O-RAN.WG2.AIML-v01.03

Technical Report

O-RAN Working Group 2

AI/ML workflow description and requirements

O-RAN ALLIANCE e.V.
Buschkauler Weg 27, 53347 Alfter, Germany
Register of Associations, Bonn VR 11238
VAT ID DE321720189.

Revision History

Date	Revision	Author	Description
2020.08.31	01.02.00	R. Jana	Clean Baseline doc
2020.08.31	01.02.01	Intel, ATT, CMCC, Altran, Samsung	Adding approved CR INT.AO-2020.07.06-WG2-CR-0001-AIML model termination procedure-v03.docx
2020.10.09	01.02.02	IBM, ZTE, CMCC	IBM.AO-2020.06.05-WG2-CR-0001-AIML-v05.docx
2020.11.29	01.02.03	Intel, Samsung, Amdocs	INT.AO-2020.10.19-WG2-CR-0006-Reinforcement Learning-v02.docx
2020.11.29	01.02.03	Intel, Samsung, Amdocs	INT.AO-2020.10.19-WG2-CR-0007-DS for RL-v04.docx
2021.01.16	01.02.04	IBM	IBM-2020.06.05-WG2-CR-0002-AIML-v11.docx
2021.01.16	01.02.04	IBM	IBM-2020.06.05-WG2-CR-0003-AIML-v08.docx
2021.01.16	01.02.04	NOK	NOK-2020.11.26-WG2-CR-0001-ModelLifecycle-v03.docx
2021.02.23	01.02.04	ZTE, CMCC	ZTE.AO-2020.06.03-WG2-CR-001-AIML-v8.doc
2021.03.11	01.02		Editorial updates for publication
2021.05.09	01.03.01	INT	INT.AO-2020.10.19-WG2-CR-0007-DS for RL-v05.docx
2021.05.09	01.03.01	INT	INT-2021.05.05-WG2-CR-00014-DS for RL in Table 2-v01.docx
2021.06.28	01.03.02	INT	INT-2021.06.28-WG2-CR-0019-Federated learning among Non-RT RIC and Near-RT RICs-v02.docx
2021.06.28	01.03.02	QC	QC-2021.06.28-WG2-CR-0001-Correction on control loops v2.docx
2021.06.28	01.03.02	QC	QC-2021.06.28-WG2-CR-0002-Model performance monitoring v2.docx
2021.07.20	01.03		Editorial updates for publication (final)

Contents

Chapter 1 Introduction

1.1 Scope

This Technical Report (TR) has been produced by O-RAN Alliance. TRs are informative but they can contain potential functional requirements that will feed into Techincal Specifications (TS) eventually.

The contents of the present document are subject to continuing work within O-RAN WG2 and may change following formal O-RAN approval. In the event that O-RAN Alliance decides to modify the contents of the present document, it will be re-released by O-RAN Alliance with an identifying change of release date and an increase in version number as follows:

Release x.y.z

where:

- x the first digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc. (the initial approved document will have x=01).
- y the second digit is incremented when editorial only changes have been incorporated in the document.
- z the third digit included only in working versions of the document indicating incremental changes during the editing process.

The current document addresses the overall architecture and solution for AI/ML related requirements for the use-cases described in O-RAN WG2 UCR doc [O-RAN-WG2.UCR.02.00.00]. The document provides the terminology, workflow, and requirements, related to AI/ML model training, and its distribution and deployment in the Radio Access Network (RAN).

1.2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document in Release 16.

[1] 3GPP TR 21.905: “Vocabulary for 3GPP Specifications”

[2] 3GPP TS 38.401: "NG-RAN; Architecture description".

[3] "O-RAN: towards an Open and smart RAN", O-RAN white paper, https://www.o-ran.org/s/O-RAN-Use-Cases-and-Deployment-Scenarios-Whitepaper-February-2020.pdf

[4] O-RAN WG2 Use Case and Requirements v02.00

[5] O-RAN WG1 O-RAN Architecture Description - v01.00

[6] Intelligent Transportation Systems (ITS), ETSI TS 102 637-2 v1.2.1

[7] https://lfaidata.foundation/projects/adlik/

1.3 Definitions and Abbreviations

1.3.1 Definition

For the purposes of the present document, the terms and definitions given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

NMS: A Network Management System

O-DU: O-RAN Distributed Unit: a logical node hosting RLC/MAC/High-PHY layers based on the 7-2x fronthaul split defined by O-RAN.

O-RU: O-RAN Radio Unit: a logical node hosting Low-PHY layer and RF processing based on the 7-2x fronthaul split defined by O-RAN.

Non-RT RIC: O-RAN non-real-time RAN Intelligent Controller: a logical function that enables non-real-time control and optimization of RAN elements and resources, AI/ML workflow including model training and updates, and policy-based guidance of applications/features in Near-RT RIC.

Near-RT RIC: O-RAN near-real-time RAN Intelligent Controller: a logical function that enables near-real-time control and optimization of RAN elements and resources via fine-grained data collection and actions over E2 interface.

O1: Interface between orchestration & management entities (Orchestration/NMS) and O-RAN managed elements, for operation and management, by which FCAPS management, Software management, File management and other similar functions shall be achieved.

A1: Interface between Non-RT RIC and Near-RT RIC to enable policy-driven guidance of Near-RT RIC applications/functions, and support AI/ML workflow.

E2: Interface between Near-RT RIC and underlying RAN functions (CU-CP, CU-UP, and DU).

1.3.2 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

eNB	eNodeB (applies to LTE)
gNB	gNodeB (applies to NR)
O-DU	O-RAN Distributed Unit

O-RU	O-RAN Radio Unit
O-CU	O-RAN Central Unit
RIC	O-RAN RAN Intelligent Controller
Non-RT RIC	Non-real-time RIC
Near-RT RIC	Near-RT RIC
QoE	Quality of Experience
KQI	Key Quality Indicator
KPI	Key performance indicator
CNN	Convolutional neural network
PCA	principal components analysis
RL	reinforcement learning
DRL	deep reinforcement learning
GPU	graphics processing unit
KNN	k nearest neighbors
LSTM	long short-term memory
ML	machine learning
NN	neural network
RL	reinforcement learning
RNN	recurrent neural network
SMO	service management and orchestration
SVM	support vector machine

Chapter 2 Machine Learning

Machine learning is a field of study that provides computers the ability to learn without being explicitly programmed. The ability to learn useful information from input data can help improve RAN or network performance. For example, convolutional neural networks and recurrent neural networks can extract spatial features and sequential features from time-varying signal strength indicators (e.g., RSSI).

This chapter introduces some of the common terminology related to AI/ML based use-cases development in context of O-RAN architecture.

2.1 Common terminology and Definitions

Table 1 - Common terminology

Definitions	Note/example
Application: An application is a complete and deployable package, environment to achieve a certain function in an operational environment. An AI/ML application is one that contains some AI/ML models.	Generally, an AI/ML application should contain a logically top-level AI/ML model and application-level descriptions
ML-assisted Solution: A solution which addresses a specific use case using Machine-Learning algorithms during operation.	As an example, video optimization using ML is an ML-assisted solution.
ML Model: The ML methods and concepts used by the ML-assisted solution. Depending on the implementation a specific ML model could have many sub-models as components and the ML model should train all sub-models together.	ML models include supervised learning, unsupervised learning, reinforcement learning, deep neural network, and depending on use-case, appropriate ML model has to be chosen. Separately trained ML models can also be chained together in a ML pipeline during inference.
ML Workflow: A ML workflow is the process consisting of data collection and preparation, model building, model training, model deployment, model execution, model validation, continuous model self-monitoring and self-learning/retraining related to ML-assisted solutions	Based on ML model chosen, some or all of the phases of workflow will be included.

ML (model) Life-cycle: The life-cycle of the ML model includes deployment, instantiation and termination of ML model components.	These are operational phases: the initial training, inference, possible re-training
ML Pipeline: The set of functionalities, functions, or functional entities specific for an ML-assisted solution.	a ML pipeline may consist of one or several data sources in a data pipeline, a model training pipeline, a model evaluation pipeline and an actor.
ML Training Host: The network function which hosts the training of the model, including offline and online training	Non-RT RIC can also be a training host. ML training can be performed offline using data collected from the RIC, O-DU and O-RU.
ML Inference Host: The network function which hosts the ML model during inference mode (which includes both the model execution as well as any online learning if applicable).	The ML inference host often coincides with the Actor. The ML-host informs the actor about the output of the ML algorithm, and the Actor takes a decision for an action.
Actor: The entity which hosts an ML assisted solution using the output of ML model inference.	
Action: An action performed by an actor as a result of the output of an ML assisted solution.	
Subject of Action: The entity or function which is configured, controlled, or informed as result of the action.	
Model training information: Information needed for training the ML model.	This is the data of the ML model including the input plus optional labels for supervised training
Model inference information: Information needed as input for the ML model for inference.	The data needed by an ML model for training and inference may largely overlap, however they are logically different.
Model compiling info: Information needed for compiling the ML model.	It includes the trained ML model , model inference host info, and other possible requirements for model compiling, e.g., acceptable accuracy loss thresholds, any specific operations to be performed.
Model inference host info: Information of the inference host needed as model compiling info.	This may include information e.g., network bandwidth, memory and computing capabilities of the inference host.

Model Compiling Host: Optional network function which compiles the trained model into specific format for optimized inference execution in an inference host based on the compiling information .	Model compiling involves hardware-specific optimization to achieve improved computing and memory efficiency. According to different deployment methods, the compiled model could be published to the model store/ management module as a containerized image (including Inference Engine and compiled models) or a compiled model file format, and then deployed to the inference host. Non-RT RIC can also be a compiling host. ML compiling can be performed offline.
Model Inference Engine: The specific inference framework used to run a compiled model in the inference host.	The functions of model inference engine includes parsing model files, splitting operations, and executing inference instruction stream to finish ML model inference calculation and return inference result.
Model Management: The network function that manages the ML models to be deployed in the inference host.	Model management manages models that are onboarded directly from ML training host or those from ML compiling jost when model comiling is executed after training.
Data Discovery: Data discovery uses smart tools to collect data from multiple sources and then consolidate them to a single source for easy use.	The general process includes data collecting, data cleansing, data loading, data transforming, data mining and data visualization
Data Augmentation: Data augmentation could be used to enhance the model generalization and reduce overfitting, especially when dataset is limited or imbalance exists.	There are two types of data: Time series data: augmentation with consideration of event distribution pattern in temporal space Non-time series data: augmentation with physical constraint
Data Labelling: Data labelling is a time-consuming task and needs domain knowledge. The platform should provide or integrate various labelling/ auto-labelling tools for offline/online annotation.	The labelled data should be validated crossly, and some label mistakes should be eliminated
Feature Engineering: At the beginning of model design, we need to study and define what kinds of features could be learnt to	Design the feature extraction mechanism, such as learning from multiple modalities, or learning from single modality, etc. Meanwhile, we need to

represent the objectives or data, and what kinds of actions should be used to for better prediction performance.	define which features could be leveraged and which could be fused with other features. Nowadays, end2end system building is emerging also, but raw data is also a kind of feature in this context. This is an optional operation for model training.
Model Selection: When initiating a training task, ML designer will assess the cost and resources for training or inference, such as hardware platform to inference, GPU memory, inference speed, training accuracy, training time, etc. Based on those requirements analysis, the system can select the corresponding configurations for training or inference.	For model design, the following factors should be considered: • ML meta architecture for specific tasks • ML/DL framework (pytorch, tensorflow, caffe, etc.) • Data format of input and output • Requirements on model performance (accuracy, responding time, real-time factor, etc.) • Model footprint and HW platform (ARM, GPU, CPU, FPGA, etc.) • Task requirements
Model Optimization: Model optimization refers to the efforts to optimize model performance. Optimizing models based on certain hardware or performance metrics requirements	For example, using auto machine learning (AutoML) to optimize the hyperparameters or deep learning neural networks (DNN, CNN, RNN, etc.) structure. Optimization Metrics: • Model size • Memory used • Inference speed • Accuracy (precision/recall, etc.) Hardware platform processing capability, etc. According to the hardware platform running models, model compression could also be involved as one of optimization technique.
Model Compression: With specific real-time requirement and hardware constraints, inference engine running on edge device	At this stage, the system will check the model requirements and running

for example, model compression will be critical and a must to further reduce the model footprint and therefore to speed up the inference engine.	hardware platform to decide which compression strategy that could be adopted. The training process will be integrated with compression to generate the satisfied models
Model Training: Model training should consider the training platform capability, available resources and energy consumption for training	Meanwhile, the training process should be monitored to see whether the process is converged or collect key information, such as memory used, loss, accuracy, etc.
Model Testing: Model testing refers to validate model performance with testing data	It could include the validation process and real testing in product environment. With model testing, we can understand the trained model capability in product environment, and also possible defects.
Model Deployment: Model deployment should fully consider the inference hardware capability and stability.	Data input/output should also be considered to reduce I/O delay.
Inference Monitoring: Monitoring inference process and collect ML system/model key messages.	
Model Refine: After deploying a model for certain time, the model should be refined based on the test results or feedback from the test loop due to the shifting of data distribution, changing of environment, accumulated errors, etc.	
Continuous Operations: Provides a series of online functionalities for the continuous improvement of AI/ML models within the whole AI/ML lifecycle. It includes Verification/Monitoring /Analysis /Recommendation /Continue Optimization.	
Verification: Verified the model performance online in the real deployed environment	
Analysis: Includes data analysis, data/label conflict analysis, performance prediction, business insight etc.	
Recommendation : Co-work with analysis to provide continuous improvements recommendations.	
Continuous Optimization: Provides AI pipeline optimization, Decision optimization etc. during AI/ML LCM.	

1

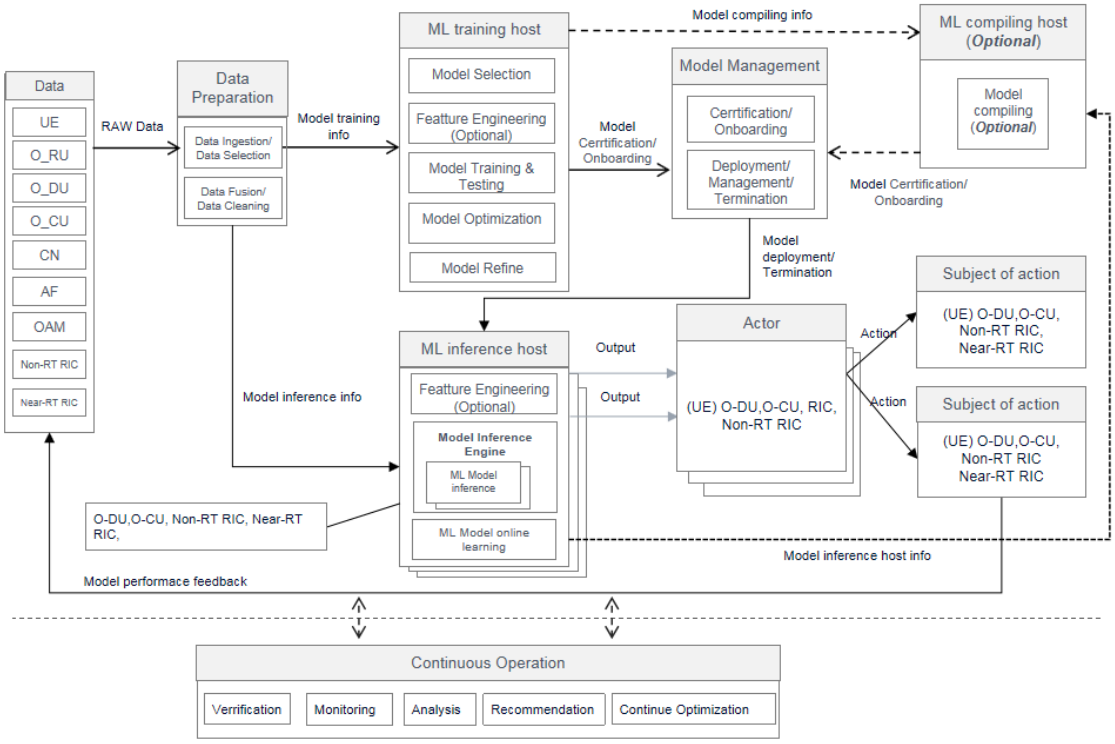


Figure 1 depicts the use of the ML components and terminologies as described in Table 1.

2.2 General principles

Principle 1: In O-RAN we will always have some offline learning as a proposed best practice (even for reinforcement learning type of scenarios). In the current document, offline training means a model is first trained with offline data, and trained model is deployed in the network for inference. Online training refers to scenarios such as reinforcement learning, where the model 'learns' as it is executing in the network. However, even in the latter scenario, it is possible that some offline training may happen.

Principle 2: A model needs to be trained and tested before deploying in the network. A completely untrained model will not be deployed in the network.

Principle 3: As a best practice, it would be useful if ML Applications are designed in a modular manner that are decoupled from one another. This includes their ability to share data without knowing each other's data needs. It also implies that they need not understand the location or nature of a data source. For example, an ML Application that is consuming RAN data need not know whether that data is being provided directly via E2, or by some other ML Application on the same Inference Host that is consuming and re-publishing that RAN data. Sections 4.2 and 4.3 describe this in more detail.

Principle 4: Given that the criteria for determining the deployment scenario for a given ML Application may differ between service providers, as a best practice, it should be possible for a Service Provider to decide whether an ML Application should

be deployed to a Non-RT RIC or a Near-RT RIC as its Inference Host. See Section 5.2 for a discussion on the types of criteria that may be weighed differently by different providers.

Principle 5: As a best practice, to improve the execution efficiency and inference performance in the inference host, the ML model for inference should be optimized and compiled with the consideration of the inference host hardware capability. A trade off between efficiency and inference accuracy need to be taken into consideration. Therefore, the optimization should take acceptable accuracy loss as one of the goals, and optimization parameters should be obtained based on this threshold.

Note: In Appendix X, an example is provided to show the comparison of the inference performance of the original models and the compiled/optimized model in different inference framework.

2.3 Types of AI

Today, AI systems have been able to accomplish certain tasks with a level of accuracy surpassing that of humans. These achievements have been within the scope of narrow AI. However, narrow AI faces challenges and gaps when being applied in real world production environment in scale. For example, most AI systems currently require enormous amounts of data to train machine learning or deep learning models and they have little ability to transfer the knowledge gained through learning from one task to another. To make AI in real-life production environment, we need evolve our AI technologies from narrow AI into advanced, trusted, and scalable AI as shown inFigure 2-1.

- **Advancing AI:** Powering advances in perception, reasoning, and understanding to help AI address complex human-like tasks.
- **Trusting AI:** Novel techniques to instrument key dimensions of trust and enable AI solutions that inspire confidence.
- **Scaling AI:** Novel technologies across the full computing stack that make AI faster, easier, and able to scale to larger and more complex problems.

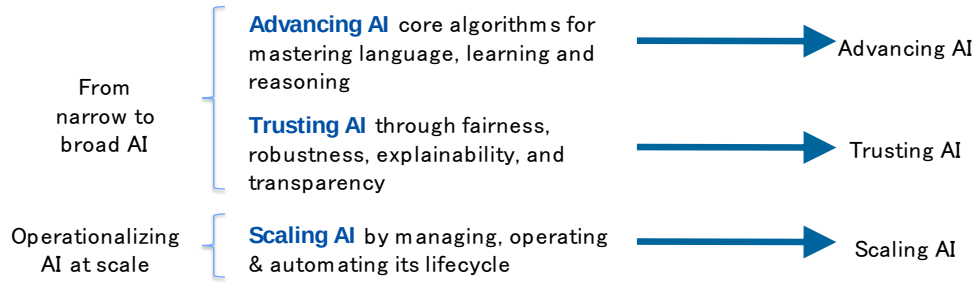


Figure 2-1 - From Narrow AI to Advanced, Trusted, and Scalable AI

Chapter 3 Types of Machine Learning algorithms

This section provides a view of how the different ML algorithms can be deployed and realized in O-RAN architecture. It does not detail or recommend the various machine learning algorithms available or recommend specific ML algorithms that should be applied to the use-cases realized in O-RAN architecture.

3.1 Supervised learning

Input data is called training data and has a known label or result. Supervised learning is a machine learning task that aims to learn a mapping function from the input to the output, given a labeled data set.

1. Regression: Linear Regression, Logistic Regression
2. Instance-based Algorithms: k-Nearest Neighbor (KNN)
3. Decision Tree Algorithms: CART
4. Support Vector Machines: SVM
5. Bayesian Algorithms: Naive Bayes
6. Ensemble Algorithms: Extreme Gradient Boosting, Bagging: Random Forest

Supervised learning can be further grouped into Regression and Classification problems. Classification is about predicting a label whereas Regression is about predicting a quantity.

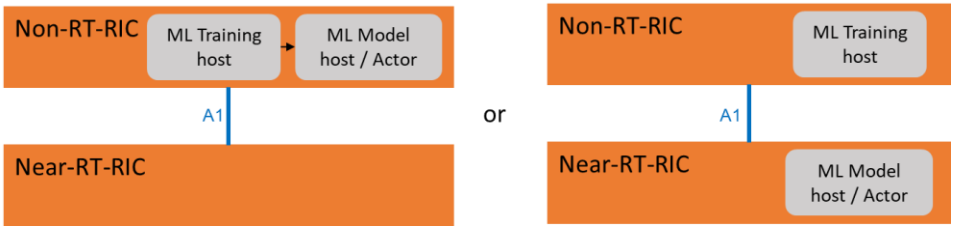


Figure 3-1 Supervised learning model training and actor locations

In supervised learning (seeFigure 3-1), Non-RT RIC is part of the SMO and thus is part of the management layer. ML training host and ML model host/actor can be part of Non-RT RIC or Near-RT RIC.

3.2 Unsupervised learning

Input data is not labeled and does not have a known result. Unsupervised learning is a machine learning task that aims to learn a function to describe a hidden structure from unlabeled data. Some examples of unsupervised learning are K-means clustering and principal component analysis (PCA).

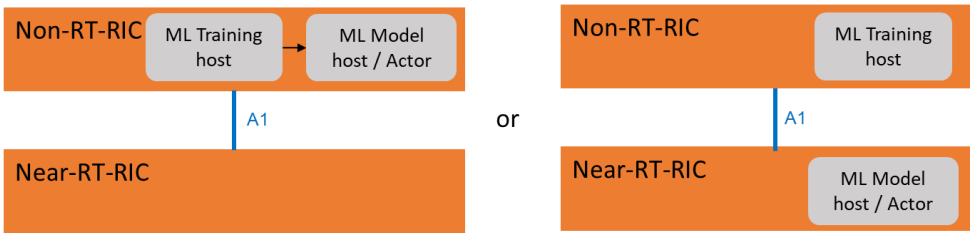


Figure 3-2 - Unsupervised learning model training and actor locations

In unsupervised learning (seeFigure 3-2), ML training host and ML model host/actor can be part of Non-RT RIC or Near-RT RIC.

3.3 Reinforcement learning

A goal-oriented learning based on interaction with environment. In reinforcement learning (RL), the agent aims to optimize a long-term objective by interacting with the environment based on a trial and error process. The most typical RL framework is based on Markov Decision Process (MDP), which is defined by a 4-tuple *(S, A, R, T)*, where:

- *S* is the state space of an environment
- *A* is the set of actions an agent can select from
- *R* is the reward function
- *T* is the transition probability function

There are multiple ways to categorize RL algorithms, and one widely used classification is: model-based RL vs. model-free RL. In wireless network, it might be beneficial to rely on a model for predicting future behavior of the environment. In model based RL algorithms, reward and transition probability is based on a predictive model of the environment. The agent uses the model to find out what will happen if an action is taken. On the other hand, the model of the environment might not be of interest or hard to obtain for some use cases. Model-free algorithms forgo any explicit knowledge of the dynamics of the environment and evaluate how good actions are through exploration and exploitation. RL algorithms can also be categorized into value-based RL vs. policy-based RL, on-policy RL vs. off-policy RL, etc. There are several RL algorithms

- Q-learning
- Multi-armed bandit learning
- Deep Q Network
- State-Action-Reward-State-Action (SARSA)
- Temporal Difference learning
- Actor-critic reinforcement learning
- Deep deterministic policy gradient
- Trust region policy optimization
- Dyna-Q
- Monte-Carlo tree search

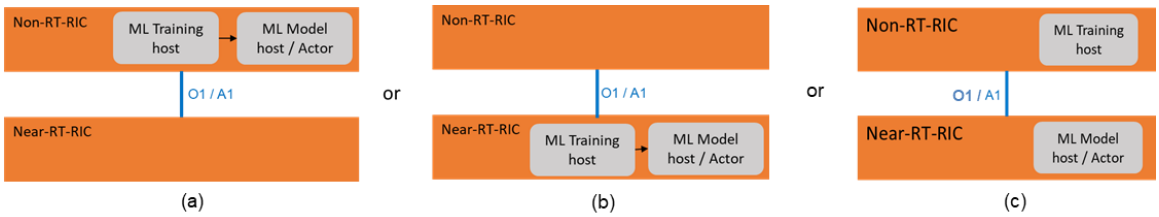


Figure 3-3- Reinforcement learning model training and actor locations

For reinforcement learning (seeFigure 3-3a), ML training host and ML model host/actor shall be co-located as part of Non-RT RIC . For online RL in the Near-RT RIC (see Figure 3-3b), ML training host and ML model host/actor shall be co-located as part of Near-RT RIC. For offline RL (see Figure 3-3c), ML training host is located as part of Non-RT RIC, and ML model host/actor is located as part of Near-RT RIC. The offline trained model is updated based on the performance monitoring of the interaction between the ML model host/actor (RL agent) and the RAN environment.

3.4 Mapping AI/ML functionalities into O-RAN control loops

There are three types of control loops defined in O-RAN. ML assisted solutions fall into the three control loops. Time scale of O-RAN control loops depend on what is being controlled, e.g., system parameters, resources or radio resource management (RRM) algorithm parameters. For example, if O-RAN control loop adapts the parameters of RRM algorithms, its time scale is slower than that of the RRM algorithm which directly controls resource.

Loop 1 deals with per TTI msec level scheduling and operates at a time scale of the TTI or above. Loop 2 operates in the near RT RIC operating within the range of 10-1000 msec. Loop 3 operates in the Non-RT RIC at greater than 1000 msec (policies, orchestration). It is not expected that these loops are hierarchical but can instead run in parallel.

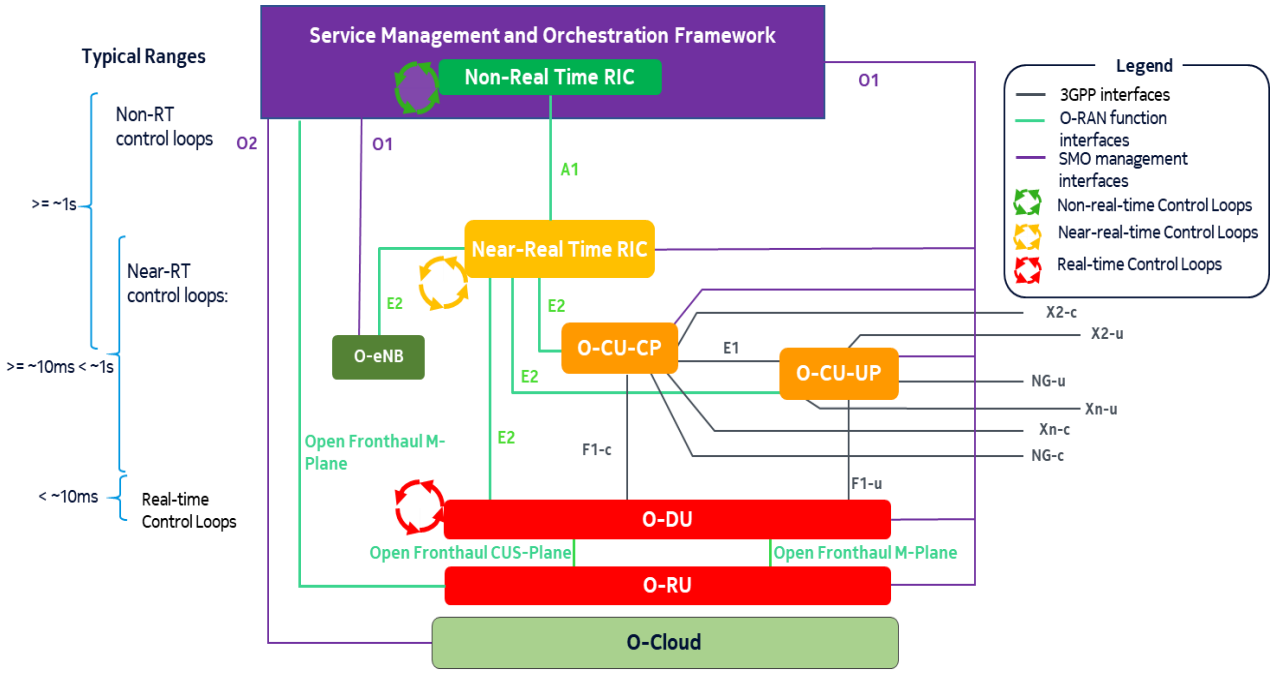


Figure 3-4 - Control loops in O-RAN

Figure 3-4 shows the three control loops in O-RAN architecture. AI/ML related functionalities can be mapped into the three loops. The location of the ML model training and the ML model inference for a use case depends on the computation complexity, on the availability and the quantity of data to be exchanged, on the response time requirements and on the type of ML model. For example, online ML model for configuring RRM algorithms operating at the TTI time scale could run in O-DU, while the configuration of system parameters such as beamforming configurations requiring a large amount of data with no response time constraints can be performed using the combination of Non-RT RIC and SMO where intensive computation means can be made available.

In the first phase of O-RAN, ML model training will be considered mainly in the Non-RT RIC and ML model inference will be considered in loops 2 and 3. For loop2, the ML inference is typically running in Near-RT RIC. For Loop 1, the ML model inference is typically running in an O-DU. ML workflow on loop 1 is not covered by this version of the technical report. While ML model implementation in O-RU could be envisaged, it is presently not supported in O-RAN.

3.5 Federated learning

Federated learning is a well-known distributed learning methodology where multiple AI/ML entities collaboratively train an AI/ML model. Instead of gathering training data into a central server, federated learning keeps the training data decentralized to mitigate privacy risks. AI/ML models are trained locally in distributed entities, and the local models are aggregated by a central server. The central server orchestrates the federated learning.

In O-RAN architecture, since both Non-RT RIC and Near-RT RIC are capable of AI/ML training, the Non-RT RIC can serve as the central server in the federated learning, and its connected Near-RT RICs can serve as distributed AI/ML entities as illustrated in Figure 3-5.

Note: Other federated learning deployment scenarios are FFS.

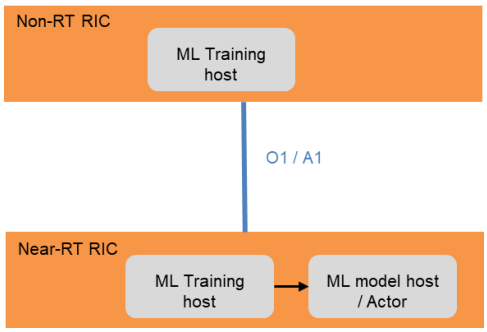


Figure 3-5: Federated learning model training and actor locations

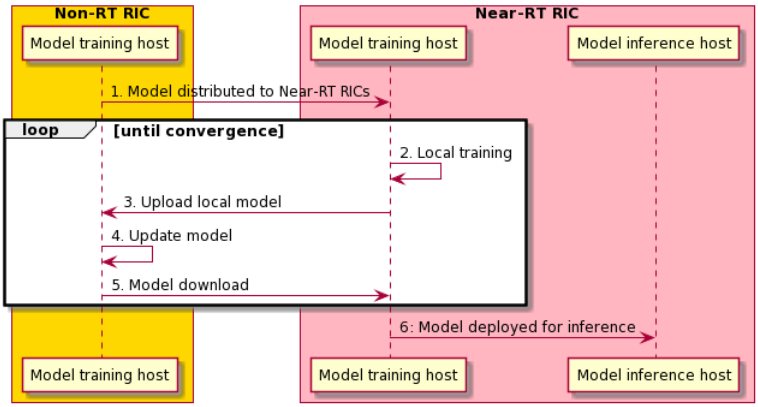


Figure 3-6: Procedure of federated learning among Non-RT RIC and Near-RT RICs

The procedure of federated learning among Non-RT RIC and Near-RT RICs can be summarized into following steps:

- Step 1: The Non-RT RIC distributes the AI/ML model to its connected Near-RT RICs.
- Step 2: The ML training host in the Near-RT RIC trains local model using training data collected locally. Note that training data collected locally are not required to be transferred to the Non-RT RIC.
- Step 3: Local model weights or gradients are uploaded to the Non-RT RIC.
- Step 4: The training host in the Non-RT RIC update the AI/ML model (e.g., using FedSGD, FedAvg, etc.).
- Step 5: The updated model is downloaded to the Near-RT RICs.
- Step 6: If the model training converges, then the model is deployed for inference in the Near-RT RIC.

A1-ML (model management) service can be used to manage model downloading/distribution and uploading/aggregation in federated learning. For example,

- The Non-RT RIC can subscribe/request model uploading from its connected Near-RT RICs.
- The Non-RT RIC can notify its connected Near-RT RIC to download a model.

ML model download and upload between Non-RT RIC and Near-RT RICs over O1 or A1 interface is FFS.

Chapter 4 Procedure/Interface framework, Data/Evaluation pipelines

4.1 AI/ML General Lifecycle Procedure and Interface Framework

This chapter first provides the general framework of AI/ML procedure and interfaces, which addresses the ML components rather than network functions (Non/Near-RT RIC, etc.). The potential mapping relationship between the ML components and network functions, interfaces defined in O-RAN are also illustrated in Figure 4-1.

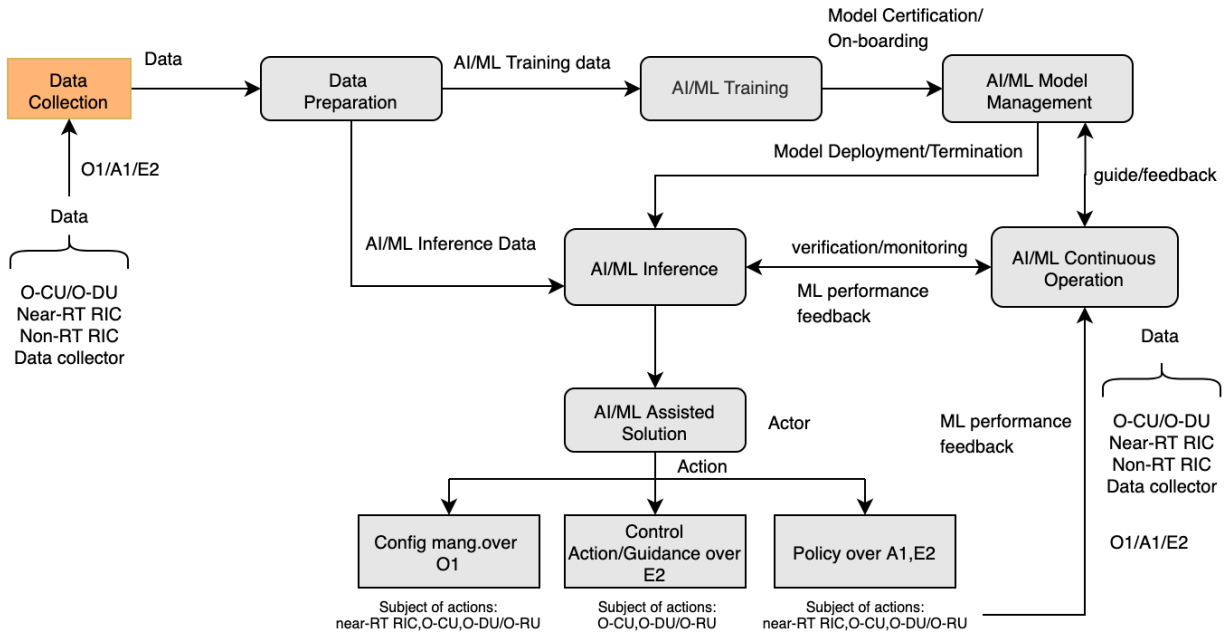


Figure 4-1 - AI/ML General Procedure

Note: *ML capabilities shall be stored in the management system (FFS). Query and Discovery of ML capabilities are terminated in the management system but not shown. ML inference host often coincides with the actor*

Note*:"AI/ML Training" in the* Figure 4-1 *indicates "Training" only. (currently does not include re-training)*

As Model Management/Data Preparation/AI/ML Training are implementation variability components, there are many combinations of the deployment scenarios. The typical deployment scenarios that are considered for AI/ML architecture/framework in O-RAN architecture are:

1. Deployment Scenario 1.1: AI/ML Continuous Operation/AI/ML Model Management/Data Preparation/ AI/ML Training and AI/ML Inference are all in Non-RT RIC

2. Deployment Scenario 1.2: AI/ML Continuous Operation/Data Preparation (for training)/AI/ML Training are in Non-RT RIC, AI/ML Model Management is out of Non-RT RIC (in or out of SMO). Data Collection (for inference)/Data Preparation (for inference)/AI/ML Inference is in Near-RT RIC
3. Deployment Scenario 1.3: AI/ML Continuous Operation/AI/ML Inference are in Non-RT RIC. Data Preparation/AI/ML Training / AI/ML Model Management are out of Non-RT RIC (in or out of SMO).
4. Deployment Scenario 1.4: Non-RT RIC acts as the ML training host for offline model training and the Near-RT RIC as the ML training host for online learning and ML inference host.
5. Deployment Scenario 1.5: Continuous Operation/Model management/Data Preparation/ML Training host are in Non-RT RIC. O-CU/O-DU act as the ML inference host (for FFS).

Note: Deployment of "AI/ML Continuous Operation" outside of non-RT RIC is in study and FFS

エラー! 参照元が見つかりません。 shows the various deployment scenarios and interfaces.

Table 2 - AI/ML deployment scenarios

Deployment Scenario	**Data Preparation**	**AI/ML Training**	**AI/ML Inference**	**Model Management**	**Continuous Operation**	**Subject of Action**	**Action from inference host to subject**		**Enrichment data for inference**
							Config Mgmt. (CM)	**Policy / Control**	
Scenario 1.1	Non-RT RIC	Non-RT RIC	Non-RT RIC	Non-RT RIC	Non-RT RIC	Near-RT RIC	O1	A1 (policy)	SMO internal
						O-CU, O-DU, O-RU	O1	N/A	SMO internal
Scenario 1.2	Non-RT RIC and Near-RT RIC	Non-RT RIC	Near-RT RIC	Out of Non-RT RIC	Non-RT RIC	Near-RT RIC	near-RT RIC internal	near-RT RIC internal	A1
						O-CU, O-DU, O-RU	N/A	E2 (control/policy)	E2 (if applicable)
Scenario 1.3	Out of Non-RT RIC	Out of Non-RT RIC	Non-RT RIC	Out of Non-RT RIC	Non-RT RIC	Near-RT RIC	near-RT RIC internal	near-RT RIC internal	A1
						O-CU, O-DU, O-RU	FFS	FFS	FFS

Scenario 1.4	Non-RT RIC and Near-RT RIC	SMO/Non-RT RIC for offline training Near-RT RIC for online learning	Near-RT RIC	Non-RT RIC and Near-RT RIC	Near-RT RIC	Near-RT RIC	Near-RT RIC internal	Near-RT RIC internal	A1
						O-CU, O-DU, O-RU	N/A	E2 (control/policy)	E2 (if applicable)
Scenario 1.5 (FFS)	Non-RT RIC	Non-RT RIC	O-CU / O-DU	Non-RT RIC	Non-RT RIC	O-CU, O-DU, O-RU	FFS	FFS	FFS

Note: Configuration management for scenario 1.2 and 1.4 via E2 are FFS; Non-RT RIC can use SMO internal interfaces to trigger configuration changes over O1

Note 2: ML model transfer between Non-RT RIC and Near-RT RIC over A1 interface is FFS.

Based on the framework, some key phases of machine learning are expected to be applied to any ML-assisted solution planned in O-RAN architecture. Any use case defined for ML-assisted solution shall have one or more phases (as applicable) and the phases are defined below:

1. **ML model and inference host capability query/discovery**

This procedure shall be executed whenever AI/ML model is to be used for ML-assisted solution. This procedure can be executed at start-up or run-time (when a new ML model is to be executed or existing ML model is to be updated). The SMO/Non-RT RIC will discover various capabilities and properties of the ML inference host, and the ML-assisted solution such as:

a) Processing capability of the inference host (for example: dedicated resources such as CPU/GPU, memory etc. for ML model inference).

b) Requirements on the ML assisted solutions such as execution time cap, delay sensitivity

c) Properties such as supported ML model formats and ML engines (for example: Protobuf, JSON, or any ONAP specific VES data formats).

d) NFVI based architecture support in MF to run ML model(s)

e) Data-sources available to run ML-pipeline (for example: support for data streams, data lake, or any specific database access)

This discovery of the capabilities and properties shall be used to check:

a) if an ML model can be executed in the target ML inference host (MF),

b) which ML models can be executed in the MF.

Note: Exact mechanism and contents of capabilities discovery is FFS.

2. **ML model Training and Generation**

This procedure corresponds to design time selection and training of a ML model in relation with a specific ML-assisted solution (use case) to be executed. The ML designer or the SMO/Non-RT RIC will select and onboard the ML model and relevant meta data into the ML training host. Utilizing on the ML training data collection, the ML training host will initiate the model training. Once the model is trained and validated, it is published back in the SMO/Non-RT RIC catalogue. In addition, new trained ML models for a given ML-assisted solution can be generated also by modification techniques such as compression, truncation etc., or training on different training data sets. These ML models are also published in the SMO/Non-RT RIC catalogue.

3. **ML model Selection**

The ML designer can check whether a trained ML model from the SMO/Non-RT RIC catalogue can be deployed in the ML inference host for the given ML-assisted solution, by mapping the ML model and ML assisted solution requirements to performance capabilities discovered from Step 1. Upon successful validation, ML designer will inform the SMO/Non-RT RIC to initiate model deployment.

4. **ML model Deployment and Inference**

The AI/ML model that is selected for the ML use case can be deployed via containerized image to MF where ML model shall be running. The container image also includes the necessary configuration of ML inference host with the AI/ML model description file.

Note: The O1 interface mechanism for ML model deployment is being specified by WG1.

Once the ML model is deployed and activated, ML online data shall be used for inference in ML-assisted solutions, which includes:

a) 3GPP specific events/counters (across all different Managed Elements) over O1/E2 interface
    a. Events: 3GPP 32.423
    b. Counters: 3GPP 32.425

b) Non-3GPP specific events/counters (across all different Managed Elements) over O1/E2 interface (to be defined in O-RAN WGs)

c) Enrichment information from Non-RT RIC over A1 interface (to be defined in O-RAN WGs)

Based on the output of the ML model, the ML-assisted solution will inform the Actor to take the necessary actions towards the Subject. These could include CM changes over O1 interface, policy management over A1 interface, or control actions or policies over E2 interface, depending on the location of ML inference host and Actor.

5. **ML model performance monitoring**

The ML inference host and actors are expected to feedback or report the performance of the ML model to the SMO/Non-RT RIC so that the SMO/Non-RT RIC can monitor the performance (e.g., accuracy, running time, network KPIs) of the ML

model and potentially update the model or reselect the model to be executed. Based on the use-case, specific set of data as applicable for use-case shall be used for ML model re-training. Based on the performance evaluation, either some guidance can be provided to use a different model in the ML inference host, or a notification can be sent indicating the need for retraining the model.

Note: Feedback mechanism and how the ML model switching can occur at runtime is FFS.

6. **ML model retraining update**

Based on the feedback and data received from various MFs and actors, the SMO/Non-RT RIC can inform the ML designer that an update is required to the current model. The ML designer will initiate the model selection and training step, but with the existing trained model. Once the model has been retrained, it will be deployed as described in Step 3, and the updated model will be used for ML inference.

7. ML model reselection

Based on the feedback and data received from various MFs and actors, the SMO/Non-RT RIC can inform the ML designer or the the respective module that the running ML model does not comply with the requirements (e.g., because the HW capabilities of the ML inference host or the execution time/delay constraints have changed) and thus a different ML model is necessary to drive the ML application. The SMO/Non-RT RIC or the ML designer selects a suitable ML model from the ML model catalogue and deploys it in the ML inference host as in Step 4.

8. ML model termination

In certain scenarios involving severe degradation in the ML model performance, a simple model update and redeployment may not be the suitable option. To avoid large impact on network, malfunctioning ML models (or models that are degrading) need to be terminated. Non-RT RIC needs to have access to ML model's termination conditions to determine whether a ML model is working properly or not. Termination conditions are regarded as ML model attributes.

For example, if a ML model is used to predict QoE, then one termination condition for this ML model could be "when the prediction accuracy is below a certain threshold". Typically, the statistical properties of a target variable, which the model is trying to predict, changes over time (i.e., drift) in unforeseen ways. This causes problems because the prediction become less accurate as time passes. In another example, if a ML model is used to optimize handover sequences, then one possible termination condition could be "when the generated neighbor relation table leads to an increase in number of handover ping pongs or handover failures above a certain threshold". Network KPI monitoring can be used as general termination conditions for an AI/ML model to detect anomaly, i.e., when the performance of a model does not meet its baseline.

For AI/ML-assisted solutions, there are at least following scenarios:

1. A single ML model (for one application/use case) impacts the monitored network KPIs
2. Multiple ML models are chained together in one application/use case, and these models collectively contribute to the monitored network KPIs
3. Multiple ML models for different applications/use cases contribute to the monitored network KPIs

Mechanism to identify malfunctioning ML model varies for the above scenarios. For scenario 1, network KPI degradation would serve as a good indicator for ML model performance degradation. However, for scenario 2 and 3, it might be hard to pin-point the degraded model(s) only based on the network KPI monitoring. Different approaches can be used to address scenario 2 and 3. For example, Non-RT RIC could first identify candidate models that may be causing the network KPI

degradation, and then Non-RT RIC could terminate these suspect models one by one until the RAN performance gets back to normal. Another approach could be that Non-RT RIC decides to terminate all relevant AI/ML models at the same time.

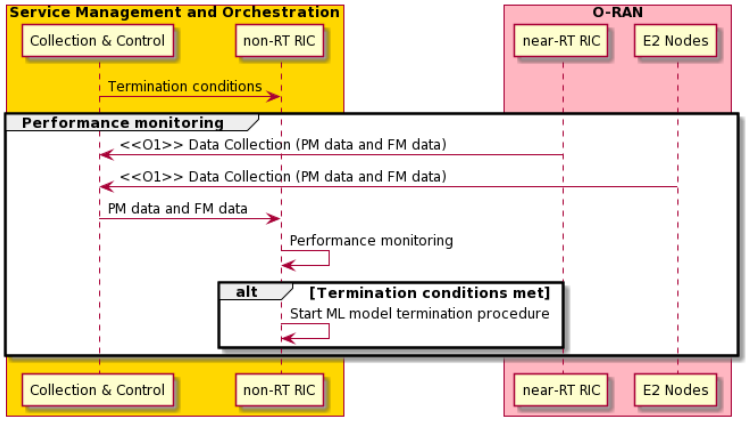


Figure 4-2. ML model termination based on performance monitoring at Non-RT RIC

As illustrated in Figure 4-2, AI/ML model performance monitoring is assumed to be placed in the Non-RT RIC. Based on the model performance monitoring, the Non-RT RIC can calculate performance metrics and compare them against termination conditions of a model. If the termination conditions are met, the model termination procedure should be triggered.

In the ML model termination procedure, ML inference host, which can be in Non-RT RIC or Near-RT RIC based on deployment scenarios, would receive termination commands requesting to stop the ML inference. A backup solution, which can be another well trained and tested ML model, a slightly modified version of the given ML model (e.g., compressed or less compressed versions, truncated or less truncated versions etc.), or, more conservatively, a non-AI-based algorithm, needs to be provided to ML inference host. After terminating the model and activating the backup solution, the ML inference host should send out messages to acknowledge the commands. The above discussion is summarized in Figure 4-3.

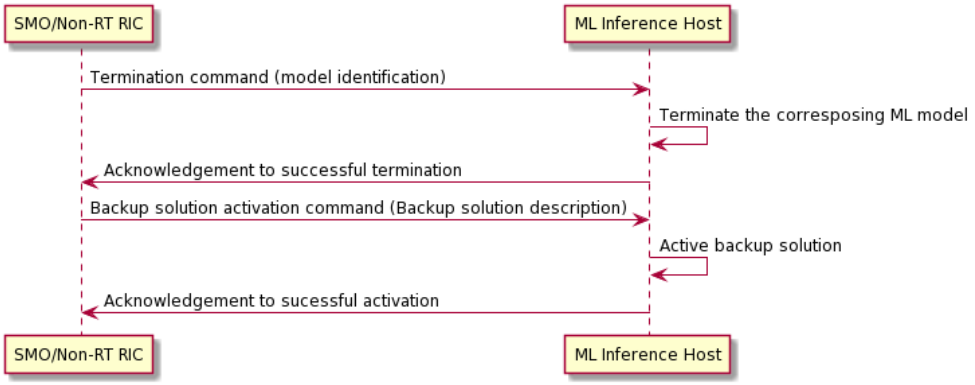


Figure 4-3. ML model termination procedure

For deployment scenario 1.1, where Non-RT RIC acts as inference host, termination/backup activation commands and acknowledgements would be communicated via SMO internal interface. On the other hand, for deployment scenario 1.2, in which AI/ML model is deployed in the Near-RT RIC, whether the termination command and acknowledgement should be

carried over O1 or A1 is left for further study. Moreover, how SMO/Non-RT RIC could delegate model performance monitoring and the decision making on ML model termination to the Near-RT RIC is also left for further study. Other deployment scenarios, e.g., in which inference host could be in O-CU/O-DU, are FFS.

4.2 Model Design and Composition

ML model design is the first step to conceiving the initial model. This requires connecting to data sources, parsing messages and tokenizing to create and select features. This activity is offline and requires data exploration mechanisms to help the model designer.

One question to address for complex problem spaces is whether to design the solution as a single model with many inputs, or as a chain of modular models. These two approaches can be seen in the figures below.

A chain of models provides a more modular solution that can facilitate reusability. For example, another model X may be shown to produce a better prediction *A-out* by considering an additional input *m*. In such a case, model A can be replaced with model X without the need for retraining or otherwise modifying models B or C.

On the other hand, in a chaining approach any errors in the prediction *A-out* will be propagated to model C, perhaps resulting in a less accurate prediction C-out than produced by the Single model approach shown below.

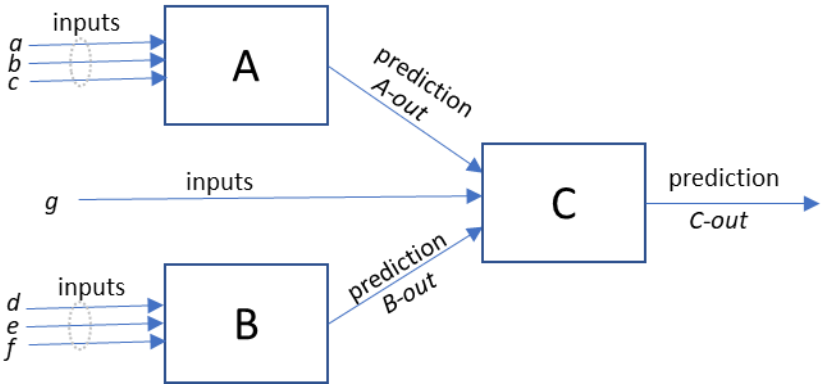


Figure 4-4 - Chained modular models

A single model allows machine learning to have access to all inputs, which can detect unexpected data relationships and perhaps lead to a more accurate overall model.

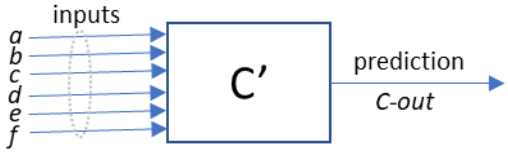


Figure 4-5 - Single model with many inputs

Note that the predicted values *A-out* and *B-out* are not directly available to the single model C' whereas they are available to the chained model C. However, the model C' could be designed to derive these in the same way models A and B did in the chained approach.

Note also that inputs *a*, *b*, *c*, *d*, *e* and *f* are not directly available to as inputs to model C, only the predicted values *A-out* and *B-out*. If they are not needed, then the chained approach may work well and achieve the desired modularity. In this case on would expect models C and C' to produce equivalent "*C-out*" predictions. If, however, it turns out that the raw variables *c* and *d* are also useful for a good prediction by model C, then the chained approach may not yield as good a prediction as the single model C' approach might which has access to those inputs. Of course such a problem could be mitigated by also providing input values *c* and *d* to model C as shown below.

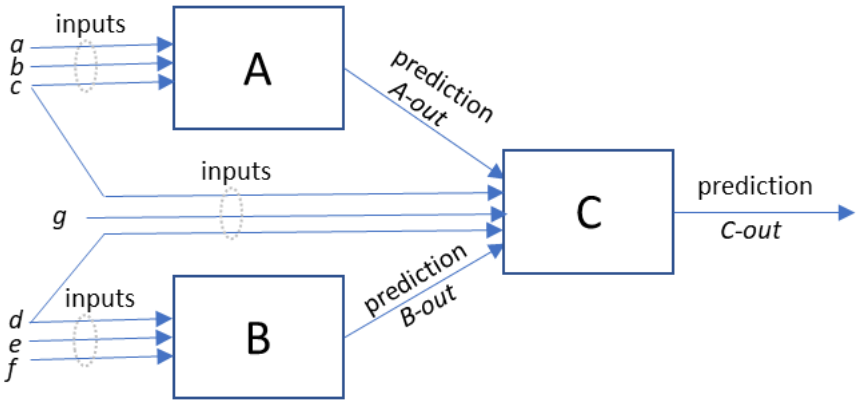


Figure 4-6 - Chained modular models with common inputs

As an example of a chained model, consider a business problem that attempts to predict when a UE's serving cell QoE will deteriorate to an unacceptable level, and also predicts when each of the neighbor cell's QoE reaches an acceptable level. Such a prediction would be used it to determine when to trigger a handover, as well as to select among the various handover cell options.

One way to design such a solution would be as follows:

- A: RF signal strength predictor – Predicts RF signal KPIs a UE would experience with a neighbor cell at the current time "x", as well as predict the signal strength that same UE would experience with both its current serving cell and its neighbor cells at time "x+Δ".
- B: Cell utilization predictor – Predicts the cell utilization KPIs for both the serving and neighbor cells above at time "x+Δ".
- C: QoE predictor – Predicts the QoE KPIs that a UE would experience at time "x+Δ" for both its current serving and neighbor cells.

Such a modular approach could be desirable in that other uses for RF signal strength prediction and cell utilization prediction might be envisioned other than to predict UE future QoE. Also, a modular approach would allow each of the prediction models to evolve separately. For example, cell utilization predictor "B" might be improved by including as input trending

information or venue schedule information among its inputs without having to retrain a single model "C'". Or an RF signal prediction model might be improved based on inputs capturing the UEs predicted travel path based on commute patterns.

However, in order for such a chained approach to work, the variables available to the QoE predictor must be carefully considered. Whether a chained or a single model approach would be better for solving such a business problem would require analysis that is beyond the intended scope of this paper and will be left as an exercise for the reader.

4.3 Model Runtime Access to Data

In the prior section we noted that the value of chaining models is the modularity which can be realized. In Figure 4-4 for example, the models A, B, and C could be improved upon and replaced independent of each other. Such modularity could facilitate a marketplace whereby different vendors produce different modular solutions in the predictive space, allowing service providers to select from among the various models with the best predictive abilities for their specific environment. Because such marketplace vendors would want to provide complete solutions and not simply ML models, we will extend the concepts in the last section to apply also to applications. We will thus in this section re-interpret figures Figure 4-4, Figure 4-5, Figure 4-6 as representing applications A, B, C and C' which contain the models in question.

Extending the discussion of the prior section, there can be scenarios in which two separate applications require access to the same data. This was seen in Figure 4-6 with applications A and C sharing input c and B and C sharing input d. Because such sharing of inputs may be common it would be wise to avoid an approach in which applications are seen as "owning" data. In addition, independence of the applications is impacted if one is aware of the inputs required of the other. For example, in Figure 4-6 as drawn, application A can be seen as knowing that application C requires Predication A-out as input. If application C was changed to no longer require Prediction A-out as input, then application A would need to be appropriately modified. Such coupling of applications negatively impacts overall solution modularity.

Rather, a better paradigm might be one in which data is commonly held by all applications, those applications being granted access to the data that they require. This common repository for the data along with the mechanism for making that data accessible to each application through some mechanism can be thought of as being the "platform" that underlies the "applications."

Figure 4-7 illustrates a re-imaging of Figure 4-6 with such an approach in mind

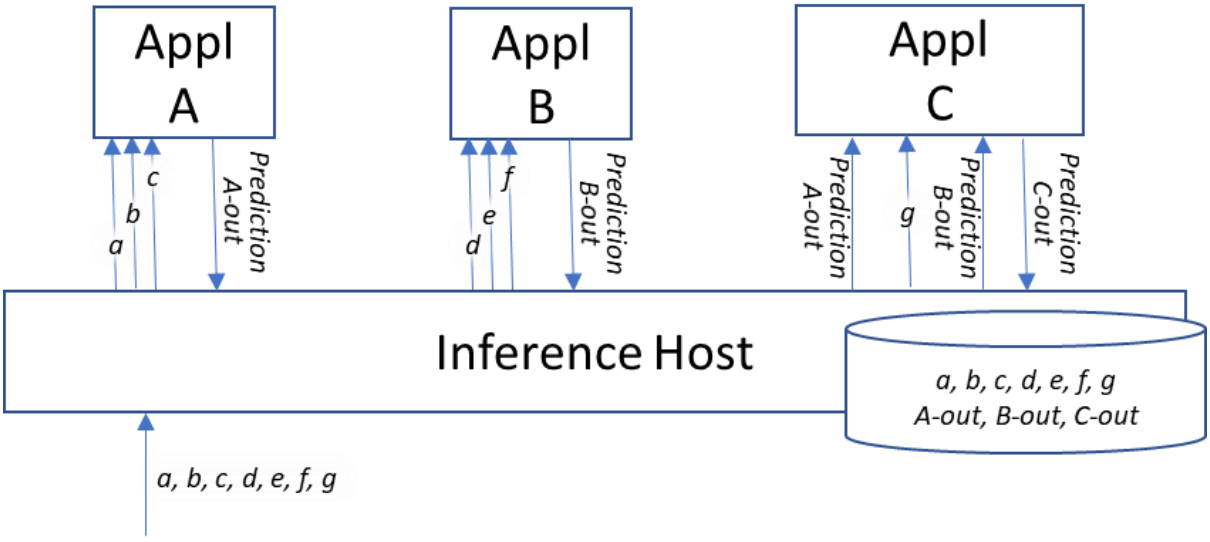


Figure 4-7 - Chained modular models with common inputs

With such an approach, applications can be created independently with their own independent descriptors. When an application is loaded, it could "register" with the Platform to declare what types of data it consumes and what types of data it produces.

For example, when application A is loaded, its descriptor would be used to declare that it consumes as input variables *a*, *b* and *c* and that it writes Prediction *A-out*. If the consumed variables are standard attributes that the Platform knows how to obtain (e.g., E2 data), it could respond that those input variables are available through the Platform. Later application A could send a formal "subscription" request with more information (such as the specific geographic "scope" from which to collect that input) to have the Platform actually start providing those values. Regarding the produced variables (e.g., *A-out*), the Platform could assign some space in the common data repository for that output to be written. The Platform might also perform some added value validations to ensure that no other registered application also produces/writes that same variable, or if it does to ensure that the two application instances work within different "scopes" (e.g., geography).

Extending our example, when application C is loaded its descriptor would be used to declare that it reads as input variables *g*, *Prediction A-out* and *Prediction B-out*. Variable *g*, being standard in our example, would be treated as described above. The Platform could also respond that input variable *Prediction A-out* is also available through the Platform. However, if application B had not yet been loaded and registered, the Platform would respond that no source for *Prediction B-out* can be found. Perhaps application C has been written such that *Prediction B-out* is optional input. In this case application C would respond as such and processing would proceed. When application B was later loaded and registered, the Platform could notify all applications of new functionality being available. At that point application C could decide to re-register, again asking for a source for *Prediction B-out*, this time with favorable results.

Thus, the Platform could provide some added value services to ensure that applications are loaded and registered in the proper order.

The above described Platform responsibilities with respect to application registration. Now we can address data subscription.

Let's return to our example in the prior section whereby application A is an RF signal predictor, application B a cell utilization predictor and application C a UE QoE predictor. Perhaps the service provider considers cell utilization prediction to be a fundamental need that should be always "on" for all cells. The service provider would thus configure application B or provide it some policy (e.g., received across an A1-P interface) to generate these predictions, specifying the prediction interval. Application B would then send a "subscribe" request to the Platform asking for variables *d*, *e* and *f* including the desired measurement interval. As part of this "subscription" request Application B would also indicate that it will begin writing *Prediction B-out* with a certain measurement interval. The Platform would determine if these variables are already in the common data repository or not, and if not the Platform would go about securing them (e.g., by sending an E2 SUBSCRIBE request to the appropriate RAN network functions). The Platform would forward to application B the data content found in the data respository that matches that application's subscription request. Upon having secured its input data, application B would continuously be writing *Prediction B-out* to the common data repository.

For applications A and C, however, perhaps the service provider only wants prediction for a certain set of UEs. One way to accomplish this is for service provider to configure application C or provide it some policy describing the UE set of interest as well as the prediction interval. Application C subscription of its input variable *g* would proceed with the Platform in the same way as was described above for Application B. Application C would also subscribe to input variables *Prediction A-out* and *Prediction B-out* with a certain measurement interval, the former including a descriptor of the UE set of interest. For *Prediction B-out*, the Platform would determine that those values are already in the common data repository with the desired measurement interval and forward that content to application C.

For *Prediction A-out*, however, the Platform would determine that there is no data in the common data repository, nor is there any application that has subscribed to write it. The Platform would then identify the application that has registered for writing this data type, in this case application A, and forward to it the subscription request information to that application A. (Note that this is the local equivalent functionality of the Platform sending the E2 SUBSCRIBE request to the RAN for the application B subscription request.)

Application A would receive the *Prediction A-out* subscription request, including the measurement interval and UE set of interest, and determine what the implications are for its own data needs. It would then send to the Platform a subscription request asking for variables *a*, *b* and *c* along with the measurement interval and UEs of interest. It would also declare its intention to begin writing *Prediction A-out* with a certain measurement interval. Assuming that the Platform does not find this data already in the common data repository, it would go about securing it (e.g., via an E2 SUBSCRIBE request), forwarding the resultant data values to application A. At that point application A would begin writing Prediction A-out to the common data repository, which the Platform would in turn begin forwarding to application C.

4.4 Data, Model Training and Model Evaluation pipeline

This section describes the data, model training and evaluation pipelines.

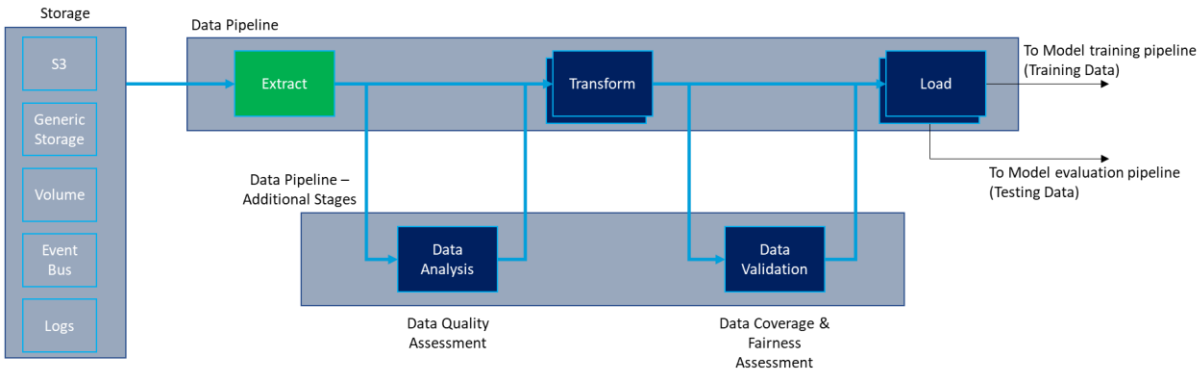


Figure 4-8 - Data pipeline

Figure 4-8 defines the data pipelines. The "extract-transform-load" (ETL) process describes how data can be extracted from storage, transformed and loaded into training and testing sets. Additional data quality and validation stages can be inserted into the ETL pipeline. Data cleaning can also be part of the Transform block. This is outside the scope of WG2.

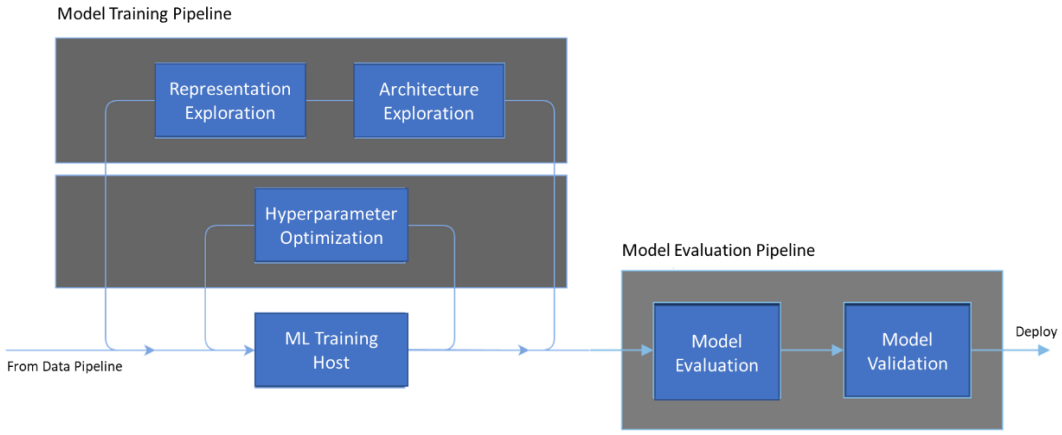


Figure 4-9 - Model training and evaluation pipelines

Figure 4-9 shows the model training and evaluation pipelines. Model training pipeline may change with model types. Model evaluation pipeline, however, is a more generic process. Model evaluation can be used to evaluate a single model or extended to select the best model from a range of models.

The end-to-end training process includes the following

- Fulfills the data requirements of the model (format, sample distribution, extent)
- Connects with requisite data via Data Pipeline (simplest ex: a data broker)
- Partitions data into appropriate sets (training, validation, testing, sample)
- Keeps Training data cached for error recovery and subsequent usage
- Manages model training though all phases

- Implements training by invoking a Model Training Pipeline
- Training Client tunes model parameters during training phases
- Scoring client monitors performance in order to declare training phase complete
- Communicates with license manager for usage and versioning

4.5 ML Model Lifecycle Implementation Example

The section provides an example (see Figure 4-10) of ML model lifecycle implementation example and key phases involved in the design and deployment in O-RAN architecture.

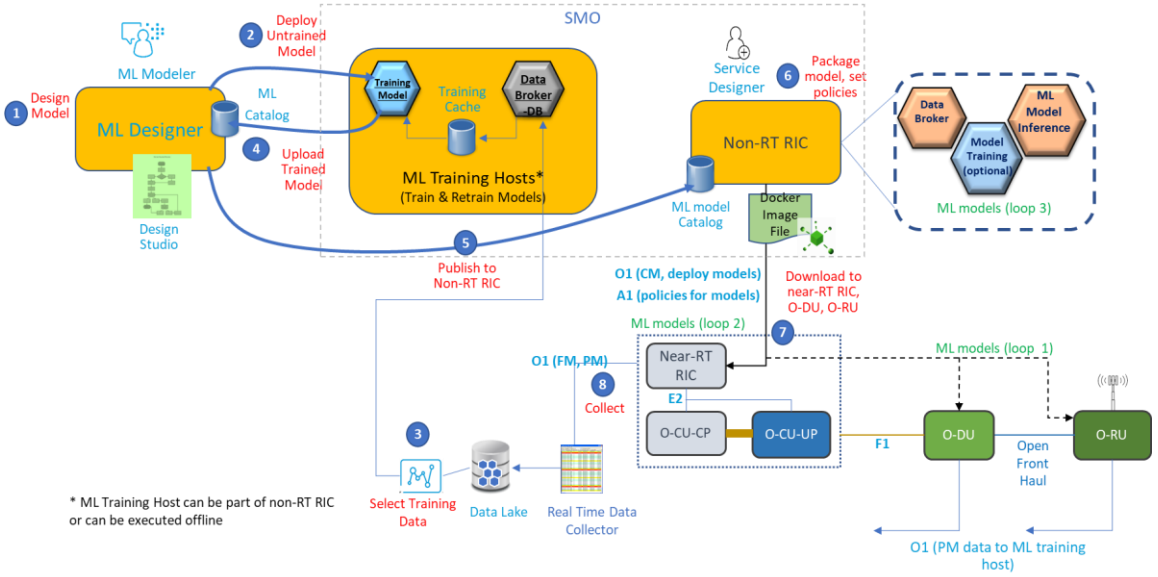


Figure 4-10 - ML model lifecycle (an implementation example)

Note: ML Model capability query and discovery can occur in ML designer and Non-RT RIC.

The typical steps involved in AL/ML based use-case application in O-RAN architecture is shown in Figure 4-10 considering supervised/unsupervised learning ML models. The steps for reinforcement model could vary with respect to ML training host and the related interaction flows.

1. ML Modeler uses a designer environment along with ML toolkits (e.g., scikit-learn, R, H2O, Keras, TensorFlow) to create the initial ML model
2. The initial model is sent to training hosts for training
3. The appropriate data sets are collected from the Near-RT RIC, O-CU and O-DU to a data lake and passed to the ML training hosts.
4. The trained model/sub models are uploaded to the ML designer catalog (one such open source catalog platform is AcumosAI). The final ML model is composed.

5. The ML model is published to Non-RT RIC along with the associated license and metadata.
6. Non-RT RIC creates a containerized ML application containing the necessary model artifacts (when using AcumosAI, the ML model's container is created in Acumos catalog itself).
7. Non-RT RIC deploys the ML application to the Near-RT RIC, O-DU and O-CU using the O1 interface. Policies are also set using the A1 interface.
8. PM data is sent back to ML training hosts from Near-RT RIC, O-DU and O-CU for retraining.

Note that Near-RT RIC can also update ML model parameters at runtime (e.g., gradient descent) without going through extensive retraining. Training hosts and ML designers can also be part of Non-RT RIC.

Chapter 5 Deployment Scenarios

This chapter describes the high-level architecture of deployment scenarios defined in Section 5.1 and also captures the sequence diagrams to show end-to-end flows.

The current version captures the deployment scenarios 1.1 (see Figure 5-1), scenarios 1.2 (see Figure 5-2), scenarios 1.3 (see Figure 5-3) , scenario 1.4 (see Figure 5-4).

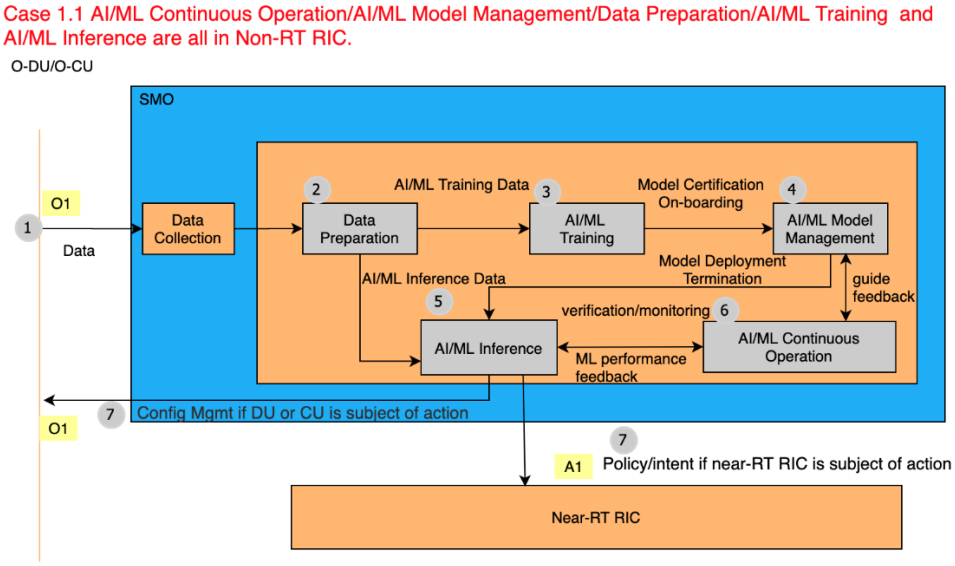


Figure 5-1 - Deployment scenario 1.1 -– AI/ML training and inference host locations

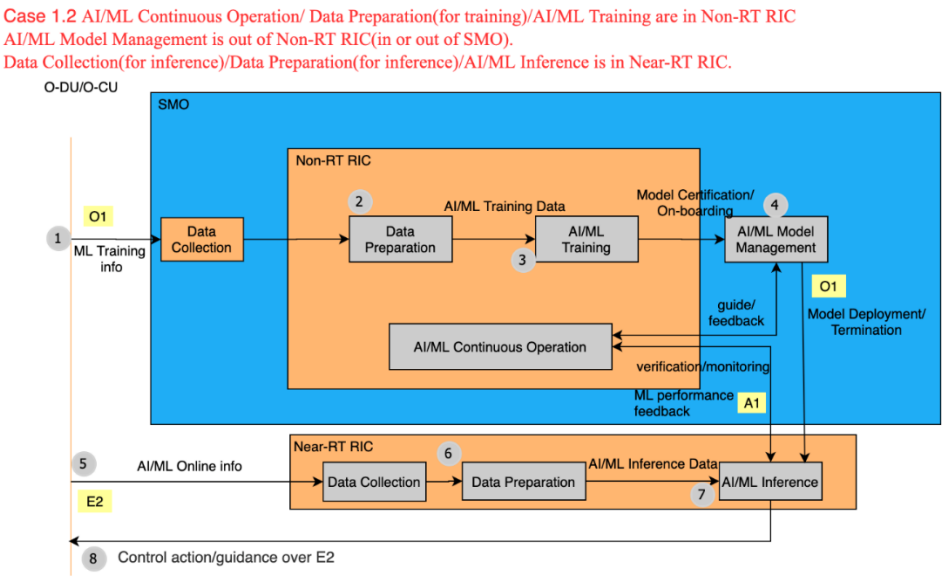


Figure 5-2 Deployment scenario 1.2 -– AI/ML training and inference host locations

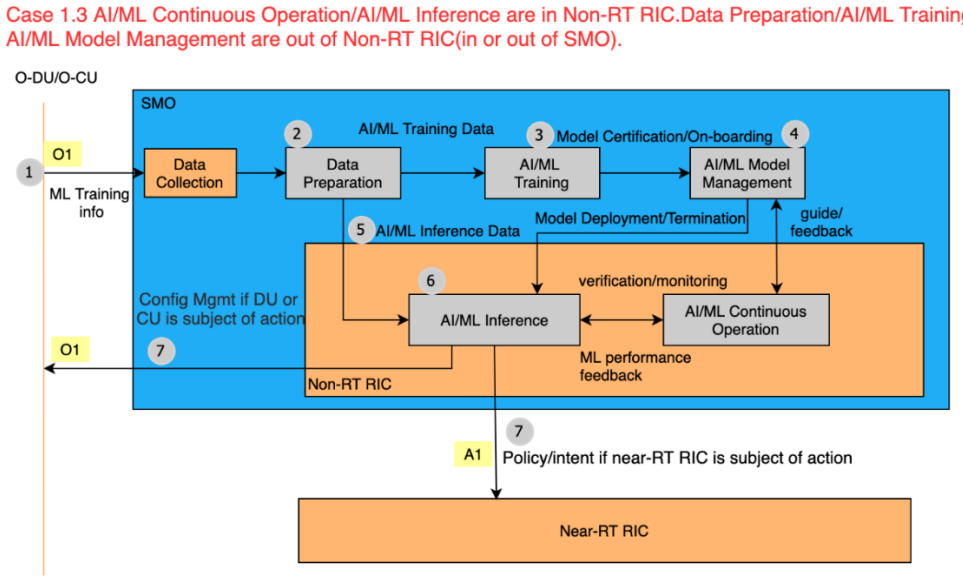


Figure 5-3 Deployment scenario 1.3 - AI/ML training and inference host locations

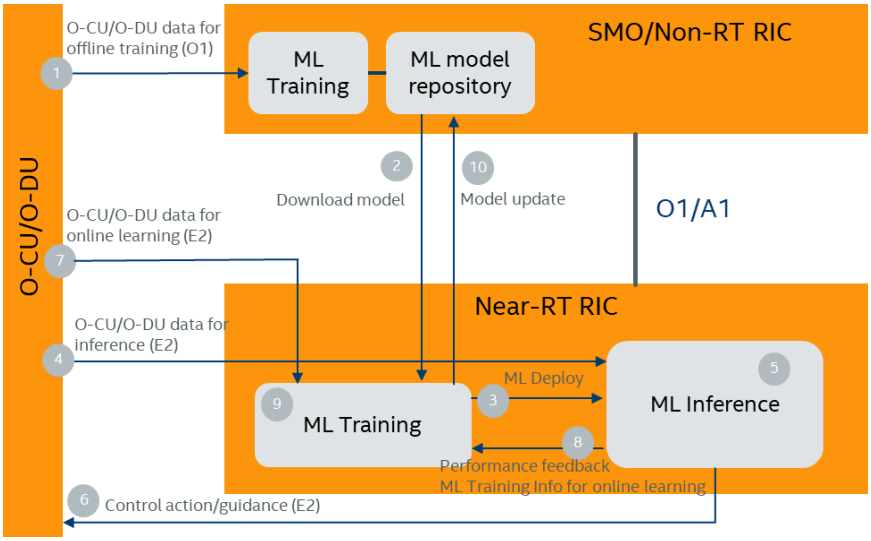


Figure 5-4 - Deployment scenario 1.4 - ML training and inference host locations

In deployment scenario 1.4, the AI/ML model is first offline trained by training host in the SMO/Non-RT RIC. Training host in the Near-RT RIC is used for continuous online learning, and ML inference host is located as part of the Near-RT RIC. ML model repository in the SMO/Non-RT RIC is used to save backup ML training models. When the ML training host in the Near-RT RIC observes severe performance degradation, it can request the stored (previously well-performing) AI/ML model from model repository in the SMO/Non-RT RIC. Based on the retrieved ML model from SMO/Non-RT RIC, the training host in the Near-RT RIC updates the online model using the input data over E2 and from the ML inference host.

The interactions among various functional blocks are described as follows. Note that the learning workflow does not strictly follow the sequence of descriptions.

1. O-CU/O-DU data for offline training is collected over O1 interface and the initial offline model is trained in the SMO/Non-RT RIC.
2. The offline trained model or the backup model is moved to the Near-RT RIC.
3. The AI/ML model is deployed to the ML inference host in the Near-RT RIC.
4. O-CU/O-DU data for inference in the Near-RT RIC is collected over E2 interface.
5. The ML inference host performs inference using the deployed model and collected O-CU/O-DU data.
6. The ML inference host enforces control action/guidance via E2 interface.
7. O-CU/O-DU data for online learning in the Near-RT RIC is collected over E2 interface.
8. The ML inference host provides performance feedback to the ML training host in the Near-RT RIC for monitoring and training data for online learning.
9. The ML training host in the Near-RT RIC updates the model.
10. A well-performing model can be added to the model repository.

Note 1: O-CU/O-DU data collected over O1 for offline training and O-CU/O-DU data collected over E2 for online learning and/or inference can be different.

Note 2: The model online update frequency can be different from the E2 update frequency.

Note 3: Deployment scenario 1.4 is essential to enable online RL in the Near-RT RIC. However, it is not exclusive for online RL.

5.1 Sequence Diagram for Deployment Scenarios 1.1 and 1.2

The sequence diagram (seeFigure 5-5) for Deployment Scenario-1.1 (SMO/Non-RT RIC for model training, Non-RT RIC for model inference host) and Deployment Scenario-1.2 (SMO/Non-RT RIC for model training, Near-RT RIC as model inference host) is captured below

Figure 5-5 - Non-RT RIC as ML training host, Non-RT RIC or Near-RT RIC as ML inference host (Note that the SMO components are defined in WG1 OAM architecture, Appendix B [i.x9])

5.2 Criteria for Determining Deployment Scenario (Options)

This section discusses some criteria that can be used to decide whether an ML Application should execute in a Non-RT RIC or Near-RT RIC Inference Host. It also discusses some criteria for whether an ML Model within such an ML Application should be initially trained (offline learning) in a Non-RT RIC as its Training Host, or in a more centralized or remote location. (Note that subsequent online learning would be expected to occur in the execution environment.) Considerations for these decisions relate to the amount of data that is required by the ML Model both in training as well as in execution, as well as latency considerations in execution. It is assumed in this section that the type of data that an ML Model requires at runtime will also be required, perhaps in aggregated form, during training.

It is assumed that the Near-RT RIC would not be a suitable candidate for offline Training Host. Initial training would typically require a large pool of compute and storage resources. The very nature of the Near-RT RIC as a Network Function is geared towards high performance runtime processing with a small footprint. It is unlikely to have sufficient compute and storage resources to also handle initial offline learning. In addition, offline learning at the Near-RT RIC would also introduce complexity into its design to provide the assurances that ML training would not affect its runtime network function processing. The Non-RT RIC functionality of the SMO, a management (as opposed to a network) function, seems better suited to host initial training.

Regarding the Inference Host decision, criteria include:

1. The availability of data across a given interface. For example, if the data needed at execution time is available only across the E2 interface, then either the ML Application that includes this ML Model must either consider the Near-RT RIC as its Inference Host, or a Near-RT RIC function must be employed to forward across the O1 interface that E2 data to the Non-RT RIC as Inference Host. If the data needed at execution time is available only across the O1 interface, then the ML Application would need to consider the Non-RT RIC as its Inference Host, or a Non-RT RIC function must be employed to forward that O1 data to the Near-RT RIC with its own O1 interface. Clearly some of these options are very inefficient and likely undesirable.
2. The cost of data movement. Obviously the data movement costs of the various choices described in #1 above differ from one another. Ideally, if an ML Model requires E2 data, the Inference Host of its associated ML Application would be the Near-RT RIC. Similarly, if an ML Model requires O1 data, the Inference Host of its associated ML Application would be the Non-RT RIC. If an ML Model requires E2 data but considers the Non-RT RIC to be its Inference Host, then some function within the Near-RT RIC would need to be used as a vehicle to forward that E2 data to the Non-RT RIC. This would obviously be inefficient and costly. However, if the Non-RT RIC is also the Training Host for ongoing learning of that ML Model, then perhaps this inefficiency and cost would be warranted. Less understandable would be using the Non-RT RIC as a vehicle to forward O1 data to the Near-RT RIC acting as Inference Host.
3. Latency considerations. The data latency associated with the various choices described in #1 above differ from one another, and the Loop 2 versus Loop 3 considerations will factor heavily into whether a ML Model consider its Inference Host to be a Near-RT RIC or a Non-RT RIC.
4. Compute resource availability considerations. The Near-RT RIC may run in an “edge” location where compute resources are very limited and hence expensive. The Non-RT RIC may more likely run in a location with more ready access to compute resources, such as a data center.

Regarding the initial Training Host decision, criteria include:

1. The local versus general significance of the data. A data lake will be required in order to train an ML Model. Assuming that a typical Service Provider will have more than one Non-RT RIC instance, if that training data lake coincides with the Non-RT RIC then the data lake will contain data of only local significance. Only by merging this "local data" from many locales into a "central data lake" could future ML applications perhaps find unexpected correlations within data that was incorrectly assumed to be of only local significance.
2. The cost of data movement. While the previous item discussed the benefits of a "central data lake", transporting "local data" to such a central location is not without its costs.

Thus the decision as to whether to use a "local" Non-RT RIC as an initial (offline) Training Host or not in large part depends on the service provider's assessment of the value of that training data for more general purposes. The results of such an assessment can clearly differ from one service provider to another.

Looking at the criteria above, one can see how the decision as to whether the appropriate Inference Host for an ML model is a Non-RT RIC or a Near-RT RIC can reasonably differ between Service Providers. For example, consider two service providers below in their assessment of how to deploy a ML Application "X" that takes large volumes of E2 data to calculate some "enrichment information" regarding UEs (e.g., the UE's predicted QoE on various cells measured on the order of seconds), which can be used by a different "traffic steering" application (perhaps not ML-enabled) to appropriately drive handovers of those UEs. Assume that the "traffic steering" application would run at the Near-RT RIC using E2 mechanisms to drive handovers. Service Provider A and Service Provider B may come to quite different conclusions of how to deploy such an ML Application "X":

	Service Provider A	**Service Provider B**
"X" Data Source	E2	E2
"X" Data Movement Cost	Very High Volume, Very High Cost to Move Data over Distance	Very High Volume, Very High Cost to Move Data over Distance
"X" Latency Consideration	Loop 3 acceptable	Loop 3 acceptable
"X" Training data evaluation	This E2 data is of only local significance.	This E2 data is of potential global significance. Willing to pay to move data to a central data lake.
Pertainent Business Considerations	Minimize data transport costs for data with only local significance. Reserve Near-RT RIC for applications requiring Loop 2 latency	Any application requiring E2 data should run on the Near-RT RIC. Do not use Near-RT RIC as a data relay to the Non-RT RIC.
Inference Host Decision	Because "X" requires only Loop 3 latency, deploy it as an ML Application using the Non-RT RIC as its Inference Host. "X" will communicate with the "traffic steering" application via A1-EI.	Because it requires E2 data, "X" will run as an ML Application using the Near-RT RIC as its Inference Host. "X" will communicate with the "traffic steering" application via Near-RT RIC internal mechanisms.
Runtime E2 Data Movement	RAN -> E2 -> Near-RT RIC -> O1 -> Non-RT RIC	RAN -> E2 -> Near-RT RIC
Initial (Offline) Training Host Decision	Non-RT RIC will be used for training "X" (local training)	A central off-line location will be used for training "X" (training on a central data lake)

Initial (Offline) Training E2 Data Movement	RAN -> E2 -> Near-RT RIC -> O1 -> Non-RT RIC (leverage the Runtime E2 Data Movement path)	RAN -> Offline Data Collection -> Central Data Lake

Because the mechanism of inter-ML application communication (e.g., A1 versus Near-RT RIC internal) is so sensitive to considerations that can reasonably differ among service providers, it will be useful if the mechanism used were only a deployment decision. This should be kept in mind when the A1 and Near-RT RIC internal interface mechanisms are defined. To illustrate this extending the example above, if the mechanism for having the ML model "X" communicate with the "traffic steering" application is through a data structure capturing predicted QoE for a given UE on various cells, this data structure could be communicated either as "enrichment information" via A1-EI or communicated via a common data repository on the Near-RT RIC (such as described in section 4.3). Such an approach could preserve the service provider's ability to make the choice of inference host a deployment consideration that does not require extensive refactoring or retraining of the solution components.

Chapter 6 Requirements

6.1 Functional Requirements

This section describes the functional requirements for A1 interface and Non-RT RIC.

[REQ-Non-RT RIC-FUN1]	Non-RT RIC may request/trigger ML model training in training hosts.

Notes: Regardless of where the model is deployed and executed, non-RT RIC should request/trigger ML model training. Note that ML models may be trained and not currently deployed. Implicitly, model re-training and model performance/evaluation.

[REQ-Non-RT RIC-FUN2]	Non-RT RIC shall provide a query able catalog for ML designer to publish/install trained ML models (executable software components) and Non-RT RIC will provide discovery mechanism if a particular ML model can be executed in the target ML inference host (MF), and what number (and type) of ML models can be executed in the MF.

Notes: Non-RT RIC is a component of the SMO framework, i.e., one component of the NMS. The catalogue is not only for external ML market place or platform to publish the models, but also the source for any internal models as well. Non-RT RIC can also connect to external ML catalogues via SMO specific interfaces (interface specification is not in scope of document). There are three types of catalogs namely (design-time catalog (outside non-RT RIC in other ML platforms), training/deployment-time catalog (inside non-RT RIC), and run-time catalog (inside near-RT RIC for scenario 1.2)). In scenario 1.1 where ML models are trained, deployed and executed in non-RT RIC.

[REQ-Non-RT RIC-FUN3]	Non-RT RIC shall support necessary capabilities (enable executable software to be installed, e.g., containers) for ML model inference in support of ML assisted solutions running in non-RT RIC

Notes: ML engines are packaged s/w executable libraries that provide the necessary routines to run the model.

Note: As an example, policies to switch and activate ML model instances under different operating conditions (busy hour vs non-busy hour or seasonal changes, etc.)

[REQ-Non-RT RIC-FUN4]	Non-RT RIC shall be able to access feedback data over O1 interface on ML model performance and perform necessary evaluation.

Note: PM and FM stats for ML model are relayed over O1. If the ML model fails during runtime an alarm can be generated as feedback to non-RT RIC. How well the ML model is performing in terms of accuracy of prediction or other operating statistics it produces can be sent to non-RT RIC over O1.

Note: The following requirements which apply to both the Non-RT RIC and the Near-RT RIC as ML Inference Host are intended to facilitate a Service Provider's ability to consider it a deployment decision whether an ML Application should be

run on a Non-RT RIC or a Near-RT RIC as its Inference Host. This is in recognition that the criteria for determining the deployment scenario for a given ML Application may differ between service providers. See Section 5.2 for a discussion on the types of criteria that may be weighed differently by different providers

[REQ-Non-RT RIC-FUN5] [REQ-Near-RT RIC-FUN1]	As part of ML Application "registration", the ML Inference Host shall be able to digest information relating the data type(s) and periodicity thereof that the ML Application produces and consumes. This requirement applies both to the Non-RT RIC function and the Near-RT RIC as ML Inference Host.

Note: The following is the corresponding requirement as applied to ML Applications.

[REQ-Non-RT RIC-FUN6] [REQ-Near-RT RIC-FUN2]	ML Applications shall be able to perform "registration" interactions with the ML Inference Host communicating information relating the data type(s) and periodicity thereof that the ML Application produces and consumes. This requirement applies both to those ML Applications that consider the Non-RT RIC function as ML Inference Host, as well as those ML Applications that consider the Near-RT RIC as ML Inference Host.

Note: "Registration" differs from "subscription" in that "registration" involves only data types and their periodicity, whereas "subscription" involves specific sets of data within a given "scope" (see next requirement).

[REQ-Non-RT RIC-FUN7] [REQ-Near-RT RIC-FUN3]	The ML Inference Host will be able to match data consumption needs with data sources. In this respect a data source could be either an ML Application (i.e., another ML Application's data "produced') or the ML Inference Host itself (e.g., mediating an O1-VES data source via the SMO). The ML Inference Host shall consider it a registration-time validation error if no corresponding source can be matched to an ML application's "consumed data" requirements. This requirement applies both to the Non-RT RIC function and the Near-RT RIC as ML Inference Host.

Note: The following requirements referring to ML Inference Host handling of data "subscription requests" are intended to allow ML Applications to share data without knowing each other's data needs (see sections 4.2 and 4.3). This functionality is seen as an enabler for modularity of ML Applications.

[REQ-Non-RT RIC-FUN8] [REQ-Near-RT RIC-FUN4]	The ML Inference Host shall be able to process scoped data "subscription" requests from ML Applications, working with other neighboring (i.e., SMO, Non-RT RIC function, Near-RT RIC Platform) functions as necessary to set up the corresponding data routing (e.g., routing of O1-VES data to the ML Application, routing of one ML Application's produced data to the consuming ML Application). This requirement applies both to the Non-RT RIC function and the Near-RT RIC as ML Inference Host.

Note: An example of "scope" for a data subscription request would be identifying a specific set of gNBs from which to collect O1 data of a particular data type.

[REQ-Non-RT RIC-FUN9] [REQ-Near-RT RIC-FUN5]	For subscription requests that correspond to data produced by another ML Application, the ML Inference Host function will pass the information content thereof (e.g., data type, scope, periodicity) to that other ML Application for processing. This requirement applies both to the Non-RT RIC function and the Near-RT RIC as ML Inference Host.

[REQ-Non-RT RIC-FUN10] [REQ-Near-RT RIC-FUN6]	ML Applications that produce data shall be able to interact with the ML Inference Host to receive and process scoped subscription requests. The ML Application will be responsible for determining and generating to the ML Inference Host any additional subscription requests needed to produce the requested data. This requirement applies both to those ML Applications that consider the Non-RT RIC function as ML Inference Host, as well as those ML Applications that consider the Near-RT RIC as ML Inference Host.

Note: It is the responsibility of the ML Application to ensure that the periodicity and scope of these "consumed data" subscription request corresponds to that ML Application's needs in producing the requested data, as described in the scoped subscription request that it received from the Inference Host.

[REQ-Non-RT RIC-FUN11] [REQ-Near-RT RIC-FUN7]	The ML Inference Host shall be able to provide data mediation functionality such that, if two separate ML Applications request the same ML Application-produced data, the ML Inference Host will split the data feed without placing a burden on the source ML Application. This requirement applies both to the Non-RT RIC function and the Near-RT RIC as ML Inference Host.

[REQ-Non-RT RIC-FUN12]	Non-RT RIC shall have access to termination conditions, and it shall be able to compute related performance metrics specified in termination conditions, compare performance monitoring results against termination conditions, and make judgement that whether termination procedure should be triggered.

[REQ-Non-RT RIC-FUN13] [REQ-Near-RT RIC-FUN8]	The ML inference host shall be able to receive ML model termination command, identify the problematic model according to the request, and terminate it accordingly. ML inference host shall be able to receive backup solution activation command, and switch to the backup solution following the instruction. ML inference host shall notify SMO/Non-RT RIC about the outcome of ML model termination and the activation of backup solution via acknowledgement messages.

REQ	Description
[REQ-O1-FUN1]	O1 interface shall support deployment and update of the ML models as a packaged s/w executable (e.g., in a container).
[REQ-O1-FUN2]	O1 interface shall support file based ML model deployment and updates.
[REQ-O1-FUN3]	O1 interface shall support PM and FM data collection for ML models, including fine-grained events/counters needed for ML training and inference.
[REQ-O1-FUN4]	O1 interface shall support collection of ML relevant capabilities of the managed function where the model is to be deployed for inference.

6.2 Non-Functional Requirements

This section describes the non-functional requirements for A1 interface and Non-RT RIC, e.g., security.

REQ	Description
[REQ-O1-NONFUN1]	O1 interface shall support scaling ML model instances running in target ML inference host (MF) by observing resource utilization in MF.

Note: The environment where the ML model instance is running will monitor resource utilization (e.g., in O-RAN-SC there is a component call Resource Monitor in near-RT RIC; similarly, in non-RT RIC there needs to be a Resource Monitor that continuously monitors resource utilization). If resources are low or fall below a certain threshold, the runtime environment in near-RT RIC and non-RT RIC needs to provide a scaling mechanism to add more ML instances. K8s runtime environments typically provide auto-scaling feature.

REQ	Description
[REQ-O1-NONFUN2]	ML model instances running in target ML inference hosts shall be automatically scaled by observing resource utilization in MF.

Chapter 7 Key Issues

7.1 AI/ML Models in O-RAN Use Cases

In [4] multiple AI/ML assisted use cases are discussed. This section summarizes the examples of the AI/ML models used in the O-RAN use cases.

Use Case	AI/ML models functionality description	AI/ML algorithms types (example)	Deploy scenarios mapping	Data Input	Data Output
QoE Optimization	service type classification	Supervised learning (e.g., CNN, DNN)	Scenario 1.2	user traffic data	service type
	KQI/QoE prediction (e.g., good, bad or video stall ratio, duration)	Supervised learning (e.g, LSTM, XGboost)	Scenario 1.2	Network data: L2 measurement report related to traffic pattern, e.g., throughput, latency, packets per-second UE level radio channel information, mobility related metrics RAN protocol stack status: e.g. PDCP buffer status Cell level information: e.g. DL/UL PRB occupation rate Application data: e.g., video QoE score, video initial delay	KQI/QoE value e.g., good/bad, stalling ratio, video stalling duration, vMoS value

				stalling detail including the timestamp stalling duration, stalling ratio,	
	Available radio bandwidth prediction	Supervised learning (e.g., DNN)	Scenario 1.2	similar to above	Available radio Bandwidth
Traffic Steering	cell load prediction/user traffic volume prediction	Supervised learning (time series prediction, e.g., SVR, DNN)	Scenario 1.1 / Scenario1.2	load related counters, e.g., UL/ DL PRB occupation	same to input
	Radio finger print prediction	supervised learning (e.g., SVR, GBDT)	Scenario 1.2	Intra-frequency MR data and PM counters, e.g., RSRP, RSRQ, MCS, CQI, etc.	Inter-frequency MR data, e.g, RSRP, RSRQ, MCS, CQI, etc.
QoE based Traffic Steering	generate relevant A1 policies to provide guidance on the traffic steering preferences	FFS	Scenario 1.1	FFS	priority order of the cells to be used for downlink data transmission.
	time-series prediction of individual performance metrics or counters	Supervised learning (e.g. lasso regression-based prediction model)	Scenario 1.2	FFS	FFS
	QoE prediction at each neighbor cell for a given targeted user	Supervised learning (e.g., binary classification	Scenario 1.2	FFS	QoE good/bad

		model using Random Forest)			
V2X Handover Management	✓ prediction / detection HO anomalies ✓ discovery of preferred HO sequences	Supervised learning	Scenario 1.2	CAM,,radio cell IDs, connection IDs, and basic radio measurements (RSRP, RSPQ etc.) GPS, direction, velocity	✓ HO anomalies probability ✓ preferred HO sequences

Note: CAM stands for Cooperative Awareness Message, as defined in [6]. It originates from the vehicular UE and terminates in the V2X App Server. It contains the GPS coordinates of the vehicle to a 0.1-1s granularity.

7.2 AI/ML Model Deployment

In deployment scenario 1.2, Non-RT RIC acts as the ML model training host and near-RT RIC acts as the ML model inference host. ML/AI model can be deployed and enabled in Near-RT RIC in different options:

1. Image based deployment: The AI/ML model will be deployed as a xAPP or within a xAPP instance, and also can be updated via the xAPP software update. In this case, the ML inference runtime is self-contained in the image, which simplifies the deployment. This is a generic deployment of ML xAPP – in the sense that ML xAPP is treated no differently from other types of xAPP.

2. File based deployment: The AI/ML model can be deployed based on the AI/ML model file, which is generally decoupled with the xAPP software version, and can be enabled and updated via the xAPP file configuration. For this scenario, a ML model catalog and an inference platform/engine are usually required for the ML model inference host (i.e., Near-RT RIC). The Near-RT RIC may have an unified inference platform/engine, which can help to accelerate the inference efficiency (exploiting native ML capabilities of the platform) and enable greater customization but which requires the ML model file format to be supported by the inference platform.

Figure 7-1 illustrates an example of image based vs file based ML model deployments.

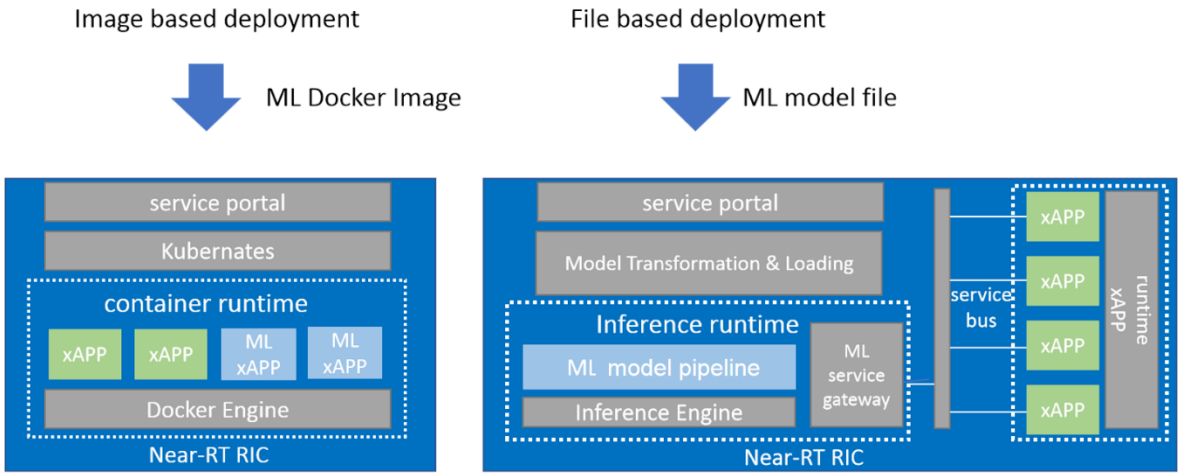


Figure 7-1 - Examples of image based and file based ML model deployment

Notes: The above figure only shows examples of how Near-RT RIC works under the image based ML model deployment and the file based ML model deployment. It may have different Near-RT RIC internal implementation.

Table 3 compares the Pros and Cons of the two approaches.

Table 3 - Pros and Cons of the image based and file based ML model deployment

Options	Pros	Cons
Opt1: Image Based ML model deployment	Faster and flexible deployment. Less requirements on the ML model inference host, i.e., Near-RT RIC, except for support of container runtime env.	The inference efficiency depends on the container capability.
Opt2: File Based ML model deployment	Better customization and efficiency by exploiting the on-device model optimization and update capabilities. Potential use of standard file formats for ML models.	Additional function requirement for the ML model inference host. Requires the matching of the ML model format and the inference engine.

Annex A (Informative)

A.1 Discussion on A1/O1 clarification

The following table tries to summarize WG2 involved information exchange over O1 and A1 interface based on the UCR doc and AI/ML workflow discussion.

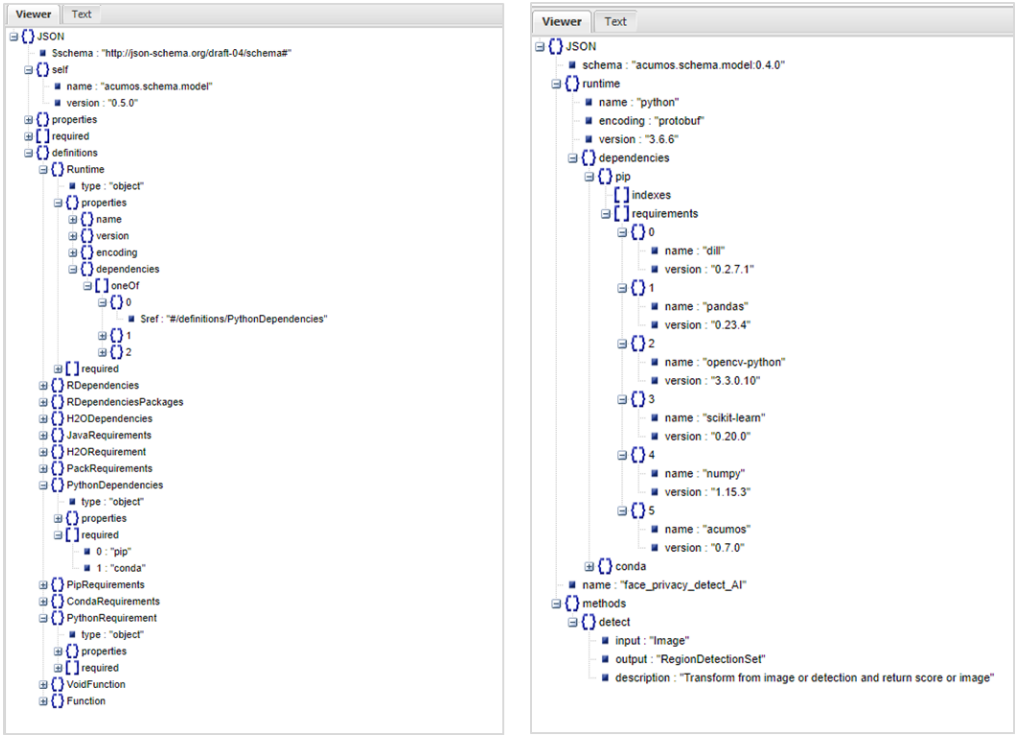
Table 4 - A1 vs O1 information exchange

Information	Interface	Management Services	Remarks
Policy	A1		
Enrichment information	A1		
Policy feedback	A1		Feedback for model state
Non-RT RIC performance data collection	O1	Performance measurement	SMO internal interface to access O1 data
Non-RT RIC Fault data collection	O1	Fault measurement	SMO internal interface to access O1 data
Network parameter configuration	O1	Provisioning management	O1-CM
AI/ML model deployment	O1	Software management	
AI/ML model update	O1	Software management	Containerized, same for xApp update/revision control as per OAM
AI/ML model performance monitoring	O1		Enhancement is needed. How to model the AI/ML in the information model needs further study.

A.2 Examples of ML model capabilities/descriptors

- ML capabilities may include performance aspects of the target network function (e.g. CPU, memory, etc.), support for ML engines and supported libraries.

- These capabilities need to be matched against an ML model descriptor to decide whether a model can be deployed in the target network function.



-

Figure A.2 1 - Example ML model descriptor schema

Figure A.2 1 **エラー! 参照元が見つかりません。** provides an example schema for ML models and an illustration for a face_privacy_detection use case. It shows the input/output mapping and a set of ML model runtime dependencies.

Annex N (Informative)

N.1 Appendix X

The following figure shows the comparison of the inference performance of the original TensorFlow models and the compiled models running in different inference framework. (The model compilation and optimization, and inference engine are provided by Adlik [7].)

Model	Inference Performance improved after Adlik Optimized	
	CPU	GPU
ResNet50	397%	422%
VGG16	360%	214%

Figure N.1 1 - Comparison of inference performance in different inference framework with Adlik toolkit

Inference performance is measured by pcs/s (processed pictures per second).

Test bed：Batch size: 64; CPU: 2; Memory: 8G; GPU: 1*NVIDIA P100

Annex Z: O-RAN Adopter License Agreement

O-RAN ADOPTER LICENSE AGREEMENT

BY DOWNLOADING, USING OR OTHERWISE ACCESSING ANY O-RAN SPECIFICATION, ADOPTER AGREES TO THE TERMS OF THIS AGREEMENT.

This O-RAN Adopter License Agreement (the "Agreement") is made by and between the O-RAN Alliance and the entity that downloads, uses or otherwise accesses any O-RAN Specification, including its Affiliates (the "Adopter").

This is a license agreement for entities who wish to adopt any O-RAN Specification.

SECTION 1: DEFINITIONS

1.1 "Affiliate" means an entity that directly or indirectly controls, is controlled by, or is under common control with another entity, so long as such control exists. For the purpose of this Section, "Control" means beneficial ownership of fifty (50%) percent or more of the voting stock or equity in an entity.

1.2 "Compliant Portion" means only those specific portions of products (hardware, software or combinations thereof) that implement any O-RAN Specification.

1.3 "Adopter(s)" means all entities, who are not Members, Contributors or Academic Contributors, including their Affiliates, who wish to download, use or otherwise access O-RAN Specifications.

1.4 "Minor Update" means an update or revision to an O-RAN Specification published by O-RAN Alliance that does not add any significant new features or functionality and remains interoperable with the prior version of an O-RAN Specification. The term "O-RAN Specifications" includes Minor Updates.

1.5 "Necessary Claims" means those claims of all present and future patents and patent applications, other than design patents and design registrations, throughout the world, which (i) are owned or otherwise licensable by a Member, Contributor or Academic Contributor during the term of its Member, Contributor or

Academic Contributorship; (ii) such Member, Contributor or Academic Contributor has the right to grant a license without the payment of consideration to a third party; and (iii) are necessarily infringed by implementation of a Final Specification (without considering any Contributions not included in the Final Specification). A claim is necessarily infringed only when it is not possible on technical (but not commercial) grounds, taking into account normal technical practice and the state of the art generally available at the date any Final Specification was published by the O-RAN Alliance or the date the patent claim first came into existence, whichever last occurred, to make, sell, lease, otherwise dispose of, repair, use or operate an implementation which complies with a Final Specification without infringing that claim. For the avoidance of doubt in exceptional cases where a Final Specification can only be implemented by technical solutions, all of which infringe patent claims, all such patent claims shall be considered Necessary Claims.

1.6 "Defensive Suspension" means for the purposes of any license grant pursuant to Section 3, Member, Contributor, Academic Contributor, Adopter, or any of their Affiliates, may have the discretion to include in their license a term allowing the licensor to suspend the license against a licensee who brings a patent infringement suit against the licensing Member, Contributor, Academic Contributor, Adopter, or any of their Affiliates.

SECTION 2: COPYRIGHT LICENSE

2.1 Subject to the terms and conditions of this Agreement, O-RAN Alliance hereby grants to Adopter a non-exclusive, non-transferable, irrevocable, non-sublicensable, worldwide copyright license to obtain, use and modify O-RAN Specifications, but not to further distribute such O-RAN Specification in any modified or unmodified way, solely in furtherance of implementations of an O-RAN Specification

2.2 Adopter shall not use O-RAN Specifications except as expressly set forth in this Agreement or in a separate written agreement with O-RAN Alliance.

SECTION 3: FRAND LICENSE

3.1 Members, Contributors and Academic Contributors and their Affiliates are prepared to grant based on a separate Patent License Agreement to each Adopter under Fair, Reasonable And Non-Discriminatory (FRAND) terms and conditions with or without compensation (royalties) a nonexclusive, non-transferable, irrevocable (but subject to Defensive Suspension), non-sublicensable, worldwide license under their Necessary Claims to make, have made, use, import, offer to sell, lease, sell and otherwise distribute Compliant Portions; provided, however, that such license shall not extend: (a) to any part or function of a product in which a Compliant Portion is incorporated that is not itself part of the Compliant Portion; or (b) to any Adopter if that Adopter is not making a reciprocal grant to Members, Contributors and Academic Contributors, as set forth in Section 3.3. For the avoidance of doubt, the foregoing license includes the distribution by the Adopter's distributors and the use by the Adopter's customers of such licensed Compliant Portions.

3.2 Notwithstanding the above, if any Member, Contributor or Academic Contributor, Adopter or their Affiliates has reserved the right to charge a FRAND royalty or other fee for its license of Necessary Claims to Adopter, then Adopter is entitled to charge a FRAND royalty or other fee to such Member, Contributor or Academic Contributor, Adopter and its Affiliates for its license of Necessary Claims to its licensees.

3.3 Adopter, on behalf of itself and its Affiliates, shall be prepared to grant based on a separate Patent License Agreement to each Members, Contributors, Academic Contributors, Adopters and their Affiliates under FRAND terms and conditions with or without compensation (royalties) a nonexclusive, non-transferable, irrevocable (but subject to Defensive Suspension), non-sublicensable, worldwide license under their Necessary Claims to make, have made, use, import, offer to sell, lease, sell and otherwise distribute Compliant Portions; provided, however, that such license will not extend: (a) to any part or function of a product in which a Compliant Portion is incorporated that is not itself part of the Compliant Portion; or (b) to any Members, Contributors, Academic Contributors, Adopters and their Affiliates that is not making a reciprocal grant to Adopter, as set forth in Section 3.1. For the avoidance of doubt, the foregoing license includes the distribution by the Members', Contributors', Academic Contributors', Adopters' and their Affiliates' distributors and the use by the Members', Contributors', Academic Contributors', Adopters' and their Affiliates' customers of such licensed Compliant Portions.

SECTION 4: TERM AND TERMINATION

4.1 This Agreement shall remain in force, unless early terminated according to this Section 4.

4.2 O-RAN Alliance on behalf of its Members, Contributors and Academic Contributors may terminate this Agreement if Adopter materially breaches this Agreement and does not cure or is not capable of curing such breach within thirty (30) days after being given notice specifying the breach.

4.3 Sections 1, 3, 5 - 11 of this Agreement shall survive any termination of this Agreement. Under surviving Section 3, after termination of this Agreement, Adopter will continue to grant licenses (a) to entities who become Adopters after the date of termination; and (b) for future versions of O-RAN Specifications that are backwards compatible with the version that was current as of the date of termination.

SECTION 5: CONFIDENTIALITY

Adopter will use the same care and discretion to avoid disclosure, publication, and dissemination of O-RAN Specifications to third parties, as Adopter employs with its own confidential information, but no less than reasonable care. Any disclosure by Adopter to its Affiliates, contractors and consultants should be subject to an obligation of confidentiality at least as restrictive as those contained in this Section. The foregoing obligation shall not apply to any information which is: (1) rightfully known by Adopter without any limitation on use or disclosure prior to disclosure; (2) publicly available through no fault of Adopter; (3) rightfully received without a duty of confidentiality; (4) disclosed by O-RAN Alliance or a Member, Contributor or Academic Contributor to a third party without a duty of confidentiality on such third party; (5) independently developed by Adopter; (6) disclosed pursuant to the order of a court or other authorized governmental body, or as required by law, provided that Adopter provides reasonable prior written notice to O-RAN Alliance, and cooperates with O-RAN Alliance and/or the applicable Member, Contributor or Academic Contributor to have the opportunity to oppose any such order; or (7) disclosed by Adopter with O-RAN Alliance's prior written approval.

SECTION 6: INDEMNIFICATION

Adopter shall indemnify, defend, and hold harmless the O-RAN Alliance, its Members, Contributors or Academic Contributors, and their employees, and agents and their respective successors, heirs and assigns (the "Indemnitees"), against any liability, damage, loss, or expense (including reasonable attorneys' fees and expenses) incurred by or imposed upon any of the Indemnitees in connection with any claims, suits, investigations, actions, demands or judgments arising out of Adopter's use of the licensed O-RAN Specifications or Adopter's commercialization of products that comply with O-RAN Specifications.

SECTION 7: LIMITATIONS ON LIABILITY; NO WARRANTY

EXCEPT FOR BREACH OF CONFIDENTIALITY, ADOPTER'S BREACH OF SECTION 3, AND ADOPTER'S INDEMNIFICATION OBLIGATIONS, IN NO EVENT SHALL ANY PARTY BE LIABLE TO ANY OTHER PARTY OR THIRD PARTY FOR ANY INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES RESULTING FROM ITS PERFORMANCE OR NON-PERFORMANCE UNDER THIS AGREEMENT, IN EACH CASE WHETHER UNDER CONTRACT, TORT, WARRANTY, OR OTHERWISE, AND WHETHER OR NOT SUCH PARTY HAD ADVANCE NOTICE OF THE POSSIBILITY OF SUCH DAMAGES.

O-RAN SPECIFICATIONS ARE PROVIDED "AS IS" WITH NO WARRANTIES OR CONDITIONS WHATSOEVER, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. THE O-RAN ALLIANCE AND THE MEMBERS, CONTRIBUTORS OR ACADEMIC CONTRIBUTORS EXPRESSLY DISCLAIM ANY WARRANTY OR CONDITION OF MERCHANTABILITY, SECURITY, SATISFACTORY QUALITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, ERROR-FREE OPERATION, OR ANY WARRANTY OR CONDITION FOR O-RAN SPECIFICATIONS.

SECTION 8: ASSIGNMENT

Adopter may not assign the Agreement or any of its rights or obligations under this Agreement or make any grants or other sublicenses to this Agreement, except as expressly authorized hereunder, without having first received the prior, written consent of the O-RAN Alliance, which consent may be withheld in O-RAN Alliance's sole discretion. O-RAN Alliance may freely assign this Agreement.

SECTION 9: THIRD-PARTY BENEFICIARY RIGHTS

Adopter acknowledges and agrees that Members, Contributors and Academic Contributors (including future Members, Contributors and Academic Contributors) are entitled to rights as a third-party beneficiary under this Agreement, including as licensees under Section 3.

SECTION 10: BINDING ON AFFILIATES

Execution of this Agreement by Adopter in its capacity as a legal entity or association constitutes that legal entity's or association's agreement that its Affiliates are likewise bound to the obligations that are applicable to Adopter hereunder and are also entitled to the benefits of the rights of Adopter hereunder.

SECTION 11: GENERAL

This Agreement is governed by the laws of Germany without regard to its conflict or choice of law provisions.

This Agreement constitutes the entire agreement between the parties as to its express subject matter and expressly supersedes and replaces any prior or contemporaneous agreements between the parties, whether written or oral, relating to the subject matter of this Agreement.

Adopter, on behalf of itself and its Affiliates, agrees to comply at all times with all applicable laws, rules and regulations with respect to its and its Affiliates' performance under this Agreement, including without

limitation, export control and antitrust laws. Without limiting the generality of the foregoing, Adopter acknowledges that this Agreement prohibits any communication that would violate the antitrust laws.

By execution hereof, no form of any partnership, joint venture or other special relationship is created between Adopter, or O-RAN Alliance or its Members, Contributors or Academic Contributors. Except as expressly set forth in this Agreement, no party is authorized to make any commitment on behalf of Adopter, or O-RAN Alliance or its Members, Contributors or Academic Contributors.

In the event that any provision of this Agreement conflicts with governing law or if any provision is held to be null, void or otherwise ineffective or invalid by a court of competent jurisdiction, (i) such provisions will be deemed stricken from the contract, and (ii) the remaining terms, provisions, covenants and restrictions of this Agreement will remain in full force and effect.

Any failure by a party or third party beneficiary to insist upon or enforce performance by another party of any of the provisions of this Agreement or to exercise any rights or remedies under this Agreement or otherwise by law shall not be construed as a waiver or relinquishment to any extent of the other parties' or third party beneficiary's right to assert or rely upon any such provision, right or remedy in that or any other instance; rather the same shall be and remain in full force and effect.