



OrchestRAN: Network Automation through Orchestrated Intelligence in the Open RAN

OrchestRAN: Network Automation through Orchestrated Intelligence in the Open RAN

▼ 1. INTRODUCTION

從傳統網路轉型到 Open RAN 的背景，並精確地指出了目前學術界與產業界在「實現網路智慧化」時遇到的具體瓶頸。

1.1 從傳統 RAN 到 Open RAN

- **硬體解構與軟體化**：第五代 (5G) 與下一代 (NextG) 行動網路標誌著一個時代的結束——即不再依賴僵化、基於硬體的無線接取網路 (RAN) 架構。取而代之的是基於 **軟體化 (Softwarization)**、**開放性 (Openness)** 和 **解構 (Disaggregation)** 原則的創新方案。
- **Open RAN 的優勢**：這種被稱為 **Open RAN** 的典範轉移帶來了前所未有的靈活性。它使得傳統上被綁在單體式 (Monolithic) 基地台中的網路功能可以被拆分，並在網路的多個節點上實例化與控制。

1.2 O-RAN 架構與關鍵組件

論文介紹了 O-RAN 聯盟正在推動的標準化架構，其核心特徵包括：

- **功能拆分 (Functional Split)：** 採用 3GPP 的功能拆分概念，將基地台分為 **Central Units (CUs)**、**Distributed Units (DUs)** 和 **Radio Units (RUs)**，分別執行協議堆疊的不同功能。
- **開放介面 (Open Interfaces)：** 引入一組標準化的開放介面，用於與網路中的每個節點互動、控制並收集數據，從而實現多供應商設備的互通性。
- **RAN 智慧控制器 (RICs)：** 這是 O-RAN 的核心創新，允許在抽象層上執行第三方應用程式來控制 RAN 功能：
 - **near-RT RIC (近即時)：** 執行 **xApps**，處理時間在 10 毫秒到 1 秒之間的操作。
 - **non-RT RIC (非即時)：** 執行 **rApps**，處理時間大於 1 秒的操作。

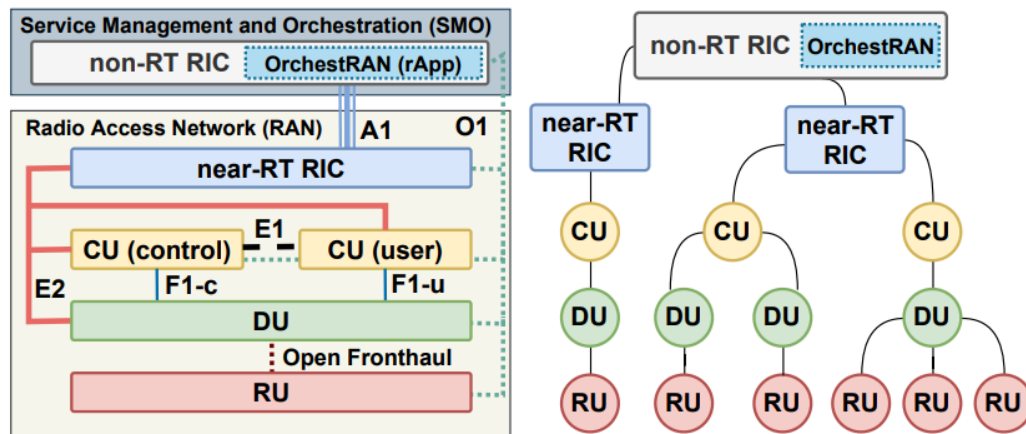


Figure 1. O-RAN reference architecture and interfaces (left). Representation of an O-RAN network architecture as a tree graph (right).

1.3 引入 AI/ML 的願景與現狀

- **自動化與智慧化：** O-RAN 架構使得透過 **機器學習 (ML)** 和 **人工智慧 (AI)** 為網路帶來自動化與智慧成為可能。
- **數據驅動：** 這些技術利用 RAN 產生的巨量數據（透過 O-RAN 介面暴露），來分析當前網路狀況、預測未來流量與需求，並實施閉迴路控制策略以優化效能。
- **現有研究：** 目前學術界與產業界對設計和部署數據驅動解決方案興趣濃厚，應用範圍涵蓋控制 RAN 資源、傳輸策略、以及預測關鍵績效指標 (KPIs)。

1.4 核心問題：編排 (Orchestration) 的挑戰

雖然 AI/ML 潛力巨大，但作者指出「如何部署和管理（即編排）這些智慧」仍然是一個未解難題 (Open Problem)。具體挑戰歸納為三點：

- **時間尺度與輸入數據的可用性 (Complying with time scales and making input available)：**
 - 調整 RAN 參數需要不同時間尺度的控制迴路（從毫秒級的即時到秒級的非即時）。
 - **難題：** 必須選擇正確的模型執行位置，以便在時限內獲取所需輸入。例如，**IQ 樣本 (IQ Samples)** 在 RAN (RU) 很容易取得，但極難在相同的時間窗口內傳輸到 RICs。因此，如果模型需要 IQ 樣本，將其放在 RICs 執行是無效的。
- **選擇正確的模型 (Choosing the right model)：**
 - 每個 ML/AI 模型都是針對特定任務設計的，需要特定的輸入數據類型與大小。
 - **難題：** 必須確保為營運商 (NO) 的請求選擇「最合適」的模型，該模型需滿足性能指標（如最低準確率），提供所需功能，並被實例化在擁有足夠資源的節點上。
- **衝突緩解 (Conflict mitigation)：**
 - **難題：** 必須確保選定的 ML/AI 模型之間不會發生衝突，且在任何時間點，同一個參數（或功能）只能由「單一」模型來控制。

1.5 本論文的解決方案：OrchestRAN

針對上述挑戰，作者提出了 **OrchestRAN**，這是一個針對 Open RAN 的自動化智慧編排框架。

- **運作方式：** 它作為一個 **rApp** 運行在 **non-RT RIC** 上。
- **四大功能流程：**
 1. 收集營運商的高層次控制請求（例如：調整紐約市中心某組基地台的調度）。
 2. 選擇最佳的 ML/AI 模型以達成目標並避免衝突。

3. 決定每個模型的最佳執行位置（例如雲端或邊緣），需考量時間尺度、資源和數據可用性。
4. 自動將模型嵌入到 O-RAN 應用程式中，並派發到選定的節點執行。

1.6 貢獻與驗證

- **演算法：** 作者設計了包含變數縮減 (Variable Reduction) 和分支 (Branching) 技術的新型演算法，以計算不同複雜度與最優性權衡的編排方案。
- **NP-hard 證明：** 論文證明了 Open RAN 中的智慧編排問題屬於 NP-hard。
- **原型驗證：** 作者在 **Colosseum**（世界上最大的硬體迴路無線網路模擬器）上的 **CoIO-RAN** 平台進行了大規模原型測試。
- **關鍵成果：** 實驗（7 個基地台，42 個用戶）顯示 OrchestRAN 能按需實例化服務，並透過自動選擇最佳執行位置（將智慧推向邊緣），將 O-RAN E2 介面上的控制開銷 **減少高達 2.6 倍**。
- **獨特性：** 據作者所知，這是第一個針對 O-RAN 架構的大規模網路智慧編排系統展示。

▼ 2. RELATED WORK

對現有的學術文獻進行全面的回顧與批判性分析，目的是為了凸顯 OrchestRAN 在「網路智慧編排」這個領域的獨特性與創新性。

2.1 AI/ML 在蜂巢式網路中的應用現狀

- **趨勢：** 作者首先承認 ML/AI 技術在網路自動化中的重要性日益增加。
- **具體應用：** 文獻中已經有大量研究將深度學習 (Deep Learning) 和深度強化學習 (DRL) 應用於各種網路任務，包括：
 - 預測網路負載與流量。
 - 流量分類 (Traffic Classification)。
 - 波束對準 (Beam Alignment)。
 - 無線電資源分配 (Radio Resource Allocation)。
 - 網路切片 (Network Slicing)。

- **結論：** 這些技術是 Open RAN 生態系統的基礎，但現有研究大多只關注「模型本身的設計」，而忽略了「如何將這些模型有效地部署到網路中」。

2.2 資源編排 vs. 智慧編排 (Resource vs. Intelligence Orchestration)

這一部分是作者用來區分 OrchestRAN 與傳統編排器的關鍵。作者列舉了多項關於「資源編排」的研究，並指出它們的不足：

- Ayala-Romero et al. [11, 29]：提出了基於貝氏學習 (Bayesian learning) 和 DRL 的框架，用於虛擬化 RAN (vRAN) 中的計算與無線電資源編排，以滿足服務層級協議 (SLAs)。
- Singh et al. [30] (GreenRAN)：專注於節能的編排，根據資源可用性來分配 RAN 組件。
- Chatterjee et al. [31]：針對 5G 應用的無線電資源編排，動態重新分配網路切片以避免效率低落。
- Morais et al. [32] & Matoussi et al. [33]：研究如何最佳化地放置 RAN 組件 (如虛擬網路功能 vNFs)，以最小化計算與能源消耗。
- **作者的批判：** 這些工作雖然都涉及「編排」，但它們編排的對象是 **底層資源** (如 CPU、頻寬、能源) 或 **網路功能**。這與 OrchestRAN 專注於編排「**網路智慧**」(即 **AI 模型本身**) 是本質上不同的問題。

2.3 ML/AI 模型的編排 (Orchestrating ML/AI models)

這部分討論了與 OrchestRAN 最接近的相關工作：

- Baranda et al. [34, 35]：展示了在 5Growth 管理與編排 (MANO) 平台中自動部署模型的架構，主要側重於服務管理層面。
- Salem et al. [20]：這是與本論文最接近的工作。它提出了一個編排器來選擇和實例化「推論模型」，以在準確度與延遲之間取得平衡。
 - **不足之處：** 作者指出 [20] 並非針對 O-RAN 系統設計，且僅關注雲端應用的「推論」任務。

2.4 OrchestRAN 的獨特貢獻 (Differentiation)

在章節的最後，作者總結了 OrchestRAN 與上述工作的四大關鍵差異，確立了其學術貢獻：

1. **專注於 Open RAN 架構**：專為 O-RAN 規範設計，利用 RIC、xApps 和 rApps。
2. **推論與控制並重**：不僅支援推論模型，還支援執行閉迴路控制 (Control loops) 的模型實例化。
3. **模型共享機制 (Model Sharing)**：引入了創新的模型共享功能，允許多個請求共用同一個 AI 模型實例，從而高效地重用稀缺的網路資源（如邊緣節點的運算能力）。
4. **大規模原型驗證**：這是第一個針對 O-RAN 架構的網路智慧編排系統的大規模展示（在 Colosseum 上驗證）。

▼ 3. O-RAN: A PRIMER

關於 O-RAN 架構的技術導讀。由於 OrchestRAN 是建立在 O-RAN 規範之上的系統，這一章詳細介紹了必要的背景知識，解釋了網路功能如何被拆分、控制與管理。

3.1 網路功能的解構與拆分 (Disaggregation & Functional Split)

O-RAN 採用了 **7-2x 功能拆分 (7-2x functional split)**，這是 3GPP 7-2 split 的擴展版本。在這種架構下，傳統基地台的功能被拆解並分配到三個主要實體：

- **Radio Units (RUs)**：
 - 負責執行底層的實體層 (Lower Physical Layer) 功能。
 - 透過 **Open Fronthaul** 介面與 DU 進行通訊。
- **Distributed Units (DUs)**：
 - 負責執行高層實體層 (Higher Physical Layer) 與 MAC 層的功能。
- **Central Units (CUs)**：
 - 負責執行剩餘的協定堆疊功能（如 RLC, PDCP 等）。
 - CU 進一步被拆分為兩個實體：**控制面 (Control Plane, CP)** 與 **用戶面 (User Plane, UP)**，兩者之間透過 **E1 介面** 連接。
 - CU 與 DU 之間則是透過 **F1 介面** 連接。

這些元件運行在通用的「白盒」(White-box) 硬體上，並透過開放的標準化介面連接，這打破了傳統設備商的鎖定 (Vendor lock-in)，實現了多供應商的互通性。

3.2 O-RAN 的核心創新：RIC (RAN Intelligent Controller)

除了硬體解構，O-RAN 最主要的創新在於引入了 **non-RT** 和 **near-RT** 兩個層級的智慧控制器。它們透過發布-訂閱模型 (Publish-subscribe model) 收集統計數據，並透過 O1 和 E2 等介面進行控制。

- **near-RT RIC (近即時 RIC)：**
 - **位置：** 位於 RAN 邊緣附近。
 - **時間尺度：** 處理 **10 毫秒 (ms)** 到 **1 秒 (s)** 之間的操作。
 - **功能：** 透過 **xApps** 執行閉迴路控制 (Closed-loop control)。
 - **應用範例：** 負載平衡 (Load balancing)、換手程序 (Handover)、排程 (Scheduling) 和 RAN 切片策略 (RAN slicing)。
 - **介面：** 使用 **E2 介面** 與 CU/DU 互動。
- **non-RT RIC (非即時 RIC)：**
 - **位置：** 運行在 **服務管理與編排 (SMO)** 框架內 (例如 ONAP)。
 - **時間尺度：** 處理 **大於 1 秒** 的操作。
 - **功能：** 負責訓練 ML/AI 模型，並透過 **A1 介面** 將模型與控制策略部署到 near-RT RIC。
 - **應用：** 執行第三方應用程式，稱為 **rApps**。

3.3 即時控制與 dApps (Real-time Control & dApps)

- **TTI 等級控制：** O-RAN 規範也設想了將 ML/AI 模型直接實例化在 CU 和 DU 上，以執行 **RT-TTI (Transmission Time Interval)** 等級的控制迴路，其運作時間尺度在 **10 毫秒** 左右。
- **dApps 的定義：** 雖然這部分功能留待未來的 O-RAN 擴展，但 OrchestRAN 已經原生設計支援這種控制迴路。為了避免混淆，論文將這類直接在 CU/DU 上運行的即時應用程式定義為 **dApps**。

▼ 4. ORCHESTRAN

詳細介紹了 **OrchestRAN** 系統的具體設計、架構流程以及核心模組的運作機制。這章是整套系統的「實作藍圖」，解釋了系統如何將抽象的網路需求轉化為實際執行的 AI 應用程式。

OrchestRAN 設計為一個運行在 **non-RT RIC** 上的 **rApp**，其運作流程與組件如下：

4.1 系統總體運作流程 (System Workflow)

整個系統的運作分為四個步驟：

- **步驟 I (意圖提交)：** 網路營運商 (NOs) 提交控制請求 (Intent)，指定想要部署的功能（如網路切片、波束成形）、目標位置（如區域 X）以及時間要求。
- **步驟 II (請求收集)：** **Request Collector** 模組收集這些請求。
- **步驟 III (編排計算)：** **Orchestration Engine** 作為大腦，結合 **ML/AI Catalog**（模型庫）與 **Infrastructure Abstraction**（網路狀態），計算出最佳的編排策略（用哪個模型？放在哪裡？）。
- **步驟 IV (實例化與執行)：** 系統自動建立容器 (Containers)，將模型封裝成 O-RAN 應用程式 (xApp, rApp, dApp)，派發到選定的節點並開始執行。

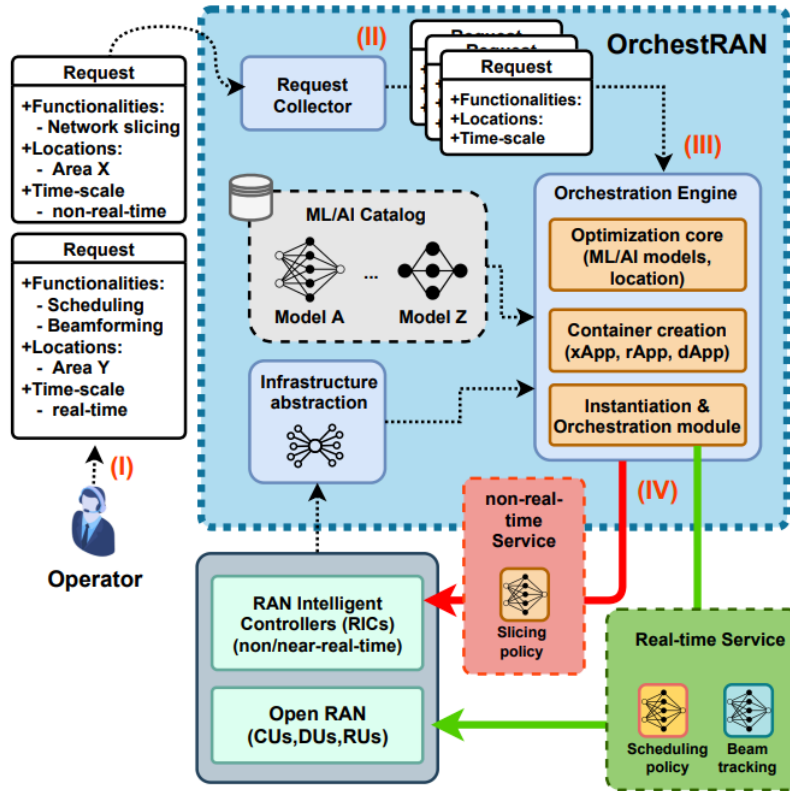


Figure 2. System design of OrchestRAN and main procedures.

4.2 四大核心模組詳解

A. 基礎設施抽象模組 (The Infrastructure Abstraction Module)

這個模組負責將物理網路世界轉化為系統可理解的數學模型：

- **邏輯分組**：將網路節點分為五類：non-RT RICs, near-RT RICs, CUs, DUs, RUs。
- **樹狀結構 (Tree Graph)**：使用無向圖來表示網路拓撲。**non-RT RIC** 是樹的根 (**Root**)，邊緣節點 (RU/DU/CU) 是葉子。
- **連接性 ($c_{d',d''}$)**：定義節點之間是否可達。這決定了數據能否從 A 點傳輸到 B 點 (例如，若兩個節點位於樹的不同分支且無橫向連接，則視為不可達)。
- **資源定義 (ρ_d^ξ)**：定義每個節點 d 擁有的資源量 (如 CPU 核心數、GPU、記憶體等)。

B. ML/AI 目錄 (The ML/AI Catalog)

這是系統的「軍火庫」，儲存所有預訓練好的 AI/ML 模型。每個模型 m 都有詳細的屬性標籤：

- **功能 (\mathcal{F}_m)**：模型能提供什麼功能（如流量預測、排程控制）。
- **資源需求 (ρ_m^ξ)**：執行此模型需要多少 CPU/記憶體。
- **輸入類型 (t_m^{IN})**：模型需要什麼格式的數據（例如 IQ 樣本、Throughput 統計）。這點至關重要，因為某些數據（如 IQ 樣本）只能在 RU 取得，若傳輸到雲端會造成極大延遲，限制了模型的部署位置。
- **適用性指標 ($\beta_{m,f,d}$)**：一個 0 到 1 的分數，表示模型 m 在節點 d 執行功能 f 的合適程度。
- **性能分數 ($\gamma_{m,f}$)**：模型的效能指標（如準確率、MSE）。
- **容量限制 ($C_{m,d}$)**：根據節點資源，計算該節點最多能同時執行多少個此模型的實例。

C. 請求收集器 (Request Collector)

負責將營運商的意圖標準化為數學 Tuple ($\mathcal{F}_i, \pi_i, \delta_i, \mathcal{D}_i^{IN}$)，以便系統處理：

- **功能與位置 (\mathcal{F}_i)**：指定在「哪些節點」需要「什麼功能」。
- **性能要求 (π_i)**：設定最低可接受的性能門檻（例如準確率需 $> 90\%$ ）。
- **時間要求 (δ_i)**：這是最關鍵的約束條件。 $\delta_{i,f,d}$ 定義了請求 i 對功能 f 的最大容許延遲。這直接限制了模型能跑多遠（若數據源在 RU，但延遲要求極低，模型就不能跑在 RIC，必須下放到邊緣）。
- **數據來源 (\mathcal{D}_i^{IN})**：指定模型必須使用哪些節點產生的數據。

D. 編排引擎 (The Orchestration Engine)

這是系統的核心決策與執行單元：

- **決策 (Optimization)**：根據上述所有資訊，解出最佳的編排策略（即解決第五、六章提到的 NP-hard 問題），決定哪些模型被選中以及它們該放在哪裡。
- **容器建立 (Container Creation)**：
 - 一旦策略確定，引擎會自動將選定的模型封裝進 Docker 容器。

- **連接器 (Connector)：** 每個容器都會注入一個特定的「連接器」函式庫。這個連接器負責與底層平台（如 near-RT RIC 或 DU）溝通，負責從指定來源收集數據並發送控制指令。
- **派發與實例化 (Dispatchment & Instantiation)：**
 - 將打包好的容器（xApps, rApps, 或 dApps）傳送到目標節點執行。
 - **自動化介接：** 例如，若是 xApp，它會自動發送 E2 訂閱請求給目標節點以獲取數據，完全無需人工介入。

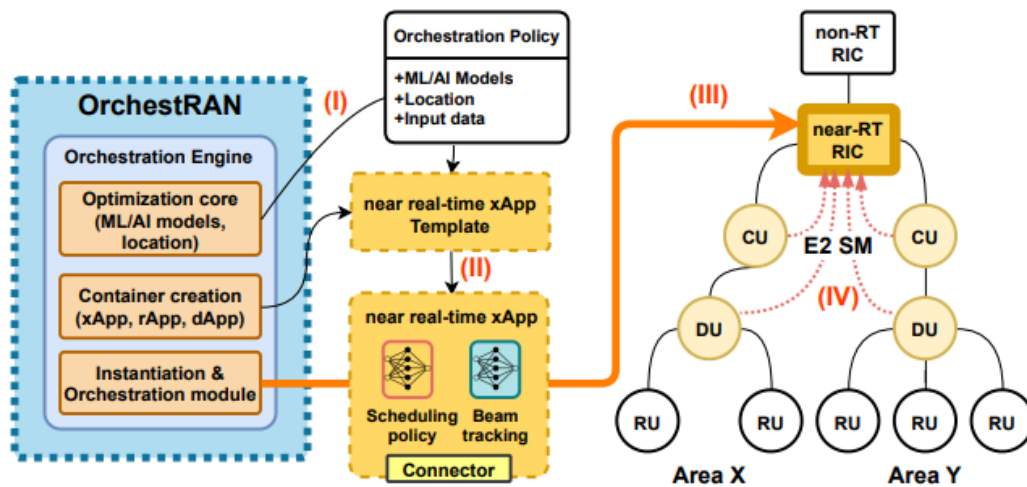


Figure 3. An example of creation and dispatchment of an xApp on the near-RT RIC via OrchestRAN.

▼ 5. THE ORCHESTRATION PROBLEM

這一章的目的是將前面提到的「自動化編排」概念，轉化為一個可以被電腦求解的 **數學優化問題 (Optimization Problem)**。

簡單來說，這一章告訴系統：「在面對有限的資源、嚴格的時間限制和多個營運商請求時，該如何用數學公式算出最佳決策？」

5.1 兩個核心前提概念 (Preliminaries)

在列出公式之前，作者先定義了 Open RAN 環境下的兩個關鍵操作特性，這是模型設計的基礎：

A. 功能外包 (Functionality Outsourcing)

- **概念：** 一個原本預定在節點 d 執行的功能，不一定要在 d 上跑，可以被「外包」到另一個節點 d' 執行。
- **條件：** 1. 兩個節點之間必須有物理連接 ($c_{d,d'} = 1$) 2. 承接的節點 (d') 必須能存取到所需的輸入數據。 3. 滿足資源與時間限制。
- **意義：** 這賦予了系統靈活性，例如將運算從資源受限的 RU 轉移到資源豐富的 DU 或 CU。

B. 模型共享 (Model Sharing)

- **概念：** 這是 OrchestRAN 節省資源的關鍵。如果有多個請求 (Requests) 都需要在同一組節點上執行相同的功能（例如：三個切片都需要預測流量），系統不需要啟動三個一樣模型。
- **作法：** 系統可以只部署 **一個模型實例 (Single Instance)**，讓它同時服務所有請求。
- **圖解 (Fig. 4)：** 論文用圖 4 舉例，如果不共享需要 4 個模型，共享後只需要 3 個，這對於資源極度受限的邊緣節點（如 DU/RU）至關重要。

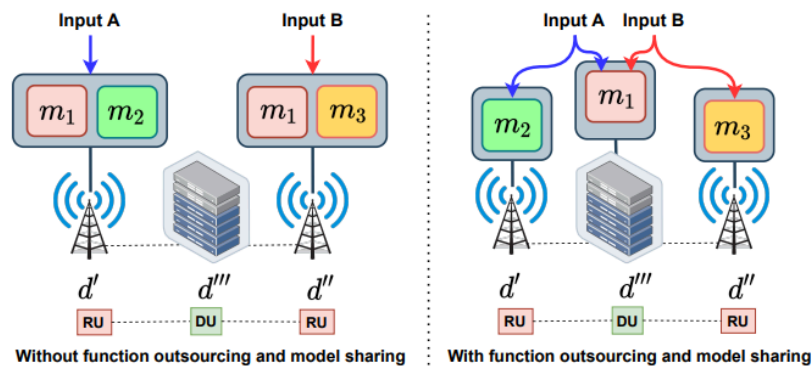


Figure 4. Example of function outsourcing and model sharing in Open RAN.

5.2 數學問題建模 (Formulating the Orchestration Problem)

這一節建立了二元整數線性規劃 (BILP) 模型。

核心決策變數 (The Decision Variable)

作者定義了一個超級複雜的二元變數 x ，它是所有決策的核心：

$$x_{m,k,d'}^{i,f,d} \in \{0, 1\}$$

這個變數的意思是：「針對請求 i 在節點 d 需要的功能 f ，我們是否決定用部署在節點 d' 上的模型 m 的第 k 個實例來提供？」。

- 如果是 1，代表選中這個組合。
- 如果是 0，代表沒選中。

四大約束條件 (Constraints)

為了讓這個決策變數符合現實物理限制，作者列出了四條規則：

A. 衝突避免 (Conflict Avoidance) - 公式 (1)

- **規則：** 一個蘿蔔一個坑。對於每個請求指定的某個功能，系統最多只能指派一個模型來負責。
- **目的：** 防止多個 AI 模型同時控制同一個網路參數（例如兩個模型同時想調整同一個天線的角度），造成系統混亂。
- **滿足指標 (y_i)：** 只有當一個請求 i 裡面的「所有」所需功能都被滿足時，該請求才算成功 ($y_i = 1$)。

$$\sum_{m \in \mathcal{M}} \sigma_{m,f} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{m,d'}} c_{d,d'} x_{m,k,d'}^{i,f,d} = y_i \tau_{i,f,d}$$

B. 滿足性能要求 (Quality of Models) - 公式 (2)

- **規則：** 選中的模型不只要能跑，還要跑得好。
- **公式邏輯：** 模型的基準性能分數 (γ) \times 在該位置的適用性 (β) 必須 \geq 營運商設定的最低門檻 (π)。
- **例子：** 如果一個波束成形模型需要 IQ 樣本，但被部署在拿不到 IQ 樣本的雲端，其適用性 β 會很低，導致總分不達標，因此這個約束會禁止這種部署。

$$\chi_{i,f,d} \sum_{m \in \mathcal{M}} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{m,d'}} c_{d,d'} x_{m,k,d'}^{i,f,d} A_{m,f,d} \geq \chi_{i,f,d} y_i \pi_{i,f,d}$$

C. 時間尺度限制 (Control-loop Time-scales) - 公式 (3), (4), (5) 這是 Open RAN 中最嚴格的物理限制。

- **總延遲公式：** 數據收集時間 (Δ) + 模型執行時間 (Δ^{EXEC}) \leq 最大容許延遲 (δ)
。
- **數據收集時間 (Eq. 3)：** 計算將數據從源頭傳到模型執行點需要多久。這取決於數據量大小 (s)、網路頻寬 (b) 和傳輸距離延遲 (T) 。
- **模型執行時間 (Eq. 4)：** 模型運算本身需要的時間 。
- **意義：** 這條公式解釋了為什麼某些模型**必須**放在邊緣。如果數據量很大 (如 IQ Data)，頻寬不夠大，傳輸時間就會爆表。為了滿足 $\leq \delta$ ，模型必須移動到離數據源很近的地方 (如 DU 或 RU) 執行。

$$\Delta_{i,f,d}(\mathbf{x}) = \sum_{m \in \mathcal{M}} \sigma_{m,f} \sum_{d' \in \mathcal{D}} \sum_{k=1}^{C_{m,d'}} x_{m,k,d'}^{i,f,d} \sum_{d'' \in \mathcal{D}_{i,f,d}^{IN}} c_{d',d''} \Theta_{m,d',d''}^{i,f,d} \quad (3)$$

$$\Delta_{i,f,d}^{EXEC}(\mathbf{x}) = \sum_{m \in \mathcal{M}} \sigma_{m,f} \sum_{d' \in \mathcal{D}} T_{m,d'}^{exec} \sum_{k=1}^{C_{m,d'}} x_{m,k,d'}^{i,f,d} \quad (4)$$

$$\Delta_{i,f,d}(\mathbf{x}) + \Delta_{i,f,d}^{EXEC}(\mathbf{x}) \leq \delta_{i,f,d} \tau_{i,f,d} \quad (5)$$

D. 避免資源超額 (Resource Constraints) - 公式 (6), (7), (8), (9)

- **規則：** 一個節點上所有跑的模型，加起來不能吃光該節點的 CPU/記憶體 。
- **模型共享的數學實現 (Big-M)：** 這裡使用了 **Big-M 法** 來處理變數 $z_{m,k,d}$ 。
 z 代表「某個模型實例是否被啟動」。
即使有 100 個請求 (x) 都指向同一個模型實例 (k)，只要這個實例被啟動一次 (

$z = 1$)，它就只佔用一份資源。這就是數學上如何表達「模型共享」所帶來的資源節省。

$$\sum_{m \in \mathcal{M}} \rho_m^\xi \sum_{k=1}^{C_{md}} z_{m,k,d} \leq \rho_d^\xi \quad (6)$$

$$n_{m,k,d} = \sum_{i \in \mathcal{I}} \sum_{f \in \mathcal{F}_i} \sum_{d' \in \mathcal{D}} x_{m,k,d}^{i,f,d'} \quad (7)$$

$$n_{m,k,d} \geq 1 - M(1 - z_{m,k,d}) \quad (8)$$

$$n_{m,k,d} \leq M z_{m,k,d} \quad (9)$$

5.3 優化目標 (Problem Formulation)

最終，OrchestRAN 的目標是要解出上述變數 x, y, z ，使得：

$$\max \sum_{i \in \mathcal{I}} y_i v_i$$

- **目標：** 最大化「被滿足的請求的總價值」。
- 通常假設每個請求價值為 1，即最大化**被接受的請求數量**。

5.4 複雜度分析 (NP-hardness)

這一章的最後（以及第六章的開頭）指出，上述定義的問題是一個**二元整數線性規劃 (BILP)** 問題。

- **結論：** 這個問題是 **NP-hard** 的。
- **證明：** 作者透過將經典的 **3-SAT 問題** 歸約到這個編排問題，證明了其複雜度極高。
- **影響：** 這意味著如果網路很大（變數很多），用傳統方法暴力求解會算到天荒地老，這為第六章提出的「啟發式加速演算法」埋下了伏筆。

▼ 6. SOLVING THE ORCHESTRATION PROBLEM

雖然第五章建立了完美的數學模型，但作者發現直接求解這個模型在計算上是「不可行」的，因為它的複雜度太高。因此，第六章提出了**兩大核心策略**來降低複雜度，確保系統能在現實世界的短時間內算出結果。

以下是第六章的詳細拆解：

6.1 問題的本質：維度詛咒 (The Curse of Dimensionality)

- **數學屬性：** 第五章定義的問題屬於 **二元整數線性規劃 (Binary Integer Linear Programming, BILP)**。
- **計算難度：** 這是一個 **NP-hard** 問題。
- **變數爆炸：**
 - 優化變數的總數量 (N_{OPT}) 與網路規模呈**平方增長**。
 - 公式為： $|x|_{[cite_start]} = \mathcal{O}(IFD^2 MC_{max})$ 。其中 D 是節點數量。
 - **具體例子：** 即使是一個只有 **20 個節點** 的小型網路，變數數量就可能高達 **100 萬個 (10^6)**。
- **後果：** 使用標準的求解器（如 CPLEX, Gurobi 基於 Branch-and-Bound 技術）會因為搜尋空間太大而無法在合理時間內算出結果，這對於需要快速反應的網路控制來說是不可接受的。

6.2 第一大策略：變數縮減 (Variable Reduction)

為了應對變數爆炸，作者利用問題本身的結構特性，設計了兩種**修剪 (Pruning)** 技術。這兩種技術的目標是找出那些「註定為 0」(Inactive) 的變數並直接移除，從而縮小搜尋空間，且**保證不會影響解的最優性 (Optimality)**。

A. 功能感知修剪 (Function-aware Pruning, FP)

- **邏輯：** 根據「需求」和「能力」來刪除無效變數。
- **操作：** 移除集合 x_{-}^{FP} 中的變數，滿足以下任一條件：

- **需求不符**：請求 i 根本不需要在節點 d 執行功能 f ($\tau_{i,f,d} = 0$)。
- **能力不足**：模型 m 根本不具備功能 f 的能力 ($\sigma_{m,f} = 0$)。
- **結果**：這些變數永遠不可能被選中（值必為 0），直接從方程式中刪除。

B. 架構感知修剪 (Architecture-aware Pruning, AP)

- **邏輯**：根據「物理連接性」來刪除無效變數。
- **背景**：模型執行時需要數據輸入。如果模型放在節點 d' ，但數據源在節點 d ，且這兩點之間**沒有物理連接** ($c_{d,d'} = 0$)，那這個部署在物理上就是不可能的。
- **操作**：移除集合 x_{-}^{AP} 中所有涉及「不可達路徑」的變數。
- **結果**：排除那些因為網路拓撲限制而無法成立的部署選項。

經過 FP 和 AP 處理後，問題被映射到一個低維度空間 \bar{x} ，但仍然能求得與原問題一樣的完美最佳解 9。

6.3 第二大策略：圖樹分支 (Graph Tree Branching)

即使經過修剪，對於擁有數千個節點的超大型網路，變數依然太多 (D^2 的威力)。因此，作者提出了一種基於**分治法 (Divide and Conquer)** 的啟發式算法。

A. 設計直覺

- **網路結構**：O-RAN 網路是大樹狀結構，但運作上是「叢集式」(Cluster-based) 的。
- **獨立性**：每個 **near-RT RIC** 只控制自己轄下的一組 CU/DU/RU。不同 RIC 的轄區之間幾乎沒有互動。
- **想法**：既然不同區域互不影響，何不把它們切開來分開算？

B. 演算法步驟

- **步驟 I (切分 - Decomposition)**：
 - 將整個巨大的網路樹狀圖切分為多個 **子樹 (Subtrees, \mathcal{D}_c)**。
 - 每一個子樹代表一個 Cluster（即一個 near-RT RIC 及其下屬的所有節點）。
- **步驟 II (分類 - Classification)**：

- 檢查所有請求，找出那些「只涉及特定子樹」的請求。
- 建立子請求集合 \mathcal{I}_c ，這些請求只與 Cluster c 有關。
- **步驟 III (並行求解 - Parallel Solving) :**
 - 針對每個 Cluster 獨立求解上述的 BILP 問題。
 - 因為每個子問題的節點數 (D) 大幅減少，且可以**並行運算 (In Parallel)**，計算速度極快。

C. 代價與優勢

- **代價 (Trade-off) :** 這種方法可能會導致**部分滿足 (Partially Satisfied)** 的請求。如果一個請求橫跨了兩個不同的 Clusters (例如需要同時控制兩個不同 RIC 下的基地台)，這個請求可能會被拆散，導致違反第五章的完整性約束 (Constraint 1)。
- **優勢 (Advantage) :**
 - **極致的速度 :** 實驗顯示，這種方法可以在 **0.1 秒** 內解決包含 **2000 個節點** 的網路編排問題。
 - **實用性 :** 對於大多數現實世界的 O-RAN 應用 (通常是區域性的)，這種解法在效能與速度之間取得了極佳的平衡。

D. 總結

第六章的核心貢獻在於證明了 OrchestRAN 不僅僅是一個數學構想，而是具備**可實作性 (Implementability)** 的系統。透過 **FP/AP 修剪技術** (保證最優) 和 **Graph Tree Branching** (保證速度)，它解決了大規模網路中的計算瓶頸。

▼ 7. NUMERICAL EVALUATION

主要透過 **MATLAB 模擬** 來量化評估 OrchestRAN 在大規模網路場景下的性能。這一章的重點不在於真實訊號傳輸 (那是第八章的事)，而在於驗證 OrchestRAN 的 **演算法效率、資源分配策略** 以及 **編排決策的合理性**。

7.1 實驗環境與參數設置 (Simulation Setup)

為了進行公平且大規模的測試，作者建立了一個標準化的模擬環境：

- **模擬工具 :** 使用 MATLAB 開發，並調用 CPLEX 求解器來執行優化運算。

- **網路拓撲 (Network Topology) :**
 - 一個標準的單一網域包含：**1 個 non-RT RIC、4 個 near-RT RICs、10 個 CUs、30 個 DUs 和 90 個 RUs。**
 - 每次模擬的樹狀結構是隨機生成的，但節點總數固定。
- **請求生成 (Requests) :** 每次模擬由營運商提交 **20 個隨機生成的請求**。
- **資源假設 (Resources) :**
 - **計算資源 (ρ) :** 以 **CPU 核心數** 代表。資源分佈呈現「核心多、邊緣少」的特徵:
 - non-RT RIC: **128 核**
 - near-RT RIC: **8 核**
 - CU: **4 核**
 - DU: **2 核**
 - RU: **1 核**
 - **網路頻寬與延遲 :** 越靠近核心頻寬越大、延遲越高；越靠近邊緣頻寬越小、延遲越低（例如 DU-RU 之間延遲僅 1ms，頻寬 20Gbps）。
- **模型庫 (Catalog) :** 包含 13 個模型，提供 7 種功能。其中 10 個模型使用一般指標數據（小數據量 100 bytes），3 個模型使用 IQ 樣本（大數據量 1000 bytes）。

7.2 關鍵結果一：計算複雜度與可擴展性 (Computational Complexity)

這部分驗證了第六章提出的加速演算法是否有效（對應 **Figure 5**）。

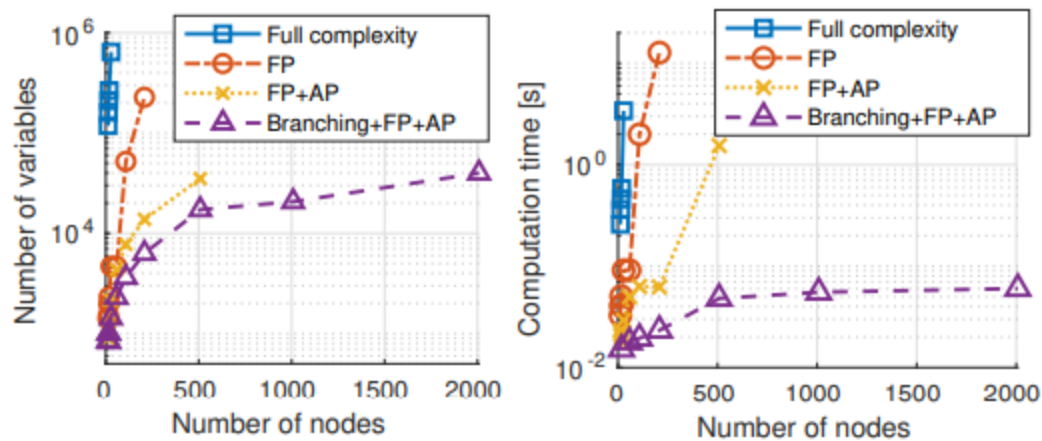


Figure 5. Number of variables and computation time for different network size.

- **變數爆炸**：隨著網路規模增加，如果不優化，變數數量會呈指數上升（Full complexity 曲線），導致計算時間過長。
- **修剪技術 (Pruning) 的效果**：
 - 使用了 **FP (功能感知)** 和 **AP (架構感知)** 修剪後，變數數量顯著下降，計算時間大幅縮短。
 - 對於 500 個節點的網路，計算時間約為 2 秒。
- **分支演算法 (Branching) 的威力**：
 - 最關鍵的發現是使用了 **Graph Tree Branching** 算法後，即使網路規模擴大到 **2000 個節點**，OrchestRAN 仍能在 **0.1 秒 (0.1 s)** 內算出結果。
 - 這證明了該系統具備極佳的 **可擴展性 (Scalability)**，足以應對真實世界的大規模部署。

7.3 關鍵結果二：請求接受率 (Acceptance Ratio)

這部分分析了不同嚴格程度的請求，其成功被系統接受的機率（對應 Figure 6）。

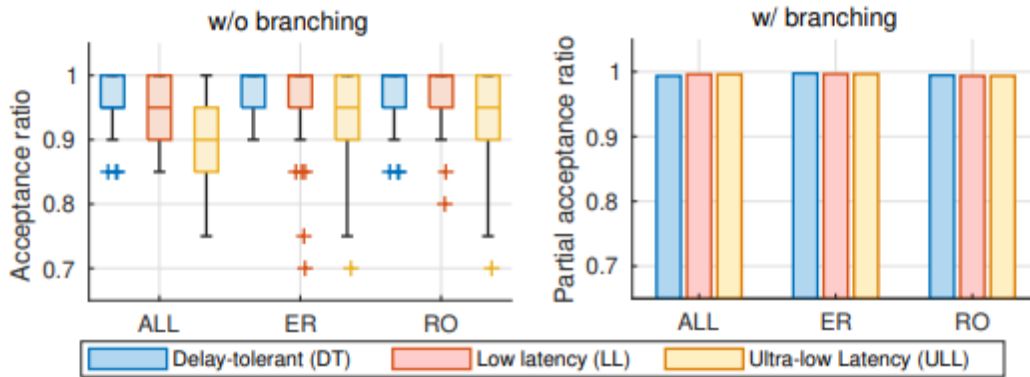


Figure 6. (Left) Ratio of accepted requests w/ model sharing but w/o branching; (Right) Ratio of partially accepted requests w/ model sharing and branching.

- 測試案例 (Table II) :
 - **DT (Delay-Tolerant)** : 延遲容忍度高 (>1s)。
 - **LL (Low Latency)** : 低延遲 (<1s)。
 - **ULL (Ultra-Low Latency)** : 超低延遲 (TTI 等級, <0.01s) 。
- 接受率趨勢 :
 - **DT 請求** : 接受率最高，平均約 **95%** 。
 - **ULL 請求** : 接受率下降至約 **70%** 。
- 原因分析 : ULL 請求通常需要 TTI 等級的控制，這迫使模型必須在 **DU 或 RU** 執行。然而，從資源假設可知，DU/RU 只有 1-2 個 CPU 核心，資源極度受限，無法同時容納大量模型，導致部分請求被拒絕。
- 部分接受 (Partial Acceptance) : 使用分支演算法時，約 **99%** 的請求能被「部分滿足」（例如在某些 Cluster 成功部署，在某些失敗），這比直接拒絕整個請求更具彈性。

7.4 關鍵結果三：模型共享的優勢 (Advantages of Model Sharing)

這部分量化了 OrchestRAN 核心功能「模型共享」的效益（對應 **Figure 7 & 8**）。

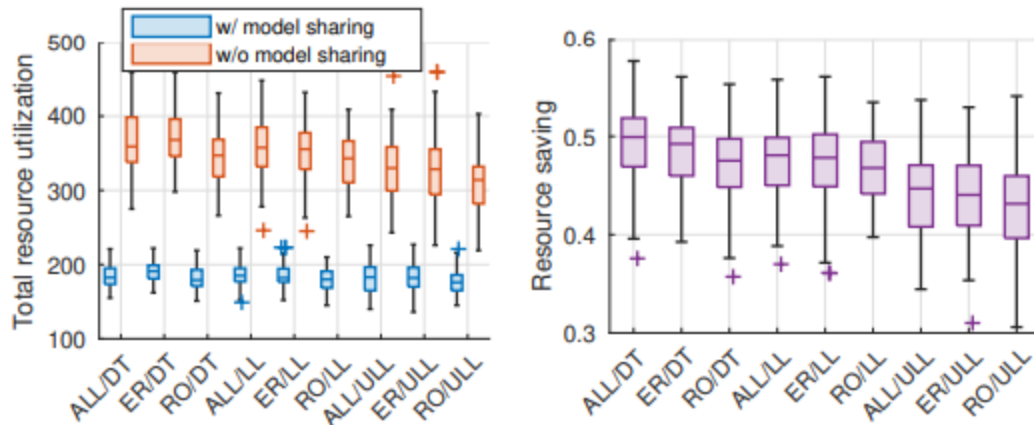


Figure 7. Resource utilization and saving with and without model sharing.

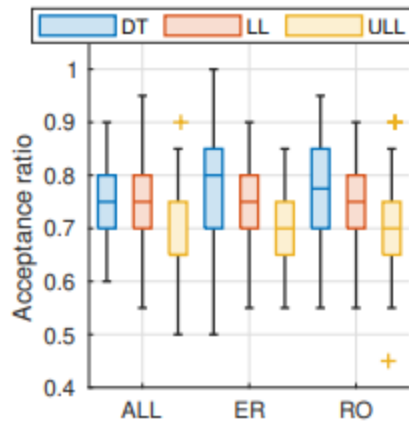


Figure 8. Percentage of accepted requests w/o model sharing for different cases.

- **資源節省 (Resource Saving) :**
 - 啟用模型共享後，總資源使用量大約是未啟用時的一半 (**uses 2x less resources**)。
 - 這對於資源稀缺的邊緣節點 (DU/RU) 至關重要。
- **接受率提升 :**
 - 如果不開啟模型共享，請求接受率會掉到 **70%** 左右。

- 開啟模型共享後，即便在資源緊張的情況下，接受率也能維持在 **90%** 以上。
- **結論：** 模型共享不僅省資源，還直接提升了網路服務更多營運商請求的能力。

7.5 關鍵結果四：智慧的自動分佈 (Distribution of Intelligence)

這部分展示了 OrchestRAN 到底把模型「放」在哪裡（對應 **Figure 9**）。

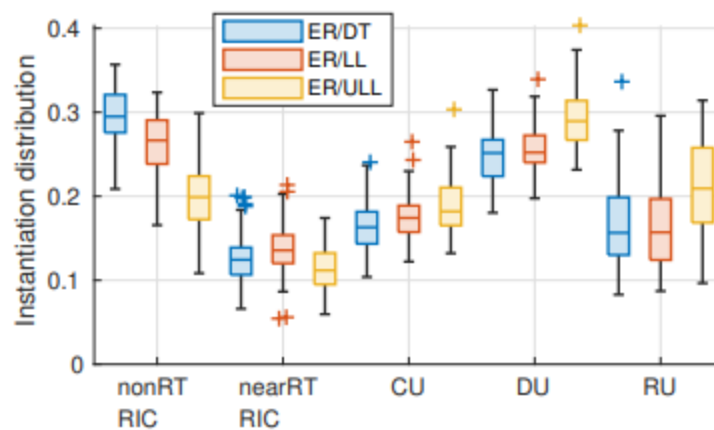


Figure 9. Distribution of model instantiation for different cases.

- **寬鬆時限 (DT)：** 當延遲要求不嚴格時，約 **45%** 的模型被部署在 **RICs**（雲端/中心化位置），利用那裡豐富的運算資源（128核）。
- **嚴格時限 (LL/ULL)：** 當延遲要求嚴格時，系統被迫將約 **70%** 的模型推向 **CUs, DUs, 和 RUs**（邊緣位置）。
- **意義：** 這證明了 OrchestRAN 能夠根據物理限制（延遲、頻寬）**智慧地 (Intelligently)** 決定模型位置，自動實現了「邊緣運算」的願景，而無需人工介入。

▼ 8. PROTOTYPE AND EXPERIMENTAL EVALUATION

OrchestRAN 的原型實作細節以及在大型模擬器上的實驗結果。這一章的重點在於驗證系統在真實（模擬）環境下的運作能力、資源調度效率以及對網路效能的實際影響。

8.1 原型實作平台 (Prototype Implementation)

為了證明 OrchestRAN 的有效性，作者構建了一個大規模的實驗平台：

- **基礎設施：** 使用 **Colosseum**（世界上最大的無線網路模擬器，具備硬體迴路 Hardware-in-the-Loop）上的 **CoIO-RAN** 平台。
 - 包含 128 個計算伺服器 (Standard Radio Nodes, SRNs)，控制 USRP X310 軟體定義無線電 (SDR)。
 - 使用 MCHEM 模擬器來重現真實且時變的無線通道特性（如路徑損耗、多徑效應）。
- **軟體環境：** 利用 **SCOPE** 框架來實例化軟體化的蜂巢式網路。
 - **規模：** 模擬羅馬市中心場景，包含 **7 個基地台** 和 **42 個用戶設備 (UEs)**（每個基地台 6 個用戶）。
 - **介面：** 基地台透過 **E2 介面** 連接到 O-RAN near-RT RIC。
 - **功能：** SCOPE (基於 srsRAN) 提供了 API，允許 O-RAN 應用程式透過閉迴路控制來重新配置基地台參數（如切片資源、調度策略），並自動生成數據集（如吞吐量、緩衝區大小）。

8.2 網路切片與流量配置 (Network Slices & Traffic)

實驗中設定了三種不同需求的網路切片 (Slices)，每種切片有 2 個用戶，使用 10 MHz 頻寬 (50 PRB)：

- **Slice 0 (eMBB - 增強型行動寬頻)：** 每個用戶請求 **4 Mbps** 的固定位元率 (CBR) 流量。
- **Slice 1 (MTC - 機器型通訊)：** 模擬 IoT 流量，泊松分佈 (Poisson-distributed)，平均速率 **45 kbps**。
- **Slice 2 (URLLC - 超可靠低延遲通訊)：** 模擬關鍵任務流量，泊松分佈，平均速率 **90 kbps**。

8.3 OrchestRAN 系統架構 (System Architecture on Prototype)

- OrchestRAN 運行在一個 Linux 容器 (LXC) 中，每 4 分鐘隨機生成一組新的控制請求。
- **編排引擎**：計算最佳策略，將模型封裝進 O-RAN 應用程式，並派發到節點執行。
- **執行位置**：模型可以作為 **xApps** 在 **near-RT RIC** 執行，或作為 **dApps** (透過 SCOPE) 在 **DU** 執行。

8.4 ML/AI 模型目錄 (ML/AI Catalog)

實驗使用了 4 個預訓練模型 (Table III):

- **M1 & M2 (預測)**：用於預測吞吐量 (Throughput) 和傳輸緩衝區大小 (Buffer size)。
- **M3 (分散式 DRL 控制)**：
 - 包含 **3 個獨立的 DRL agents** (基於 PPO 演算法)。
 - 每個 agent 只控制一個切片的調度策略 (**Scheduling**)。
 - **獎勵目標**：Slice 0 最大化吞吐量；Slice 1 最大化傳輸封包數；Slice 2 最大化 PRB 比率 (已分配/請求資源)。
- **M4 (集中式 DRL 控制)**：
 - **單一 DRL agent**，同時控制 **調度 (Scheduling)** 和 **RAN 切片資源分配 (RAN slicing)**。
 - **聯合優化目標**：同時最大化 Slice 0 吞吐量、Slice 1 封包數，並**最小化** Slice 2 的緩衝區大小。

8.5 實驗配置與硬體假設 (Experimental Configurations)

考慮了三種不同的資源配置場景，near-RT RIC 皆擁有 50 個核心：

- **RIC only**：模型只能在 near-RT RIC (作為 xApps) 執行。
- **RIC + lightweight DU**：DU 配備 **2 個核心** (可並發執行 2 個 dApps)。

- **RIC + powerful DU**：DU 配備 8 個核心。

8.6 實驗結果分析 (Experimental Results)

A. 模型實例化位置 (Instantiation Probability) - Fig. 11 (Left)

- **RIC only**：100% 模型在 RIC 執行。
- **RIC + DU (Lightweight/Powerful)**：當允許在 DU 執行時，只有約 25% 的模型留在 RIC，其餘 75% 自動被派發到 DU 作為 dApps 執行。

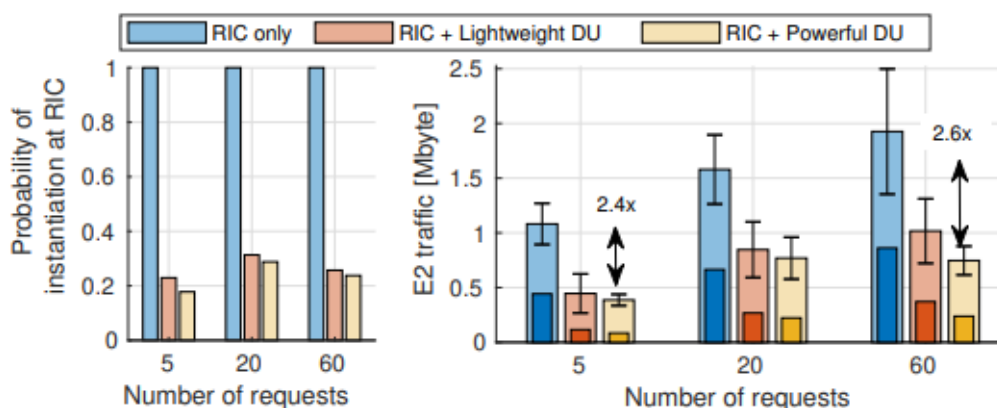


Figure 11. (Left) Probability of instantiating O-RAN applications at the near-RT RIC; (Right) Traffic over O-RAN E2 interface for different configurations. Dark bars represent traffic related to payload only.

B. E2 介面流量開銷 (E2 Traffic Overhead) - Fig. 11 (Right)

- **流量組成**：E2 流量包含初始訂閱、指標回報 (Metrics) 和控制訊息。約 40% 是有效負載 (Payload)，60% 是開銷。
- **關鍵發現**：將模型作為 dApps 在 DU 執行，相比於 "RIC only" 模式，可以將 E2 介面流量減少高達 2.6 倍。
 - 這是因為在邊緣 (DU) 處理數據，避免了將大量原始指標透過 E2 回傳給 RIC。

C. 對網路效能的影響 (Network Performance Impact) - Fig. 12

- **無縫切換：**實驗展示了 OrchestRAN 可以在 near-RT RIC 和 DU 之間動態激活 M3 和 M4 模型，而不會造成任何服務中斷。
- **M3 vs. M4 效能比較：**
 - **Slice 0 (eMBB)：** M4 因為能控制資源切片 (RAN Slicing)，提供的吞吐量比 M3 高出約 10%。
 - **Slice 1 (MTC)：** M4 的傳輸封包數比 M3 高出約 2 倍。
 - **Slice 2 (URLLC)：**
 - M3 的目標是最大化 PRB 比率，結果 PRB ratio 接近 1，但緩衝區積壓較多。
 - M4 的目標是最小化緩衝區，結果顯示 M4 激活後 緩衝區大小顯著下降 (雖然 PRB ratio 也隨之下降，但延遲性能更好)。

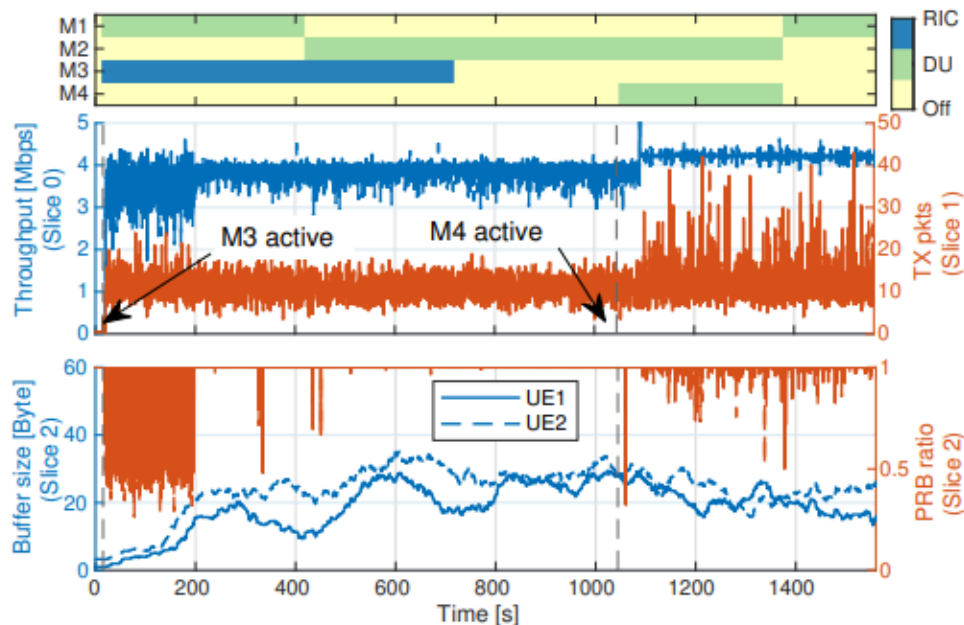


Figure 12. (Top) Dynamic activation of O-RAN applications at near-RT RIC and DU 7; (Center and bottom) Performance comparison for different deployments of O-RAN applications and network slices. Solid lines and dashed lines refer to traffic for $UE_{1,i}$ and $UE_{2,i}$ of Slice i .

第八章透過大規模原型實驗，證實了 OrchestRAN 不僅能智慧地將計算推向邊緣以節省 2.6 倍的控制流量，還能透過動態部署更先進的控制模型 (如 M4) 來顯著提升網路

切片的效能。

▼ 9. CONCLUSIONS

- **提出的框架：**OrchestRAN 是一個針對 Open RAN 的新型網路智慧編排框架，完全符合 O-RAN 規範，利用 xApps 和 rApps 來運作。
- **解決的問題：**為網路營運商 (NOs) 提供自動化工具，解決了如何部署具有不同時間尺度要求的數據驅動模型的問題。
- **技術亮點：**內建了具有不同「最優性/複雜度」權衡的演算法，能同時支援 non-RT、near-RT 和 RT 應用。
- **驗證成果：**
 - 在 Colosseum 模擬器上建立了包含 7 個基地台和 42 個用戶的大規模原型。
 - 實驗證實 OrchestRAN 能無縫地在不同節點實例化應用。
 - 通過將智慧部署在網路邊緣 (Edge)，成功將 O-RAN E2 介面的訊息開銷 **減少了 2.6 倍**。