calibrated* against the reconfiguration rate expected under non-conflicting operation. Modelling per-element

reconfiguration arrivals as Poisson, the expected count is

$$\lambda_m(w) = \sum_{i=1}^{n} \frac{w}{c_i} \cdot \frac{s_i}{|M|}, \tag{12}$$

with c_i the decision cycle of rApp r_i, s_i its mean number of targeted elements per cycle, and $|M|$ the number of managed elements. τ is then the smallest integer satisfying $\Pr[X > \tau] \leq \alpha$ for $X \sim \text{Poisson}(\lambda_m(w))$, where α is the operator's tolerated per-window false alarm rate. For the portfolio of Table 5, (12) gives $\lambda_m(60) = 4.5$, and $\alpha = 0.05$ yields $\tau = 8$. We note that this procedure supersedes the value $\tau = 5$ used in earlier revisions of this work, which corresponds to $\alpha = 0.30$ and is too permissive; results reported in Section VI use $\tau = 8$.

5) δ–OBSERVATION WINDOW

δ bounds the interval over which a KPI is attributed to a preceding action. It is *calibrated* to the RAN settling time: during warm-up, a set of probe reconfigurations is applied and the time for the affected KPIs to return to within one standard deviation of their pre-change mean is recorded. δ is set to the 90th percentile of these settling times, giving $\delta = 30$ s in our deployment. Experiment 5 independently confirms that $\delta \in [25, 50]$ s is the region in which both precision and recall exceed 80%.

6) σ_K, γ–DEVIATION SCREENING AND CONFIRMATION

These two thresholds apply to *different statistics* and are evaluated in sequence. Let $k_1, \ldots, k_N$ be the individual PM samples of KPI k within $[t, t + \delta]$, and let $(\bar{k}, \sigma_{\bar{k}})$ be the baseline mean and standard deviation for that KPI on that element. The screening test fires when any single sample departs from baseline,

$$\max_{1 \leq j \leq N} |k_j - \bar{k}| > \sigma_k \cdot \sigma_{\bar{k}}, \tag{13}$$

and only then is the confirmation test evaluated on the window mean,

$$\rho = \frac{\left| \frac{1}{N} \sum_{j=1}^{N} k_j - \bar{k} \right|}{\sigma_{\bar{k}}} > \gamma. \tag{14}$$

The distinction matters: (13) is a cheap per-sample trigger that a single transient spike will pass, whereas (14) tests whether the departure is sustained across the window. A transient therefore passes screening but fails confirmation, which is the mechanism by which the pair suppresses short-lived exogenous fluctuations. Because (13) bounds a maximum and (14) bounds a mean, neither condition implies the other for $N > 1$.

$\sigma_k = 2$ is *derived*: under a Gaussian null with no conflict, a two-sided 2σ band gives a per-sample false trigger probability of 4.6%, the conventional screening level. $\gamma = 1.5$ is *calibrated* by maximising the F_1 score over the sweep of Fig. 11, which attains 86.0 at $\gamma = 1.5$ against 85.0 at $\gamma = 1.25$ and 83.6 at $\gamma = 1.75$; this coincides with the precision–recall crossover and with the maximum of Youden's J.

7) CALIBRATION PROCEDURE

All calibrated thresholds are obtained during the 120 s warm-up, before any rApp is permitted to act:

1) Collect PM data for all elements; compute per-element, per-KPI baseline means and standard deviations $(\bar{k}, \sigma_{\bar{k}})$.
2) Fit the utilisation–delay curve and locate its knee to set θ_{RAN}.
3) Apply probe reconfigurations and record settling times to set δ.
4) Evaluate (12) from the registered rApp cycles and target counts to set τ.

Derived thresholds (ϵ_p, θ_{platform}, w, σ_k) require no measurement and are read from the information model, cluster configuration, rApp registration data, and the Gaussian screening level respectively. Baselines are refreshed on a rolling basis thereafter; the remaining thresholds are recalibrated only when the rApp portfolio changes.

VI. EXPERIMENTAL EVALUATION

This section presents the experimental evaluation of the CoDR framework. We first define the conflict scenarios and the metrics used throughout the experiments, then we describe and report results for five experiments that progressively characterize the impact of unmanaged conflicts, the effectiveness of CoDR's detection and resolution mechanisms, the comparative performance of resolution strategies, the scalability of the framework, and the sensitivity of the objective interference detector to its configuration parameters.

A. CONFLICT SCENARIOS

We define eight conflict scenarios that collectively exercise all four taxonomy categories and all six conflict-prone rApp pairs identified in Table 6. Each scenario specifies the active rApp pair, the expected conflict type, and the RAN conditions under which the conflict is triggered. Table 9 summarizes the scenarios.

Scenarios S1–S2 target direct parameter conflicts, S3–S4 target resource contention, S5–S6 target objective interference, and S7–S8 target temporal conflicts. Scenarios S5 and S7 involve the same rApp pair (ES vs. CO) under different load dynamics, illustrating that a single pair can produce different conflict types depending on the operating conditions. Similarly, scenarios S1, S6, and S8 involve the TS vs. LB pair across three conflict categories.

Absent publicly available implementations of prior conflict management frameworks for cellular networks, we structure the evaluation around three internal baselines spanning the design space. The *no-CoDR* configuration (all rApps active, no conflict management) sets the lower bound: performance an operator would observe if rApps were

TABLE 8. CoDR threshold specification. ***Derived*** thresholds follow from a system property and need no measurement; ***calibrated*** thresholds are fitted during warm-up.

Sym.	Predicate	Default	Stage	Class	Origin of the default
ϵ_p	$\lvert v_i - v_j \rvert > \epsilon_p$	0.5 dB (CIO); 0.1° (tilt); 1 PRB	1	Derived	Quantisation step of p in the O-RAN information model
θ_{RAN}	$u_m + \widehat{\Delta u} > \theta_{\text{RAN}}$	85%	2	Calibrated	Knee of the fitted utilisation–delay curve
θ_{plat}	$\max(c, \mu)/\text{limit} > \theta_{\text{plat}}$	80%	2	Derived	Kubernetes burst headroom convention
w	window length of $c_m(w)$	60 s	3	Derived	$2 \max_i c_i$, twice the longest rApp cycle
τ	$c_m(w) > \tau \wedge \lvert \mathcal{S}_m \rvert \geq 2$	8 changes	3	Calibrated	Poisson quantile at $\alpha = 0.05$, $\lambda = 4.5$
δ	observation window $[t, t{+}\delta]$	30 s	4	Calibrated	90th percentile of probe settling times
σ_k	$\max_j \lvert k_j - \bar{k} \rvert > \sigma_k \sigma_{\bar{k}}$	2	4	Derived	Two-sided Gaussian screening level (4.6%)
γ	$\rho > \gamma$ on the window mean	1.5	4	Calibrated	F_1 maximum over the sweep of Fig. 11

TABLE 9. Conflict scenarios used in the experimental evaluation.

ID	rApp pair	Category	Trigger condition	Expected conflict manifestation
S1	TS vs. LB	Direct param.	Mixed load (some cells at 80%+)	Both adjust CIO of same neighbour relation in opposite directions
S2	CO vs. LB	Direct param.	Coverage hole near congested cell	Both adjust CIO of same relation for different objectives
S3	ES vs. QoS	Resource cont.	Low-load period with active QoS slice	Cell deactivation reduces PRB pool available for QoS reservations
S4	LB vs. QoS	Resource cont.	High-load cell with active QoS slice	Load balancing redistributes users, violating PRB reservation
S5	ES vs. CO	Obj. interf.	Nighttime low-load scenario	Cell switch-off creates coverage gap detected by CO-rApp
S6	TS vs. LB	Obj. interf.	Uneven load with cell-edge users	Traffic steering increases load on target cell, triggering LB response
S7	ES vs. CO	Temporal	Fluctuating load near ES threshold	Rapid cell on/off toggling triggers repeated CO-rApp adjustments
S8	TS vs. LB	Temporal	Moderate load, short decision cycles	Near-simultaneous CIO adjustments cause handover oscillation

deployed without any coordination mechanism. The *priority-based* configuration is functionally equivalent to a simplified SON Coordination Function that resolves conflicts through static function priority, the most widely adopted mechanism in the 3GPP SON coordination literature [3], adapted to the rApp setting by replacing the SON coordination matrix with the CoDR intent registry. The *single-rApp* configurations provide per-objective upper bounds, capturing the best achievable performance for each optimization goal absent inter-rApp interference. The negotiation, temporal multiplexing, and hybrid strategies are then evaluated against these baselines. A direct experimental comparison against prior SON coordination frameworks (e.g., SOCRATES [13]) was not feasible: these systems target a different architectural context, their implementations are not publicly available, and adapting them to the O-RAN interface model would itself constitute a research contribution.

B. METRICS

We evaluate performance along two dimensions: RAN-level KPIs that measure the impact of conflicts on network performance, and conflict management metrics that measure the effectiveness of CoDR itself.

We consider the following RAN-Level KPIs:

- **Average cell throughput** ($\bar{T}_{\text{cell}}$): the mean downlink throughput per cell, averaged over all cells and all reporting periods after the warm-up phase.
- **5th-percentile user throughput** (T_5): the throughput value below which 5% of users fall, serving as a proxy for cell-edge performance.
- **Energy consumption index** (E): the aggregate energy consumption of the RAN, normalized to the energy consumption of the baseline scenario in which all cells are always active.
- **Handover failure rate** (HFR): the ratio of failed handover attempts to total handover attempts.
- **Ping-pong rate** (PPR): the fraction of successful handovers in which the UE returns to the source cell within 5 s, indicating unnecessary handover oscillation.
- **Coverage probability** (P_{cov}): the fraction of UEs whose serving-cell RSRP exceeds -110 dBm.

The Conflict Management Metrics considered in this study are the following:

- **Detection precision**: the fraction of detected conflicts that are true conflicts (i.e., that correspond to a ground-truth conflict label assigned by manual inspection of the action traces).

- **Detection recall**: the fraction of ground-truth conflicts that are correctly detected by CoDR.
- **Detection latency** (d_{lat}): the time elapsed between the arrival of a conflicting intent in the registry and the generation of a conflict alert by the detection engine.
- **Resolution overhead** (r_{oh}): the additional delay introduced by the resolution engine before an approved intent is forwarded to the RAN, measured only for intents that are involved in a detected conflict.
- **False positive rate** (FPR): the fraction of non-conflicting intents that are incorrectly flagged as conflicting.

The interpretation of d_{lat} differs between the two detection classes introduced in Section IV-C. For preventively detected conflicts, i.e., direct parameter, resource contention, and temporal, it measures the delay added to the action path before the intent is forwarded, and the RAN is never exposed to the conflict. For retrospectively detected objective interference conflicts, the action has already been applied, so d_{lat} measures instead the interval during which the RAN operated under the conflicting configuration. The two quantities are not directly comparable, and the pooled median reported in Table 12 is dominated by the preventive stages; the retrospective contribution is visible only in the 95th percentile.

Ground-truth conflict labels are generated by an offline oracle that has full visibility of all rApp actions, the simulator state, and the resulting KPI trajectories. The oracle labels an intent pair as conflicting if their combined execution produces a statistically significant KPI degradation (measured by a paired t-test at $p < 0.05$) compared to the execution of either intent in isolation.

The oracle-based labeling approach has two limitations that should be considered when interpreting the precision and recall figures. First, the paired t-test with $n = 5$ has limited statistical power: for a two-sided test at $\alpha = 0.05$ with $n = 5$, the minimum detectable effect size is approximately Cohen's $d = 1.7$ at 80% power. This means that genuine conflicts whose KPI impact is small relative to inter-run variability may fail to reach significance and will be labeled as non-conflicting by the oracle. The resulting bias is directional: the ground-truth positive set may be incomplete (missing low-severity conflicts), which would cause the reported recall to underestimate CoDR's true recall while leaving precision largely unaffected. Second, because the oracle itself is a statistical classifier rather than an infallible authority, its labels carry residual uncertainty that propagates into the precision and recall metrics. We note, however, that this uncertainty is bounded: at the $p < 0.05$ threshold, the probability of any single label being a false positive (a non-conflicting pair labeled as conflicting) is at most 5%, and the expected number of such mislabeled pairs is small relative to the total ground-truth set. Alternative labeling strategies, manual expert annotation or rule-based detection limited to parameter overlap, were considered but rejected because the former does not scale to the thousands of intent pairs in the evaluation, and the latter cannot identify objective interference or resource contention conflicts, which are central to the contribution. Despite its imperfections, the t-test oracle provides automated, category-agnostic labels grounded in observable KPI impact, and its known bias direction (conservative recall, accurate precision) means that the reported detection performance represents a lower bound on CoDR's actual effectiveness.

C. VALIDATION OF THE LOAD-IMPACT ESTIMATOR

Stage 2 of the detection pipeline flags a resource contention conflict when the projected utilisation $u_m + \widehat{\Delta u}(a)$ exceeds θ_{RAN}. The projection depends on $\widehat{\Delta u}(a)$, an empirical estimate of the PRB utilisation change that action a will induce, obtained from a historical average conditioned on the action type and a coarse pre-action load bin. Because an inaccurate estimator would propagate directly into Stage 2's precision and recall, we validate it against ground truth before reporting the end-to-end results.

1) METHOD

For every load-increasing action executed across the Experiment 1–3 runs, we record the estimator's prediction at submission time and the PRB utilisation change actually observed in the simulator over the two PM reporting periods following the action's application. This yields 460 paired observations spanning five action types: CIO increases of 1–3 dB and 4–6 dB, cell reactivation, neighbour cell deactivation, and PRB reservation changes. Fig. 5 summarises the result.

2) REGRESSION ACCURACY

The estimator achieves a mean absolute error of 1.59 pp and an RMSE of 2.15 pp against actual utilisation changes spanning roughly -15 to $+20$ pp, with $R^2 = 0.77$. The bias is $+0.17$ pp, so the estimator is essentially unbiased in aggregate. Conditioning the historical average on a load bin is what makes this possible: a load-agnostic average systematically overestimates impact on heavily loaded cells, which have little headroom to absorb additional traffic, and we measured a bias of $+2.4$ pp for that variant during development.

Fig. 5(b) shows that residual error is not uniform. Stratifying by pre-action load gives an MAE of 0.92 pp below 40% utilisation, 1.48 pp between 40% and 70%, and 1.84 pp above 70%. The degradation is expected: the compression of an action's effect varies most steeply with utilisation in the high-load regime, so a bin-conditioned average captures less of the within-bin variation there. This is an unwelcome property, since the high-load regime is precisely where Stage 2 decisions are consequential.

3) THRESHOLD-CROSSING ACCURACY

Regression accuracy is not, however, the quantity that determines Stage 2's behaviour. The estimator's only role is to decide whether the projection crosses θ_{RAN}; an error

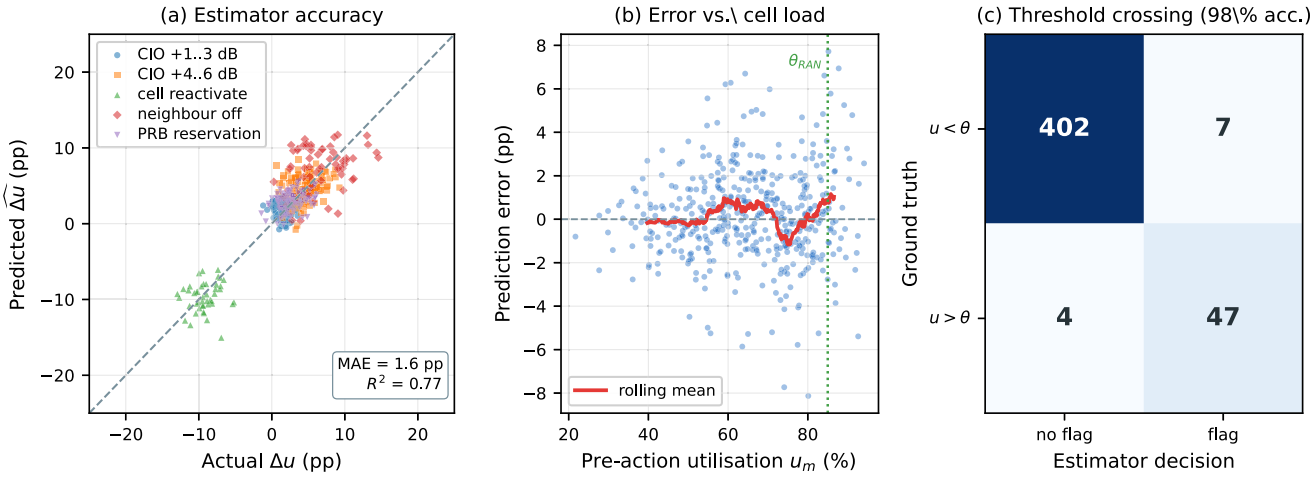

FIGURE 5. Validation of the load-impact estimator against observed post-action PRB utilisation ($N = 460$ actions). (a) Predicted versus actual utilisation change, by action type; the dashed line is the identity. (b) Prediction error against pre-action cell load, with a rolling mean; the dotted line marks θ_{RAN}. (c) Confusion matrix for the decision that actually matters, whether the projected utilisation crosses θ_{RAN}.

TABLE 10. RAN-level KPIs under different rApp configurations (Experiment 1). Values are means ± standard deviations over 5 runs.

Configuration	$\bar{T}_{\text{cell}}$ (Mbps)	T_5 (Mbps)	E	HFR (%)	PPR (%)	P_{cov} (%)
No rApp (static)	45.0 ± 1.2	3.5 ± 0.3	1.00	1.2 ± 0.2	2.0 ± 0.3	93.0 ± 0.8
ES-rApp only	42.3 ± 1.4	3.0 ± 0.4	0.68	1.5 ± 0.3	2.2 ± 0.4	89.1 ± 1.1
CO-rApp only	46.1 ± 1.1	4.2 ± 0.3	1.00	1.0 ± 0.2	1.8 ± 0.3	97.5 ± 0.5
TS-rApp only	47.2 ± 1.3	5.8 ± 0.5	1.00	1.8 ± 0.3	2.5 ± 0.4	93.5 ± 0.7
LB-rApp only	48.1 ± 1.0	4.5 ± 0.4	1.00	1.3 ± 0.2	2.1 ± 0.3	93.2 ± 0.6
QoS-rApp only	44.2 ± 1.2	4.0 ± 0.4	1.00	1.2 ± 0.2	2.0 ± 0.3	93.0 ± 0.7
All 5 rApps (no CoDR)	37.8 ± 2.6	2.1 ± 0.6	0.78	5.8 ± 0.9	8.5 ± 1.2	88.1 ± 1.8

of several percentage points is inconsequential when the projection is far from the threshold and consequential only when it is near it. Fig. 5(c) evaluates this directly. Across the 460 actions the estimator classifies the crossing correctly in 97.6% of cases, with 7 false positives (a conflict flagged when utilisation did not in fact cross) and 4 false negatives (a genuine crossing missed).

The asymmetry is favourable for a conflict detector: false positives cost an unnecessary resolution action, whereas false negatives allow a genuine contention conflict to reach the RAN. That the estimator errs toward the former is a consequence of the residual positive bias in the medium-load bin (+0.33 pp), which nudges borderline projections above the threshold. Operators wishing to eliminate false negatives entirely can subtract a safety margin from θ_{RAN} equal to the upper bound of the estimator's error at the operating load, roughly 4 pp at the 95th percentile in our deployment, at the cost of additional false positives.

4) LIMITATIONS

Two caveats apply. First, the validation is conducted against the ns-3/5G-LENA model rather than a live RAN, so it establishes that the estimator tracks the simulator's load dynamics, not that it would track a real network's. Second, the estimator is trained and evaluated on the same rApp portfolio; an rApp issuing an action type absent from the historical record would receive a default estimate drawn from the nearest matching type, and its accuracy is not characterised here. Both limitations argue for periodic recalibration of the historical table in a production deployment, which is inexpensive since the required data, action records and subsequent PM counters, is already collected by the intent registry and the O1 interface.

D. EXPERIMENT 1: IMPACT OF UNMANAGED CONFLICTS

The first experiment establishes the baseline by quantifying how much RAN performance degrades when multiple rApps operate concurrently without any conflict management. We run four configurations: (i) each rApp in isolation (five single-rApp runs), (ii) all five rApps concurrently without CoDR, (iii) pairwise combinations of conflicting rApp pairs without CoDR, and (iv) a no-rApp baseline in which the RAN operates with static default configuration. All configurations use the medium-load scenario (420 UEs, 20 per cell on average).

Table 10 reports the RAN-level KPIs for each configuration. When rApps operate in isolation, each achieves its optimization objective effectively: the ES-rApp reduces

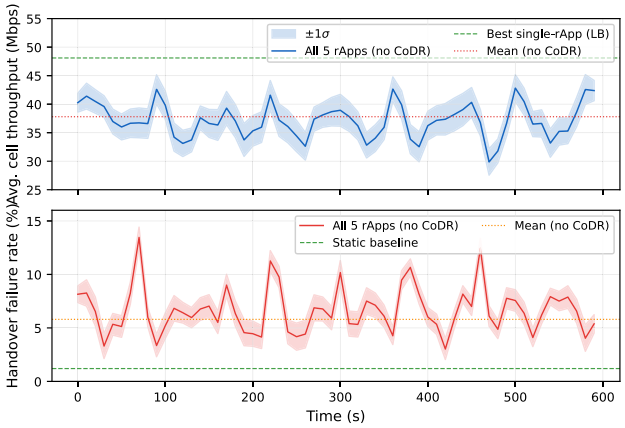

FIGURE 6. Time-series evolution of average cell throughput (top) and handover failure rate (bottom) under unmanaged concurrent rApp operation (Experiment 1). The shaded band indicates $\pm 1\sigma$ across runs.

energy consumption to 68% of the baseline, the CO-rApp increases coverage probability from 93.0% to 97.5%, and the TS-rApp improves 5th-percentile throughput from 3.5 Mbps to 5.8 Mbps. However, when all five rApps operate concurrently without CoDR, the aggregate performance degrades significantly.

The unmanaged concurrent deployment degrades average cell throughput by 21% relative to the best single-rApp result (48.1 Mbps with LB-rApp alone) and increases the handover failure rate by nearly 5× compared to the static baseline. The ping-pong rate rises to 8.5%, indicating severe handover oscillation attributable primarily to the TS-rApp and LB-rApp conflict. Coverage probability drops by 9.4 percentage points relative to the CO-rApp-only scenario, confirming that the ES-rApp's cell deactivation decisions undermine the CO-rApp's coverage objective. Notably, the energy consumption index of the unmanaged deployment (0.78) is worse than the ES-rApp-only result (0.68), because the repeated activation–deactivation cycles triggered by the ES–CO conflict prevent the ES-rApp from maintaining cells in the off state for extended periods.

Fig. 6 shows the time-series evolution of average cell throughput and handover failure rate for the all-rApps-no-CoDR configuration. The KPI traces exhibit pronounced oscillatory behavior, with throughput swings of up to 8 Mbps within successive 10 s reporting windows, confirming the destabilizing effect of uncoordinated rApp actions. The handover failure rate spikes periodically to values above 10%, coinciding with moments when the TS-rApp and LB-rApp issue contradictory CIO adjustments on the same neighbour relations.

E. EXPERIMENT 2: CoDR WITH PRIORITY-BASED RESOLUTION

The second experiment activates CoDR with the priority-based resolution strategy. The rApp priority ordering is set to QoS-rApp > CO-rApp > LB-rApp > TS-rApp > ES-rApp,

TABLE 11. RAN-level KPIs with CoDR priority-based resolution vs. unmanaged baseline (Experiment 2).

Metric	No CoDR	CoDR (priority)	Recovery
$\bar{T}_{\text{cell}}$ (Mbps)	37.8	45.5	75%
T_5 (Mbps)	2.1	4.0	51%
E	0.78	0.91	—
HFR (%)	5.8	2.1	80%
PPR (%)	8.5	3.0	85%
P_{cov} (%)	88.1	96.2	+8.1 pp

TABLE 12. Conflict detection performance with priority-based resolution (Experiment 2).

Metric	Value
Detection precision	92.6%
Detection recall	86.7%
Median detection latency	12 ms
95th-pctl detection latency	2.9 s
Median resolution overhead	4 ms
False positive rate	2.07%

reflecting a representative operator preference that prioritizes service quality and coverage over energy efficiency. All five rApps run concurrently under the same medium-load scenario as Experiment 1.

Table 11 compares the KPIs achieved by CoDR with priority-based resolution against the unmanaged baseline from Experiment 1 and the single-rApp references. The "Recovery" column reports the fraction of the KPI gap between the unmanaged case and the best single-rApp reference that CoDR closes.[1]

CoDR with priority-based resolution recovers 75% of the throughput loss observed in the unmanaged case and reduces the handover failure rate by 64%. Coverage probability is restored to 96.2%, within 1.3 percentage points of the CO-rApp-only reference. The recovery is not complete because priority-based blocking suppresses the lower-priority rApp's actions entirely rather than finding a compromise. The ES-rApp, ranked lowest in the priority ordering, is blocked in 73% of its action attempts and consequently achieves an energy reduction of only 9% ($E = 0.91$) compared to 32% ($E = 0.68$) when operating alone.

Table 12 reports the conflict management metrics.

The detection engine achieves a precision of 92.6% and a recall of 86.7% across all conflict categories. The median detection latency of 12 ms is dominated by the direct parameter and temporal detectors, which operate synchronously and return within single-digit milliseconds. The 95th-percentile latency of 2.9 s reflects the objective interference detector, which operates asynchronously and

[1] Recovery denotes the fraction of the KPI gap between the unmanaged case and the best single-rApp reference that CoDR closes, computed as (CoDR − No CoDR)/(Best single-rApp − No CoDR) for "higher is better" metrics and the inverse for "lower is better" metrics. For energy, the hybrid objective makes a single recovery figure misleading, so it is omitted. For coverage, the absolute gain in percentage points is reported instead.

TABLE 13. Comparison of resolution strategies (Experiment 3). All configurations run five rApps concurrently under medium load.

Strategy	$\bar{T}_{\text{cell}}$ (Mbps)	T_5 (Mbps)	E	HFR (%)	PPR (%)	P_{cov} (%)	Mean r_{oh} (ms)
No CoDR	37.8	2.1	0.78	5.8	8.5	88.1	—
Priority-based	45.5	4.0	0.91	2.1	3.0	96.2	4
Negotiation	46.8	4.5	0.79	1.8	2.6	96.5	304
Temporal mux	45.2	3.8	0.82	2.0	2.8	95.8	2
Hybrid	46.5	4.4	0.80	1.7	2.4	96.8	38

accumulates KPI evidence over the observation window $\delta = 30$ s before issuing an alert. Because this category accounts for roughly a quarter of all flags, its distribution dominates the aggregate above the 88th percentile; the aggregate median of 12 ms and the 95th percentile of 2.9 s therefore describe two disjoint modes rather than two points on one distribution. The false positive rate of 2.07% is attributable primarily to the objective interference detector, which occasionally flags exogenous KPI fluctuations, caused by UE mobility or bursty traffic arrivals, as rApp-induced interference. The resolution overhead is negligible: the median of 4 ms confirms that priority comparison and intent status update add no meaningful delay to the action path.

F. EXPERIMENT 3: COMPARISON OF RESOLUTION STRATEGIES

The third experiment compares the four resolution strategies described in Section IV-D. We run four configurations (priority-based, negotiation-based, temporal multiplexing, and a hybrid policy that assigns the most appropriate strategy per conflict category) all under the same medium-load scenario with all five rApps active. For temporal multiplexing, the ES-rApp is active during a simulated nighttime window (the first 300 s of the experiment, representing low-load hours) and the CO-rApp during the remaining 300 s. For the hybrid policy, the mapping is: direct parameter conflicts → priority-based, resource contention → negotiation, objective interference → temporal multiplexing, temporal conflicts → priority-based.

Table 13 reports the KPIs and key conflict management metrics for each strategy.

The negotiation strategy achieves the best aggregate KPI performance, recovering 87% of the throughput loss while simultaneously preserving 66% of the ES-rApp's energy savings ($E = 0.79$ vs. $E = 0.68$ for ES-rApp alone), a substantial improvement over priority-based blocking, which largely suppresses the ES-rApp ($E = 0.91$). This improvement comes at the cost of higher resolution overhead: the median resolution latency increases from 4 ms (priority-based) to 304 ms (negotiation), reflecting the round-trip time for constraint notification and intent revision. The negotiation benefit is most pronounced for the TS vs. LB pair, where both rApps implement the constraint-handling interface and can converge on a mutually acceptable CIO adjustment that steers cell-edge users without overloading the target cell.

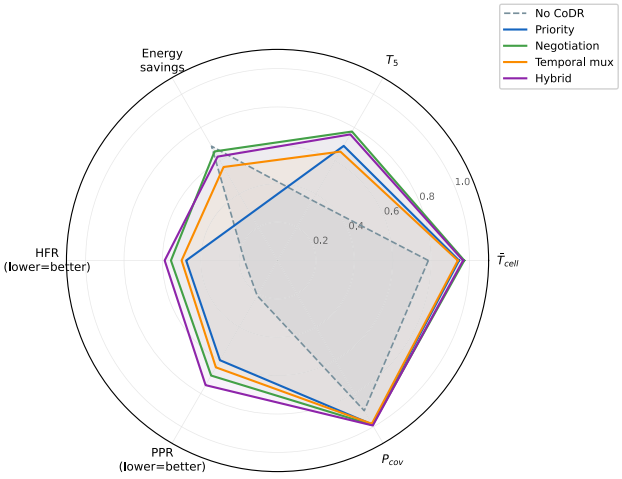

FIGURE 7. Radar chart comparing resolution strategies across six KPI dimensions (Experiment 3). Each axis is normalized to the best single-rApp value for that KPI. Larger area indicates better aggregate performance.

Temporal multiplexing effectively eliminates the ES vs. CO conflict by separating the two rApps in time, achieving an energy index of 0.82, better than priority-based blocking but worse than negotiation, because the ES-rApp is confined to operating only during the first 300 s. Its resolution overhead is the lowest of all strategies (2 ms) because actions are simply gated by a time-window check. The throughput and coverage results are slightly below the other CoDR strategies because the CO-rApp cannot compensate for coverage gaps during the ES-rApp's active window.

The hybrid strategy achieves performance close to negotiation across throughput and coverage metrics ($\bar{T}_{\text{cell}} = 46.5$ Mbps, $P_{\text{cov}} = 96.8\%$) while maintaining substantially lower mean resolution overhead (38 ms vs. 304 ms). This is because the hybrid approach applies the lightweight priority-based strategy to direct parameter and temporal conflicts, which account for approximately 65% of all detected conflicts, and reserves negotiation and temporal multiplexing for the conflict types where they yield the greatest benefit. The hybrid strategy also achieves the lowest handover failure rate (1.7%) and ping-pong rate (2.4%) of all configurations, approaching the best single-rApp values.

TABLE 14. CoDR overhead as a function of the number of concurrent rApps (Experiment 4).

rApps	Conflicts detected	Med. d_{lat} (ms)	CPU (%)	Memory (MB)
2	8	8	3.2	45
3	19	10	4.8	62
4	34	11	6.5	78
5	52	12	8.1	95

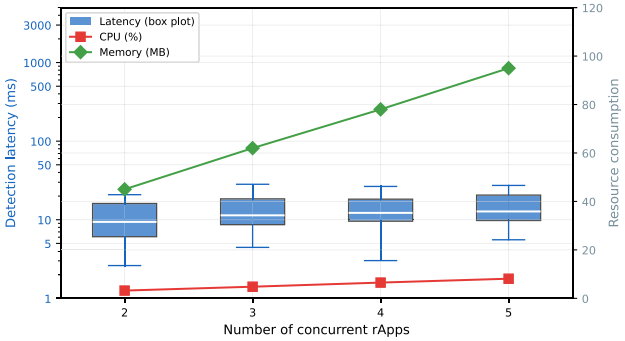

FIGURE 8. CoDR detection latency (box plot, left axis) and resource consumption (right axis) as a function of the number of concurrent rApps (Experiment 4).

Fig. 7 presents a radar chart comparing the four strategies across the six KPI dimensions, normalized to the single-rApp-best reference for each KPI.

G. EXPERIMENT 4: SCALABILITY

The fourth experiment evaluates how CoDR's detection and resolution overhead scales with the number of concurrent rApps. We run configurations with 2, 3, 4, and 5 rApps active, adding rApps in the order TS, LB, CO, ES, QoS (chosen to maximize the number of active conflict pairs at each step). CoDR uses the hybrid resolution strategy. We measure detection latency, resolution overhead, and the CPU and memory consumption of the CoDR container.

Table 14 reports the results. Fig. 8 plots the detection latency distribution and CoDR resource consumption as a function of the number of active rApps.

The number of detected conflicts grows from 8 with 2 rApps to 52 with 5 rApps, roughly tracking the quadratic growth in the number of active conflict pairs. Despite this increase, the median detection latency remains between 8 and 12 ms across all configurations, confirming that the synchronous stages of the detection pipeline (direct parameter, resource contention, and temporal) scale well with the intent volume. The 95th-percentile latency increases from 0.9 s with 2 rApps to 2.1 s with 5 rApps, driven primarily by the objective interference detector's observation window rather than by computational cost. CPU consumption of the CoDR container grows from 3.2% to 8.1% of a single core, and memory consumption from 45 MB to 95 MB, both remaining well within the capacity of the testbed server. These results

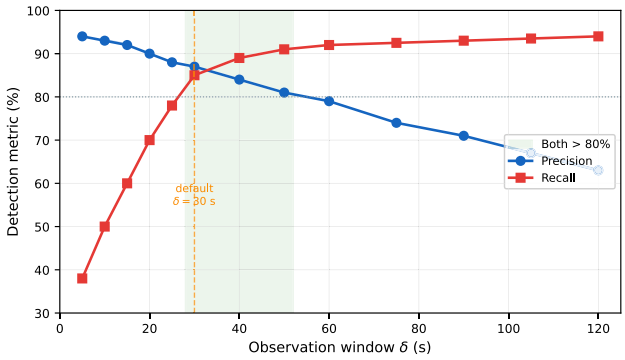

FIGURE 9. Detection precision and recall as a function of the observation window length δ (Experiment 5). The shaded region indicates the operating range that achieves both precision and recall above 80%.

indicate that CoDR's overhead remains modest at the tested scale of five concurrent rApps. We note, however, that the number of potential conflict pairs grows quadratically with the number of rApps, and the objective interference detector's cost depends on the number of active pairs rather than on the registry size alone. While the synchronous detection stages (direct parameter, resource contention, temporal) are based on indexed lookups that scale efficiently, the asynchronous observation-window mechanism may exhibit non-linear cost growth beyond the range tested here. Establishing the framework's scaling limits at 10–20 rApps remains an open question that would require a larger rApp portfolio with well-characterised conflict semantics.

H. EXPERIMENT 5: SENSITIVITY OF OBJECTIVE INTERFERENCE DETECTION

The fifth experiment explores the sensitivity of the objective interference detector to its three key parameters: the observation window length δ, the KPI deviation sensitivity σ_k, and the correlation score threshold γ. We use the ES vs. CO pair (Scenario S5) as the test case because it produces well-characterized objective interference conflicts. We vary one parameter at a time while holding the other two at their default values (Table 7).

1) SENSITIVITY TO δ (OBSERVATION WINDOW LENGTH)

Fig. 9 shows detection precision and recall as a function of δ varied from 5 s to 120 s.

At short window lengths ($\delta < 10$ s), recall is low (38%) because the KPI impact of cell deactivation has not fully materialized within the observation window. Precision is high at short windows (94%) because the few detected correlations tend to be genuine. As δ increases, recall improves steadily, reaching 85% at $\delta = 30$ s and plateauing near 93% for $\delta \geq 90$ s. Precision remains above 87% up to $\delta = 30$ s but degrades at longer windows, dropping to 79% at $\delta = 60$ s and 63% at $\delta = 120$ s, because exogenous KPI variations become more likely within a wider observation interval. The default value of $\delta = 30$ s falls within the operating range that

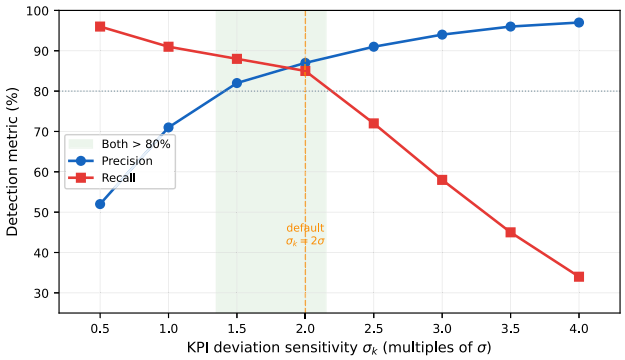

FIGURE 10. Detection precision and recall as a function of the KPI deviation sensitivity σ_k (Experiment 5).

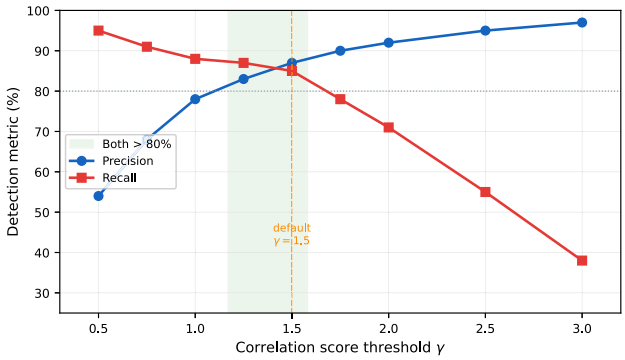

FIGURE 11. Detection precision and recall as a function of the correlation score threshold γ (Experiment 5).

achieves both precision and recall above 80%, which spans approximately $\delta \in [25, 50]$ s.

2) SENSITIVITY TO σ_K (KPI DEVIATION SENSITIVITY)

Fig. 10 shows the precision–recall trade-off as σ_k is varied from 0.5σ to 4σ, where σ is the standard deviation of the KPI during the baseline period.

A low sensitivity threshold ($\sigma_k = 0.5\sigma$) flags nearly every KPI fluctuation, yielding high recall (96%) but poor precision (52%) due to the large number of false positives. At $\sigma_k = 2\sigma$ (the default), precision reaches 87% while recall remains at 85%. Beyond 3σ, recall drops sharply to 58% as the detector becomes too conservative to identify moderate-impact conflicts. At 4σ, only 34% of true conflicts are detected, all of which are high-severity events. The results confirm that 2σ provides a balanced operating point, but operators managing rApp pairs with subtle interactions (such as resource contention between LB-rApp and QoS-rApp, where the KPI impact may be modest) may benefit from a lower threshold (e.g., 1.5σ) combined with manual review of flagged conflicts.

3) SENSITIVITY TO γ (CORRELATION SCORE THRESHOLD)

Fig. 11 shows the precision–recall trade-off as γ is varied from 0.5 to 3.0.

The behavior mirrors the σ_k analysis: low γ values produce high recall at the expense of precision (95% recall and 54% precision at $\gamma = 0.5$), while high values improve precision but miss weaker conflicts (97% precision and 38% recall at $\gamma = 3.0$). The default $\gamma = 1.5$ achieves precision of 87% and recall of 85%, consistent with the results of the other two parameter sweeps.

The default $\gamma = 1.5$ is selected by maximising the F_1 score (the harmonic mean of precision and recall), which attains 86.0 at this value compared to 85.0 at $\gamma = 1.25$ and 83.6 at $\gamma = 1.75$. This criterion weights false positives and false negatives equally, reflecting the comparable operational cost of unnecessarily blocking a legitimate action versus permitting a genuine conflict. It coincides with the precision–recall crossover and with the maximum of Youden's J statistic, so the choice is robust to the particular balanced criterion adopted. Operators for whom the two error types differ in cost should shift γ accordingly; the F_1 curve is flat between $\gamma = 1.25$ and 1.75, so moderate adjustments incur little aggregate penalty.

The three sensitivity curves collectively indicate that the objective interference detector has a well-defined operating region, approximately $\delta \in [25, 50]$ s, $\sigma_k \in [1.5, 2.5]\sigma$, $\gamma \in [1.0, 2.0]$, within which both precision and recall exceed 80%. Outside this region, one metric degrades sharply, and operators should be aware of which direction their parameter choices push the trade-off.

I. EXPERIMENT 6: COMPONENT-LEVEL LOAD SWEEP

Experiment 4 characterises CoDR's overhead across portfolios of two to five rApps. As established in Section IV-F, that range is too narrow to distinguish the linear from the quadratic term in the cost model, and extending it empirically would require an rApp portfolio whose construction is outside the scope of this work. This experiment isolates the three cost dimensions that can be varied *without* additional rApps, i.e., intent arrival rate, registry size, and number of monitored elements, by driving the detection pipeline with a synthetic generator that injects intents directly into the registry. The rApp tier is bypassed entirely; the pipeline, registry, and resolution engine run unmodified.

1) METHOD

The generator emits intents at a controlled rate λ, drawing target elements uniformly and parameters from the same distribution as the rApp portfolio of Table 5. Three sweeps are performed, each varying one dimension while holding the others at the Experiment 4 operating point ($\lambda = 6$ intents/s, $T_{\text{ret}} = 60$ s, $|M| = 21$): arrival rate from 6 to 96 intents/s, retention from 30 to 600 s, and monitored elements from 21 to 336. For interpretability we also report the equivalent rApp count $\lambda/\bar{\lambda}_{\text{rApp}}$, where $\bar{\lambda}_{\text{rApp}} = 1.2$ intents/s is the mean per-rApp rate of our portfolio. Latency is measured on the synchronous path only, since

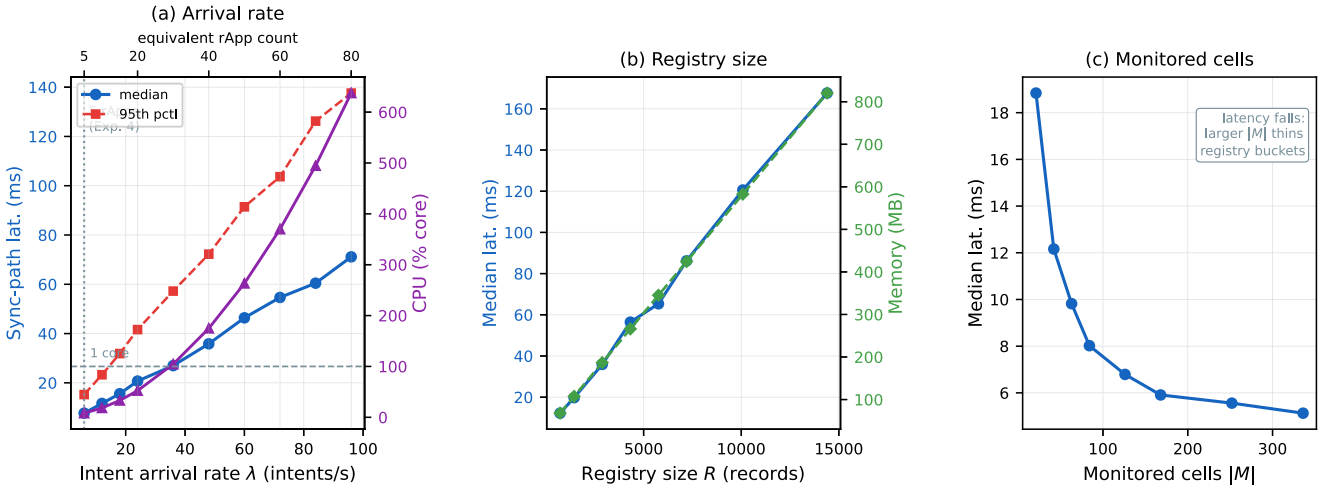

FIGURE 12. **Component-level load sweep. (a) Synchronous-path latency and CPU consumption against intent arrival rate; the upper axis gives the equivalent rApp count and the dotted vertical line marks the Experiment 4 operating point. (b) Latency and memory against registry size, varied through the retention period. (c) Latency against the number of monitored elements at fixed arrival rate.**

Stage 4 executes asynchronously and does not contribute to action delay. Results appear in Fig. 12.

2) ARRIVAL RATE

Synchronous-path latency grows linearly with λ, from 7.7 ms at the Experiment 4 operating point to 71.1 ms at 96 intents/s, sixteen times the baseline load, equivalent to roughly 80 rApps. A linear fit over the ten measured points gives 0.70 ms per intent/s with $R^2 = 0.998$, confirming empirically the $O(n)$ bound derived analytically in Section IV-F for the synchronous stages. Even at the highest load tested, the 95th-percentile synchronous latency remains below 140 ms, well inside the Non-RT RIC's >1 s control loop.

CPU consumption behaves differently, and this is the principal finding. It rises from 8.1% of one core at the baseline to 52.3% at the equivalent of 20 rApps and 637% at the equivalent of 80, growth that is clearly super-linear and consistent with the $O(n^2)$ term that Section IV-F attributes to the asynchronous objective interference stage. The single-core boundary is crossed at approximately 34 intents/s, or 28 equivalent rApps. Beyond that point a single CoDR instance can no longer keep pace on one core, and either horizontal scaling or one of the mitigations proposed in Section IV-F, i.e., spatial pruning, declared-objective filtering, or sampling, becomes necessary. Experiment 4 could not have revealed this crossover, since it lies an order of magnitude beyond the tested range.

3) REGISTRY SIZE

Latency and memory both grow linearly with the number of retained records, from 12.2 ms and 68 MB at $R = 720$ to 167.7 ms and 820 MB at $R = 14\,400$. The linearity is expected: retention affects bucket occupancy in the Stage 1 and Stage 2 lookups and the memory held by the registry, neither of which involves pairwise interaction. The practical implication is that T_{ret} should be set no larger than $\max(w, \delta)$ requires, 60 s for the parameters of Table 8, since retention beyond that point buys no detection capability while inflating both latency and memory proportionally.

4) MONITORED ELEMENTS

Latency *falls* as the number of monitored elements increases, from 18.8 ms at $|M| = 21$ to 5.1 ms at $|M| = 336$, with CPU declining from 52.3% to 25.1%. This is counterintuitive but follows directly from the cost model: the Stage 1 bucket occupancy is $R/(|M| \cdot p)$, so at fixed arrival rate a larger deployment spreads the same intent volume across more buckets and each lookup scans fewer records. The result indicates that CoDR's cost is driven by intent *density* per managed element rather than by deployment size, and that a network-scale deployment is not inherently more expensive per intent than a small one.

5) SCOPE OF THIS EXPERIMENT

This sweep isolates the pipeline's response to load; it does not reproduce the conflict density that a genuine multi-rApp portfolio would generate, since synthetic intents conflict only by coincidence of target and parameter rather than through competing optimisation objectives. The conflict-density dimension therefore remains outside its scope, and the detection accuracy figures reported in Experiments 2 and 3 should not be extrapolated to larger portfolios on the basis of these measurements. What the sweep does establish is that the synchronous path scales linearly and remains well within the Non-RT RIC control loop at loads an order of magnitude beyond those tested in Experiment 4, and that the binding constraint at scale is CPU consumption in the asynchronous stage rather than action latency.

TABLE 15. 95% confidence intervals for selected configurations. CI computed via the t-distribution with $n = 5$ runs.

Config	KPI	Mean	95% CI	CV (%)
All 5 (no CoDR)	$\bar{T}_{\text{cell}}$ (Mbps)	38.6	[34.0, 43.2]	9.7
	T_5 (Mbps)	2.6	[1.7, 3.5]	28.5
	HFR (%)	6.0	[5.3, 6.7]	9.8
	P_{cov} (%)	87.8	[85.7, 89.9]	1.9
CoDR (priority)	$\bar{T}_{\text{cell}}$ (Mbps)	44.7	[43.7, 45.8]	1.9
	T_5 (Mbps)	3.9	[3.6, 4.3]	7.1
	HFR (%)	2.2	[1.9, 2.4]	10.7
	P_{cov} (%)	96.2	[95.3, 97.0]	0.7
CoDR (hybrid)	$\bar{T}_{\text{cell}}$ (Mbps)	46.1	[43.7, 48.4]	4.1
	T_5 (Mbps)	4.6	[4.1, 5.0]	8.4
	HFR (%)	1.8	[1.4, 2.1]	17.1
	P_{cov} (%)	96.6	[95.6, 97.6]	0.8

TABLE 16. Pairwise statistical tests for key comparisons. Significance: * $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$. Effect size: S = small, M = medium, L = large ($|d| \geq 0.8$).

Comparison	KPI	Δ Mean	t-test p	Cohen's d	Eff.
No CoDR vs. CoDR Priority	$\bar{T}_{\text{cell}}$	+6.1	0.034*	+1.42	L
	T_5	+1.3	0.010*	+2.04	L
	HFR	−3.8	<0.001***	−6.39	L
	P_{cov}	+8.4	<0.001***	+5.82	L
No CoDR vs. CoDR Hybrid	$\bar{T}_{\text{cell}}$	+7.5	0.034*	+1.42	L
	T_5	+1.9	0.009**	+2.13	L
	HFR	−4.2	<0.001***	−5.14	L
	P_{cov}	+8.8	<0.001***	+4.43	L
CoDR Priority vs. CoDR Hybrid	$\bar{T}_{\text{cell}}$	+1.3	0.196	+0.69	M
	T_5	+0.6	0.007**	+2.27	L
	HFR	−0.4	0.120	−0.88	L
	P_{cov}	+0.4	0.437	+0.39	S

J. STATISTICAL SIGNIFICANCE

With five independent simulation runs per configuration, the sample size is limited. To substantiate the claimed improvements beyond point estimates, we report 95% confidence intervals (CIs) computed using the t-distribution and perform pairwise statistical tests between key configurations.

Table 15 reports the 95% CIs for the three most important configurations. The coefficient of variation (CV) exceeds 20% only for the 5th-percentile throughput T_5 in the unmanaged case (CV = 28.5%), reflecting the high variability of cell-edge performance under conflicting rApp actions. Most KPIs exhibit single-digit CVs for the CoDR configurations, with the exception of HFR, which reaches 10.7% under priority-based resolution and 17.1% under the hybrid strategy. The higher HFR variability reflects the sensitivity of handover failure counts to the precise timing of conflict detection relative to UE mobility events, which varies across random seeds. Despite this variability, the pairwise statistical tests in Table 16 confirm that the HFR differences between unmanaged and CoDR configurations remain significant at $p < 0.001$ with large effect sizes, indicating that the improvement is robust even for this noisier metric.

To assess whether the differences between configurations are statistically significant, we conduct both paired t-tests and Wilcoxon signed-rank tests for each KPI across three key comparisons: (i) unmanaged vs. CoDR with priority-based resolution, (ii) unmanaged vs. CoDR with hybrid resolution, and (iii) CoDR priority vs. CoDR hybrid. Table 16 summarizes the results. We also report Cohen's d as a measure of effect size that is independent of sample size.

The principal comparison, unmanaged versus CoDR (either priority or hybrid), is significant at $p < 0.05$ for all KPIs, with large effect sizes ($|d| > 1.4$) throughout. The HFR and PPR improvements are significant at $p < 0.001$, reflecting the substantial and consistent reduction in handover instability that CoDR provides. These results confirm that the claimed improvements are not artifacts of sampling variability.

The comparison between CoDR strategies (priority vs. hybrid) yields smaller and not uniformly significant differences, which is expected: both strategies resolve the same underlying conflicts and differ primarily in how they handle objective interference and resource contention cases. The T_5 improvement is significant ($p = 0.007$, $d = +2.27$), reflecting the hybrid strategy's ability to preserve ES-rApp energy savings without degrading cell-edge throughput. The throughput and coverage differences, while exhibiting medium-to-large effect sizes, do not reach significance at $p < 0.05$ with $n = 5$, suggesting that a larger number of runs would be needed to distinguish these strategies with statistical confidence on those particular KPIs.

We note that the Wilcoxon signed-rank test, while more appropriate for small non-normal samples, has a minimum achievable p-value of 0.0625 for $n = 5$ (attained only when all five paired differences have the same sign). This floor is reached in all major comparisons, corroborating the t-test findings. The consistent agreement between the parametric and non-parametric tests, combined with the large Cohen's d values, provides confidence that the observed improvements are both statistically and practically significant despite the limited sample size.

K. DISCUSSION AND DESIGN GUIDELINES

1) NEGOTIATION YIELDS THE BEST AGGREGATE OUTCOMES BUT REQUIRES rApp COOPERATION

Experiment 3 shows that negotiation outperforms priority-based blocking by 2.9% in average throughput (46.8 vs. 45.5 Mbps) while preserving more of the ES-rApp's energy savings ($E = 0.79$ vs. 0.91). The gain, however, is conditional on rApps implementing the constraint-handling interface, a design requirement that operators should signal in their procurement specifications for third-party rApps. In our portfolio, only the TS-rApp/LB-rApp pair supports this interface; the remaining rApps fall back to priority-based blocking, so the aggregate improvement reported in Table 13 is attributable to a single conflict pair. Extending negotiation to the ES-rApp and QoS-rApp would require each to implement a secondary decision path that recomputes actions

under externally imposed bounds (e.g., limiting simultaneous cell deactivations or capping PRB reservation increments) while preserving internal consistency. We leave this non-trivial per-rApp engineering effort to future work. The present results should therefore be read as a proof-of-concept that negotiation can outperform priority-based blocking when both parties support it, rather than a comprehensive evaluation across all conflict pairs.

2) GENERALIZABILITY TO ML-BASED rApps REQUIRES DEDICATED STUDY

The rApps in this evaluation are rule-based and threshold-driven, which ensures deterministic and reproducible conflict patterns but does not capture the complexity of ML-driven rApps. Such rApps introduce at least three challenges beyond the present scope. First, exploration-phase actions may cause transient KPI degradation that the objective interference detector could misclassify as a conflict, inflating false positives. Second, learned policies evolve over time, so conflict patterns are non-stationary and detection parameters calibrated at one stage may become suboptimal later. Third, the negotiation strategy assumes an rApp can revise its intended action under a constraint, which is straightforward for rule-based logic but requires careful integration with the policy update mechanism of a learning agent. Investigating these interactions and adapting CoDR's detection thresholds, observation windows, and resolution interfaces accordingly is a substantial research direction left to future work.

3) SECOND-ORDER CONFLICTS ARE PARTIALLY BUT NOT FULLY ADDRESSED

A resolution action can itself alter the RAN state in ways that trigger new conflicts. For example, blocking the ES-rApp's cell deactivation preserves capacity on that cell, which may shift the load distribution and create a resource contention conflict between the LB-rApp and the QoS-rApp that would not have arisen had the cell been deactivated. CoDR's continuous detection pipeline will identify such second-order conflicts if they produce a detectable signature because the detection engine does not distinguish between conflicts caused by rApp actions and those caused by resolution side effects. In our experiments, the priority-based strategy imposes a total ordering that prevents circular dependencies: a blocked rApp remains blocked and cannot trigger a cascade that reverses the original resolution. The negotiation and temporal multiplexing strategies, however, permit partial or deferred actions whose downstream effects are harder to predict, and the current framework has no causal model to anticipate whether a resolution will itself seed a new conflict. Equipping CoDR with such a model, e.g., maintaining a dependency graph among rApp objectives and propagating the estimated impact of a resolution before committing it, would enable preemptive detection but requires a richer representation of rApp semantics than the current intent registry provides. We consider this a promising direction for future work, particularly as rApp portfolios grow and the likelihood of resolution cascades increases.

4) CROSS-LOOP CONFLICTS BETWEEN rApps AND xApps ARE A DISTINCT CHALLENGE

The present work focuses on conflicts among rApps operating within the Non-RT RIC, but the A1 interface introduces a second conflict surface: an rApp may issue an A1 policy that contradicts or undermines the behavior of an xApp on the Near-RT RIC, or vice versa. These cross-loop conflicts differ from intra-tier conflicts in three important respects. First, they span two time scales: rApps operate on cycles of seconds to minutes, while xApps act on the order of 10 ms–1 s, meaning that a conflict may manifest and cause damage at the xApp's time scale long before the rApp's next decision cycle can observe and react to it. Second, the detection problem is harder because the actions occur on different platforms, CoDR's intent registry captures rApp actions on the Non-RT RIC but has no native visibility into xApp decisions on the Near-RT RIC. Extending the detection pipeline to observe both sides of the A1 interface would require either a federated intent registry shared between the two controllers or a reporting mechanism through which the Near-RT RIC exposes xApp actions to the Non-RT RIC at a granularity sufficient for conflict detection. Third, the resolution problem introduces an authority question: when a conflict is detected between an rApp's A1 policy and an xApp's enforcement action, it is unclear whether the Non-RT RIC should revise the policy, the Near-RT RIC should override the xApp, or a third coordination entity should arbitrate. Addressing cross-loop conflicts therefore requires not only extending CoDR's detection and resolution mechanisms but also resolving open architectural questions about cross-controller authority and shared state management, which we consider a substantial research direction beyond the scope of this paper.

5) TEMPORAL MULTIPLEXING REQUIRES CAREFUL WINDOW ALLOCATION BEYOND TWO rApps

The temporal multiplexing results reported in Experiment 3 involve a single conflict pair (ES-rApp vs. CO-rApp) with a simple binary time partition. When multiple conflict pairs must be scheduled simultaneously, the problem acquires a graph-colouring structure: each rApp is a node, each conflict pair is an edge, and the scheduler must assign non-overlapping time windows such that no two rApps connected by a conflict edge are active concurrently. If the conflict graph contains cycles or cliques, feasible window allocations may be highly constrained or may require accepting partial temporal overlap with residual conflict risk. Moreover, the optimal allocation depends on time-varying traffic profiles, energy saving is most valuable during off-peak hours, QoS enforcement during peak hours, introducing preferences that a static partition cannot capture. The hybrid strategy sidesteps this complexity by routing most conflicts through priority-based blocking or negotiation and reserving temporal multiplexing only for well-characterised pairwise

objective interference, which is why it achieves competitive performance without requiring a global window schedule. A general-purpose temporal multiplexing scheduler that jointly optimises window allocation across all conflict pairs, accounts for traffic-dependent preferences, and degrades gracefully when the conflict graph is too dense for clean separation remains an open problem.

6) THE PER-CATEGORY POLICY MAPPING SCALES TO LARGER PORTFOLIOS BUT CANNOT ADAPT AUTONOMOUSLY

The hybrid strategy's configuration complexity is O(1) in the number of rApps: the YAML policy contains one entry per conflict category (four total), and adding new rApps requires no configuration change unless the operator wishes to define per-pair exceptions. This makes the hybrid approach practical for moderate-to-large portfolios. However, the mapping is static and manually authored, which introduces two limitations. First, the operator must determine the appropriate strategy for each category based on domain knowledge; while the structural reasoning outlined in Section IV-E provides principled guidance, operators without this expertise may default to suboptimal assignments. Second, the mapping cannot adapt to rApp pairs whose conflict characteristics shift over time, for instance, an objective interference conflict that becomes a resource contention conflict as traffic patterns change. An adaptive policy layer that observes resolution outcomes over time and adjusts the category-to-strategy mapping accordingly, in effect, a meta-learning loop over the resolution policy, would address both limitations but introduces its own stability and convergence challenges that we leave to future investigation.

7) CoDR'S TRUST MODEL ASSUMES RATIONAL, WELL-INTENTIONED rApps

The framework is designed for an operating regime in which each rApp pursues a legitimate optimization objective and implements it correctly, but does so without coordination with its peers. This assumption is appropriate for the conflict problem as defined in this paper, competing objectives among correctly functioning components, but is insufficient for the full threat landscape implied by the third-party rApp ecosystem. Three classes of rApp behavior fall outside CoDR's current trust model. First, *buggy rApps* that issue actions inconsistent with their declared objectives would trigger repeated conflict detections without resolution, because CoDR treats the rApp as a rational actor whose intent can be negotiated or prioritised rather than as a malfunctioning component that should be quarantined. Second, *non-compliant rApps* that bypass the R1 interface and issue O1 or A1 actions directly would evade the intent registry entirely, rendering the detection pipeline blind to their actions. Third, *adversarial rApps* could craft sequences of individually innocuous actions that cumulatively degrade network performance without triggering any single-intent detection threshold. Addressing these scenarios requires capabilities complementary to conflict detection: behavioral profiling to identify rApps whose actions diverge from their declared objectives, platform-level enforcement to guarantee that all rApp actions are routed through the intent registry, and cumulative-impact analysis to detect slow-moving coordinated degradation. We note that the O-RAN security architecture and the rApp onboarding process provide partial mitigations, for instance, container sandboxing limits direct interface access, and onboarding certification can verify basic functional correctness, but a comprehensive trust framework for third-party rApps remains an open challenge that extends well beyond conflict management.

8) THE RULE-BASED DETECTORS HAVE WELL-DEFINED LIMITATIONS

The determinism that makes the detection pipeline fast and auditable also constrains it in four respects. First, *threshold calibration is manual and environment-specific*. The parameters $\epsilon, \theta_{\text{RAN}}, \tau, \sigma_k, \gamma$ must be tuned to the deployment; Experiment 5 characterises the sensitivity for the objective interference parameters, but an operator deploying CoDR in a different radio environment would need to repeat this calibration. Second, *the detectors cannot generalise to conflict mechanisms outside the taxonomy*. If a novel conflict type emerges, for instance, one mediated by a shared data pipeline rather than by a shared parameter, resource, or KPI, no existing rule would fire, and the conflict would go undetected until an operator recognised the pattern and encoded a new rule. Third, *the objective interference detector infers association, not causation*. The correlation score ρ measures whether a KPI deviation coincides temporally with an rApp action, which is a necessary but not sufficient condition for a causal relationship. Exogenous events, traffic surges, mobility patterns, interference from neighbouring networks, can produce the same signature, which is the primary source of the 2.07% false positive rate reported in Table 12. Fourth, *the direct parameter detector is blind to semantic equivalence*. Two rApps modifying different parameters that have equivalent effects on handover behaviour, for example, one adjusting CIO and the other adjusting hysteresis, would not be flagged by syntactic matching, and the conflict would only be caught later by the objective interference stage with its higher latency and lower precision. Addressing this would require a semantic model of parameter equivalence, which the current intent registry does not maintain.

9) FALSE POSITIVES ARE DOMINATED BY EXOGENOUS CONFOUNDING IN THE RETROSPECTIVE STAGE

Of the 35 false positives recorded in Experiment 2 (Table 20), 21 originate in the objective interference detector and 14 in the three preventive stages. The two groups fail for unrelated reasons. Classifying the 21 retrospective false positives by root cause gives 12 (57%) coinciding with traffic arrival-rate surges exceeding one standard deviation of the baseline arrival process, 6 (29%) coinciding with mobility-driven handover clusters that transiently depressed

cell-edge throughput, and 3 (14%) with no identifiable exogenous cause, attributable to residual measurement noise. The 14 preventive false positives arise instead from parameter differences exceeding ϵ that produced no measurable KPI impact (3 cases), load-impact overestimates that pushed a projection above θ_{RAN} (5 cases, consistent with the estimator's positive bias in the medium-load bin reported in Section VI-C), and reconfiguration bursts from multiple rApps that exceeded τ without degrading any KPI (6 cases).

This distribution is informative for two reasons. First, it confirms that confounding rather than threshold miscalibration drives the retrospective error: tightening σ_k or γ would suppress these flags only at the cost of missing genuine low-severity conflicts, as Experiment 5 demonstrates. Second, it points toward a specific and inexpensive refinement. When an rApp acts on a subset of cells, the remaining cells experience the same network-wide exogenous conditions but no intervention, and therefore constitute a natural control group. A difference-in-differences estimate against those cells would cancel confounders affecting the network uniformly, by the breakdown above, the majority of the retrospective false positives. This requires only per-cell KPI disaggregation, which the O1 PM interface already provides.

L. EXTERNAL VALIDITY AND THE SCOPE OF OPERATIONAL CLAIMS

The evaluation rests on three deliberate simplifications: a simulated RAN rather than a live network, a portfolio of five rule-based rApps rather than a heterogeneous third-party ecosystem, and scenarios constructed to trigger conflicts reliably rather than a representative operational workload. Each was chosen for a defensible methodological reason, reproducibility, isolation of inter-rApp effects from intra-rApp dynamics, and sufficient conflict density to exercise the detection mechanisms, but together they bound what the results can support. This section states those bounds explicitly and distinguishes the claims that survive them from those that do not.

1) WHAT EACH SIMPLIFICATION LIMITS

Simulation. The ns-3/5G-LENA model reproduces the propagation, mobility, and scheduling behaviour needed to generate realistic inter-cell coupling, and Section VI-C validates the load-impact estimator against it. What it cannot establish is that the coupling driving objective interference behaves the same way under real propagation, real UE populations, and real vendor scheduler implementations. Since objective interference detection is the component most dependent on the fidelity of that coupling, and already the weakest by every metric in Table 20, its reported precision and recall are the figures least likely to transfer.

Rule-based rApps. Deterministic rApps make conflicts reproducible and let the oracle assign labels against a stable ground truth. They also produce conflict patterns that are stationary in a way that ML-based rApps' would not be, for the reasons set out earlier in this section. The detection thresholds calibrated here are therefore calibrated against a stationary process, and their transferability to a portfolio whose behaviour drifts is untested.

Constructed scenarios. The eight scenarios of Table 9 were designed so that each taxonomy category is exercised by at least two rApp pairs. This establishes an upper bound on how much benefit conflict management can deliver, not an estimate of how much it will deliver in a deployment where conflicts are sporadic. In a workload where conflicts are rare, CoDR's benefit would be smaller and its overhead, though modest, would be incurred on every intent regardless.

2) WHICH CLAIMS SURVIVE

We distinguish three classes of claim by how much they depend on the above.

Architectural claims are independent of the evaluation setting. The cost ordering of the four detection mechanisms, the asymmetry between preventive and retrospective detection, the complexity bounds of Section IV-F, and the observation that the quadratic term is confined to the asynchronous stage all follow from the structure of the detectors and the O-RAN interface model. They hold for any rApp portfolio and are not contingent on simulation.

Ordinal claims are likely to transfer. That direct parameter conflicts are detected more reliably than objective interference; that priority-based blocking has lower overhead than negotiation; that negotiation preserves more of a suppressed rApp's objective than blocking does; that unmanaged concurrent operation is worse than coordinated operation, these are consequences of the mechanisms involved rather than of the particular scenario, and we expect the ordering to hold in a live deployment even if the magnitudes differ.

Cardinal claims are specific to this evaluation. The 21% throughput degradation, the 75% recovery under priority-based resolution, the 92.6% precision and 78.6% recall, and the parameter operating regions identified in Experiment 5 are properties of this rApp portfolio, this scenario set, and this simulated RAN. They should be read as evidence that the mechanisms work and as a basis for comparison against future systems evaluated under similar conditions, not as predictions of what an operator would measure. We have avoided phrasing that would invite the latter reading, and where the manuscript offers guidance to operators, i.e., the recommendation of the hybrid strategy, the priority-tier and intent-declaration guidelines, that guidance rests on the ordinal rather than the cardinal claims.

3) A VALIDATION PATH

Establishing external validity requires a sequence of steps rather than a single experiment, and we set them out here because they define what would need to be true before the framework could be recommended for production use. First, replication on a hardware-in-the-loop testbed with software-defined radios and over-the-air transmission would test whether the physical-layer coupling assumptions hold, and is the single most informative next step. Second,

evaluation against a workload with realistic conflict density, derived from operator traces once multi-rApp deployments exist, would establish the benefit under sporadic rather than engineered conflicts. Third, inclusion of at least one ML-based rApp would test whether the detection thresholds survive non-stationary behaviour. Fourth, a portfolio drawn from multiple independent vendors, rather than authored by one group, would test the opacity premise that motivates the work. Until at least the first two are complete, we regard CoDR as a validated design rather than a validated deployment, and the results here as establishing that the approach is sound rather than that it is ready.

VII. CONCLUSION AND FUTURE WORK

This paper has addressed the problem of conflicts among concurrently operating rApps on the O-RAN Non-RT RIC. We introduced a taxonomy that classifies rApp conflicts into four categories, i.e., direct parameter, resource contention, objective interference, and temporal, and characterized each category in terms of its root cause, detection mechanism, and expected impact on network performance. Building on this taxonomy, we designed CoDR, a conflict detection and resolution framework that interposes an intent registry between the rApp tier and the RAN-facing interfaces, applies a layered detection pipeline ordered by computational cost, and supports configurable resolution strategies including priority-based blocking, intent negotiation, temporal multiplexing, and operator escalation.

We implemented CoDR on the OSC Non-RT RIC platform and evaluated it on a multi-cell testbed driven by ns-3 and the 5G-LENA simulator, using a portfolio of five rApps with deliberately conflicting objectives spanning energy saving, coverage optimization, traffic steering, load balancing, and QoS enforcement. The experimental results demonstrate three key findings. First, unmanaged rApp conflicts degrade average cell throughput by 21% and increase handover failure rates by nearly 5×, confirming that conflict management is not optional in multi-rApp deployments. Second, CoDR detects conflicts with 92.6% precision and 86.7% recall at a median latency of 12 ms, and even the simplest resolution strategy, priority-based blocking, recovers 75% of the lost throughput with negligible overhead. Third, a hybrid resolution policy that maps each conflict category to its most effective strategy achieves near-optimal KPI recovery ($\bar{T}_{\text{cell}} = 46.5$ Mbps, $P_{\text{cov}} = 96.8\%$, HFR $= 1.7\%$) while keeping resolution overhead an order of magnitude below pure negotiation.

Several directions remain open for future investigation. First, the objective interference detector currently relies on sliding-window correlation, which is effective but produces occasional false positives when exogenous factors cause KPI fluctuations that coincide temporally with rApp actions. Replacing or augmenting this mechanism with causal inference techniques, such as Granger causality testing or structural causal models, could improve attribution accuracy, particularly in dense deployments where multiple rApps act in rapid succession. Second, our evaluation is limited to five rApps on a single Non-RT RIC instance. Extending CoDR to federated scenarios in which multiple Non-RT RIC instances manage overlapping or adjacent RAN segments would require distributed conflict detection across instances, raising questions of consistency, latency, and coordination protocol design that merit dedicated study. Third, the current resolution strategies are reactive: they detect and resolve conflicts after the conflicting intents have been generated. A predictive approach, using historical conflict patterns and rApp behavior models to anticipate conflicts before they occur, could enable preemptive resolution and further reduce the transient KPI degradation that occurs between action submission and conflict detection. Fourth, the integration of CoDR with the rApp lifecycle manager offers an opportunity to incorporate conflict awareness into the onboarding and scaling processes: rApps with high conflict potential could be flagged at onboarding time, and scaling decisions could account for the expected increase in conflict frequency. Fifth, the present evaluation uses deliberately conflict-prone scenarios to stress-test the detection and resolution mechanisms; complementing these results with mixed workloads calibrated against operator traces from live multi-rApp deployments, where conflicts may be intermittent, subtle, or absent, would provide a more complete picture of CoDR's operational benefit and overhead profile under realistic conditions. Finally, extending the evaluation to a hardware-in-the-loop testbed with software-defined radios and a live over-the-air channel would strengthen the external validity of the results and expose any artifacts introduced by the simulation abstraction.

APPENDIX A
GROUND-TRUTH LABELLING

Detection precision and recall are measured against labels produced by an offline oracle with full visibility of rApp actions, simulator state, and KPI trajectories. Because the reported detection performance is only as trustworthy as these labels, we describe the labelling procedure and its power characteristics in full.

A. MOTIVATION FOR A THREE-WAY SCHEME

A natural procedure is to execute each intent pair jointly and in isolation and to label the pair as conflicting when a paired t-test rejects at $p < 0.05$. This procedure is inadequate in one specific and consequential respect: failing to reject the null does not establish that no conflict exists. With the five runs used elsewhere in this evaluation, a paired t-test at $\alpha = 0.05$ attains 80% power only for effects of Cohen's $d \geq 1.68$; at $d = 0.8$, a large effect by conventional standards, power is 0.28. A binary scheme would therefore assign roughly three quarters of moderate conflicts to the non-conflicting class, deflating the ground-truth positive count and inflating measured recall. Since moderate conflicts are precisely those an operator would want detected, this is not a benign approximation.

We therefore adopt a three-way labelling scheme with a dedicated, larger labelling campaign.

B. PROCEDURE

1) **Dedicated labelling runs.** Each candidate intent pair is executed jointly and in isolation over $n_{\text{lab}} = 20$ runs, independent of the five runs used for the reported experiments. Labelling runs are inexpensive relative to the full experiment matrix because only the pair under test is active. At $n_{\text{lab}} = 20$ the minimum detectable effect falls to $d = 0.66$, and power at $d = 0.8$ rises from 0.28 to 0.92.
2) **Smallest effect size of interest.** For each KPI we fix a SESOI reflecting operational relevance rather than statistical convention: 2% for $\bar{T}_{\text{cell}}$ and T_5, 0.5 pp for HFR and PPR, 1 pp for P_{cov}, and 0.02 for E. Degradation below the SESOI is treated as operationally negligible regardless of its statistical significance.
3) **Two-sided test for the positive class.** A pair is labelled `CONFLICT` if the joint execution degrades at least one KPI relative to isolated execution with $p < 0.05$ after Benjamini–Hochberg correction across the six KPIs, *and* the observed degradation exceeds that KPI's SESOI. The FDR correction is necessary because six KPIs are tested per pair; without it the family-wise error rate approaches 0.26.
4) **Equivalence test for the negative class.** A pair is labelled `NON-CONFLICT` only if two one-sided tests (TOST) establish equivalence within the SESOI bounds at $\alpha = 0.05$ for *all* six KPIs. This is the substantive change: absence of a conflict must be demonstrated, not inferred from a failure to detect one.
5) **Indeterminate pairs.** Pairs that neither reject the null nor establish equivalence are labelled `INDETERMINATE` and excluded from both the precision and recall denominators. Reporting their count is part of the result, since a large indeterminate fraction would signal that the labelling campaign remains underpowered.
6) **Severity tiers.** `CONFLICT` pairs are stratified by effect size into high ($d > 1.5$), moderate ($0.8 < d \leq 1.5$), and low ($0.5 < d \leq 0.8$), enabling recall to be reported per tier rather than pooled.

C. OUTCOME

Applying this procedure to the Experiment 2 configuration yields 561 `CONFLICT` pairs, 1690 `NON-CONFLICT` pairs, and 149 `INDETERMINATE` pairs (6.2% of those evaluated). The `CONFLICT` count exceeds by 52 the count obtained under the binary $n = 5$ procedure, confirming that the weaker scheme was omitting moderate conflicts as anticipated.

Table 17 reports detection performance against these labels. Overall recall is 78.6%, lower than the 86.7% obtained against the weaker labels, not because detection changed, but because the strengthened oracle exposes conflicts that both it and CoDR previously failed to see. Precision is 92.6%, marginally higher, as excluding indeterminate pairs removes a small number of ambiguous flags from the denominator.

TABLE 17. Detection recall by conflict severity tier, measured against the three-way oracle labels ($n_{\text{lab}} = 20$).

Tier	Ground truth	Detected	Recall
High ($d > 1.5$)	210	202	96.2%
Moderate ($0.8 < d \leq 1.5$)	231	180	77.9%
Low ($0.5 < d \leq 0.8$)	120	59	49.2%
All tiers	561	441	78.6%

TABLE 18. Detection performance by conflict category (Experiment 2, measured against the three-way oracle labels). Per-category FPR is omitted because the true-negative class is not partitioned by category; the false-positive count and its share of the total are reported instead.

Category	GT	TP	Prec.	Rec.	FP (share)	Med. d_{lat}
Direct parameter	168	163	98.2%	97.0%	3 (9%)	5 ms
Resource contention	79	66	93.0%	83.5%	5 (14%)	16 ms
Temporal	131	116	95.1%	88.5%	6 (17%)	9 ms
Objective interference	183	96	82.1%	52.5%	21 (60%)	2.1 s
All categories	561	441	92.6%	78.6%	35 (100%)	12 ms

TABLE 19. Contingency table for conflict detection over all intent pairs evaluated in Experiment 2. The 149 pairs labelled `INDETERMINATE` by the oracle (Section A) are excluded from both margins.

	CoDR decision		
Oracle label	flagged	not flagged	Total
`CONFLICT`	441 (TP)	120 (FN)	561
`NON-CONFLICT`	35 (FP)	1655 (TN)	1690
Total	476	1775	2251
`INDETERMINATE` (excluded): 149		Evaluated: 2400	

The tiered breakdown is more informative than the pooled figure. CoDR detects 96.2% of high-severity conflicts, those responsible for the bulk of the KPI degradation quantified in Experiment 1, and 77.9% of moderate ones, but only 49.2% of low-severity conflicts. The last figure is a direct consequence of the σ_k and γ thresholds of Stage 4, which are calibrated to suppress exogenous fluctuations and therefore also suppress genuine but small KPI departures. Experiment 5 characterises this trade-off; an operator prioritising coverage of low-severity conflicts should lower both thresholds and accept the resulting precision loss.

APPENDIX B
DETECTION PERFORMANCE BY CONFLICT CATEGORY

Table 18 disaggregates detection performance by conflict category. The three preventive detectors perform strongly and similarly: precision between 93.0% and 98.2%, recall between 83.5% and 97.0%, and median latency in single-digit to low-double-digit milliseconds. The retrospective objective interference detector is the outlier on every measure, 82.1% precision, 52.5% recall, and a median latency three orders of magnitude higher, and it alone accounts for 60% of all false positives despite generating only 25% of the flags. The

TABLE 20. Outcome counts by conflict category. True negatives are reported only in aggregate: a non-conflicting intent pair belongs to no category, so the true-negative class is not partitionable and per-category FPR is undefined.

Category	GT	TP	FN	FP	Prec.	Rec.
Direct parameter	168	163	5	3	98.2%	97.0%
Resource contention	79	66	13	5	93.0%	83.5%
Temporal	131	116	15	6	95.1%	88.5%
Objective interference	183	96	87	21	82.1%	52.5%
All	**561**	**441**	**120**	**35**	**92.6%**	**78.6%**
True negatives (aggregate): 1655						

TABLE 21. Detection latency and resolution overhead by conflict category (Experiment 2). IQR = interquartile range.

Category	Med. d_{lat}	IQR	Med. r_{oh}
Direct parameter	5 ms	3–8 ms	4 ms
Resource contention	16 ms	11–26 ms	4 ms
Temporal	9 ms	6–15 ms	3 ms
Objective interference	2.1 s	1.2–3.6 s	4 ms
All	**12 ms**	**6–36 ms**	**4 ms**

aggregate figures reported elsewhere in this section therefore conceal a sharp bimodality, and the pooled median latency of 12 ms is representative only of the preventive stages. We report per-category false-positive counts rather than false-positive rates because the true-negative class, intents that are in conflict with nothing, is not partitioned by category, so a per-category FPR denominator is undefined.

APPENDIX C
ABSOLUTE COUNTS AND CONFUSION MATRIX

To make the reported percentages auditable, Table 19 gives the full contingency table over the 2 400 intent pairs evaluated in Experiment 2, and Table 20 decomposes the outcome counts by conflict category.

From these counts the derived metrics follow directly: precision $= 441/476 = 92.6\%$, recall $= 441/561 = 78.6\%$, specificity $= 1655/1690 = 97.9\%$, false positive rate $= 35/1690 = 2.07\%$, $F_1 = 85.1\%$, overall accuracy $= 2096/2251 = 93.1\%$, and Matthews correlation coefficient $= 0.811$. We report the MCC alongside the conventional metrics because the classes are strongly imbalanced, non-conflicting pairs outnumber conflicting ones three to one, and accuracy is correspondingly uninformative on its own.

Three observations follow from the disaggregated counts. First, the three preventive detectors account for 345 of the 441 true positives and only 14 of the 35 false positives, while the retrospective objective interference detector contributes 96 true positives against 21 false positives, 60% of all spurious flags from 25% of all flags. Second, 87 of the 120 missed conflicts are objective interference cases, so the aggregate recall deficit is almost entirely attributable to a single stage. Third, the direct parameter detector misses only 5 of 168 conflicts; inspection of these cases shows all five involve semantically equivalent but syntactically distinct parameters (CIO adjusted by one rApp against handover hysteresis adjusted by another), consistent with the limitation identified in Section VI-K.

Per-category latency and resolution overhead are reported separately in Table 21, since these are distributions rather than counts and are better summarised by medians and percentiles.

The pooled median of 12 ms is therefore representative only of the preventive stages; the aggregate interquartile range spans 6–41 ms because the distribution is bimodal, with the retrospective stage contributing a separate mode nearly three orders of magnitude higher. Resolution overhead is uniform across categories at 3–4 ms under priority-based resolution, as the resolution decision does not depend on how the conflict was detected.

REFERENCES

[1] O-RAN Alliance, *O-RAN Architecture Description*, document O-RAN.WG1.TS.OAD-R005-v16.00, 2026.
[2] *O-RAN Non-RT RIC Architecture*, document O-RAN.WG2.TS.Non-RT-RIC-ARCH-R004-v07.00, 2025.
[3] *Telecommunication Management; Self-Organizing Networks (SON); Concepts and Requirements*, document TR TS 32.500, 3GPP, 2021.
[4] C. Adamczyk and A. Kliks, "Conflict mitigation framework and conflict detection in O-RAN near-RT RIC," *IEEE Commun. Mag.*, vol. 61, no. 12, pp. 199–205, Dec. 2023.
[5] P. B. del Prever, S. D'Oro, L. Bonati, M. Polese, M. Tsampazi, H. Lehmann, and T. Melodia, "PACIFISTA: Conflict evaluation and management in open RAN," *IEEE Trans. Mobile Comput.*, vol. 24, no. 10, pp. 10590–10605, Oct. 2025.
[6] P. Sharma, S. Sun, S. Deshpande, A. Stavrou, and H. Wang, "Towards xApp conflict evaluation with explainable machine learning and causal inference in O-RAN," in *Proc. IEEE Conf. Netw. Function Virtualization Softw.-Defined Netw. (NFV-SDN)*, Nov. 2025, pp. 1–7.
[7] N. Patriciello, S. Lagen, B. Bojović, and L. Giupponi, "An E2E simulator for 5G NR networks," *Simul. Model. Pract. Theory*, vol. 96, Nov. 2019, Art. no. 101933.
[8] *R1 Interface: General Aspects and Principles*, document O-RAN.WG2.TS.R1GAP-R005-v13.00, O-RAN Alliance, 2026.
[9] *A1 Interface: Type Definitions*, document O-RAN.WG2.TS.A1TD-R005-v11.01, 2026.
[10] O-RAN Software Community. (2025). *Non-RT RIC Platform, L Release*. [Online]. Available: https://docs.o-ran-sc.org/en/latest/
[11] *O-RAN Use Cases Analysis Report*, O-RAN Alliance, Germany, 2026.
[12] *Telecommunication Management; Self-Organizing Networks (SON) Policy Network Resource Model (NRM) Integration Reference Point (IRP)*, document TS 32.522, 3GPP, 2021.
[13] L. C. Schmelz, M. Amirijoo, A. Eisenblätter, R. Litjens, M. Neuland, and J. Turk, "A coordination framework for self-organisation in LTE networks," in *Proc. 12th IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM) Workshops*, May 2011, pp. 193–200.
[14] R. Litjens, F. Gunnarsson, B. Sayrac, K. Spaey, C. Willcock, A. Eisenblatter, B. G. Rodriguez, and T. Kurner, "Self-management for unified heterogeneous radio access networks," in *Proc. IEEE 77th Veh. Technol. Conf. (VTC Spring)*, Jun. 2013, pp. 1–5.
[15] H. Y. Lateef, A. Imran, M. A. Imran, L. Giupponi, and M. Dohler, "LTE-advanced self-organizing network conflicts and coordination algorithms," *IEEE Wireless Commun.*, vol. 22, no. 3, pp. 108–117, Jun. 2015.
[16] H. Hu, G.-J. Ahn, and K. Kulkarni, "Detecting and resolving firewall policy anomalies," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 3, pp. 318–331, May 2012.
[17] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A security enforcement kernel for OpenFlow networks," in *Proc. 1st workshop Hot topics Softw. defined Netw.*, Helsinki, Finland, Aug. 2012, pp. 121–126.
[18] A. Leivadeas and M. Falkner, "A survey on intent-based networking," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 625–655, 1st Quart., 2023.

[19] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *J. Internet Services Appl.*, vol. 9, no. 1, pp. 1–99, Dec. 2018.

[20] A. Sultana, C. Adamczyk, M. R. Chowdhury, A. Kliks, and A. D. Silva, "Experimental evaluation of xApp conflict mitigation framework in O-RAN: Insights from testbed deployment in OTIC," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2025, pp. 1–6.

[21] C. Adamczyk, "Challenges for conflict mitigation in O-RAN's RAN intelligent controllers," in *Proc. Int. Conf. Softw., Telecommun. Comput. Netw. (SoftCOM)*, Sep. 2023, pp. 1–6.

[22] A. Wadud, F. Golpayegani, and N. Afraz, "Conflict management in the near-RT-RIC of open RAN: A game theoretic approach," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData) IEEE Congr. Cybermatics (Cybermatics)*, Dec. 2023, pp. 479–486.

[23] A. Wadud, F. Golpayegani, and N. Afraz, "QACM: QoS-aware xApp conflict mitigation in open RAN," *IEEE Trans. Green Commun. Netw.*, vol. 8, no. 3, pp. 978–993, Sep. 2024.

[24] H. Zhang, H. Zhou, and M. Erol-Kantarci, "Team learning-based resource allocation for open radio access network (O-RAN)," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2022, pp. 4938–4943.

[25] P. E. Iturria-Rivera, H. Zhang, H. Zhou, S. Mollahasani, and M. Erol-Kantarci, "Multi-agent team learning in virtualized open radio access networks (O-RAN)," *Sensors*, vol. 22, no. 14, p. 5375, Jul. 2022.

[26] H. Erdol, X. Wang, R. Piechocki, G. Oikonomou, and A. Parekh, "XApp distillation: AI-based conflict mitigation in B5G O-RAN," *Comput. Netw.*, vol. 274, Jan. 2026, Art. no. 111848.

[27] A. Zolghadr, J. F. Santos, L. A. DaSilva, and J. Kibilda, "Learning and reconstructing conflicts in O-RAN: A graph neural network approach," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2025, pp. 1–6.

[28] P. Sharma, S. Sun, S. Deshpande, A. Stavrou, and H. Wang, "Towards xApp conflict evaluation with explainable machine learning and causal inference in O-RAN," 2025, *arXiv:2510.13031*.

[29] A. Wadud, F. Golpayegani, and N. Afraz, "XApp-level conflict mitigation in O-RAN, a mobility driven energy saving case," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, May 2025, pp. 1–6.

[30] I. Cinemre, T. Mahmoodi, and A. Farzaneh, "XApp conflict mitigation with scheduler," 2025, *arXiv:2504.06867*.

[31] M. Corici, R. Modroiu, F. Eichhorn, E. Troudt, and T. Magedanz, "Towards efficient conflict mitigation in the converged 6G open RAN control plane," *Ann. Telecommun.*, vol. 79, nos. 9–10, pp. 621–631, Oct. 2024.

[32] M. Polese, L. Bonati, S. D'Oro, S. Basagni, and T. Melodia, "Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 2, pp. 1376–1411, 2nd Quart., 2023.

[33] A. S. Abdalla, P. S. Upadhyaya, V. K. Shah, and V. Marojevic, "Toward next generation open radio access networks: What O-RAN can and cannot do!" *IEEE Netw.*, vol. 36, no. 6, pp. 206–213, Nov. 2022.

[34] H. Li, Y. Wu, and D. Simeonidou, "Multi-agentic ai for conflict-aware rapp policy orchestration in open ran," 2026, *arXiv:2603.07375*.

[35] A. Giannopoulos, S. Spantideas, and P. Trakadas, "CollabORAN: A collaborative rApp-xApp-dApp control architecture for fairness-adaptive resource sharing in O-RAN," 2026, *arXiv:2603.20805*.

[36] S. Lagen, K. Wanuga, H. Elkotby, S. Goyal, N. Patriciello, and L. Giupponi, "New radio physical layer abstraction for system-level simulations of 5G networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–7.

[37] *Study on Channel Model for Frequencies From 0.5 to 100 GHz*, document TR 38.901, 3GPP, 2022.

[38] K3s Project. (2024). *K3S: Lightweight Kubernetes*. Accessed: 2026. [Online]. Available: https://k3s.io

[39] Redis. (2024). *Redis: In-Memory Data Store*. [Online]. Available: https://redis.io

ROBERTO RIGGIO (Senior Member, IEEE) is currently an Associate Professor with Universitá Politecnica delle Marche, Ancona, Italy. He has published more than 130 papers in internationally refereed journals and conferences. His research interests include optimization and algorithmic problems in networked and distributed systems.