ITUPublications

International Telecommunication Union
Standardization Sector

ITU-T Technical Specification

(07/2020)

ML5G-DEL1

Serving framework for ML models in future networks including IMT-2020

Technical Specification ITU-T ML5G-DEL1

Serving framework for ML models in future networks including IMT-2020

Summary

This Technical Specification describes a serving framework for machine learning (ML) models in future networks including IMT-2020. The Technical Specification includes requirements and architecture components for such a framework.

Keywords

IMT-2020, machine learning, marketplace, model, optimization, serving framework.

Note

This is an informative ITU-T publication. Mandatory provisions, such as those found in ITU-T Recommendations, are outside the scope of this publication. This publication should only be referenced bibliographically in ITU-T Recommendations.

Editor: Slawomir Stanczak
Fraunhofer HHI Germany
E-mail:
slawomir.stanczak@hhi.fraunhofer.de

© ITU 2026

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Table of Contents

Technical Specification ITU-T ML5G-DEL1

Serving framework for ML models in future networks including IMT-2020

1 Scope

This Technical Specification provides the mechanism for serving machine learning (ML) models in telecommunication networks to enable machine learning in future networks including IMT-2020.

The scope of this Technical Specification includes:

– Background and motivation

– High level requirements for the serving framework

– Architecture for serving models in IMT-2020 networks.

2 References

[ITU-T Y.3172] Recommendation ITU-T Y.3172 (2019), *Architectural framework for machine learning in future networks including IMT-2020*.

[ITU-T Y.3174] Recommendation ITU-T Y.3174 (2020), *Framework for data handling to enable machine learning in future networks including IMT-2020*.

[ITU-T Y.3176] Recommendation ITU-T Y.3176 (2020), *Machine learning marketplace integration in future networks including IMT-2020*.

3 Definitions

3.1 Terms defined elsewhere

This Technical Specification uses the following terms defined elsewhere:

3.1.1 machine learning sandbox [ITU-T Y.3172]: An environment in which machine learning models can be trained, tested and their effects on the network evaluated.

3.1.2 machine learning marketplace [ITU-T Y.3176]: A component which provides capabilities facilitating the exchange and delivery of machine learning models among multiple parties.

NOTE 1 – A network operator may use a machine learning marketplace deployed internally and/or externally to the network operator's administrative domains. Internal and external marketplace differ only in the deployment perspective.

NOTE 2 – A marketplace which is internal to a network operator may act as an external marketplace to another network operator and vice versa.

3.2 Terms defined in this Technical Specification

This Technical Specification defines the following terms:

3.2.1 model inference: The process by which the deployed machine learning (ML) model generates a result.

NOTE – Examples of results generated by an ML model are prediction or classification.

3.2.2 model serving: The process by which a machine learning (ML) model becomes integrated with other ML pipeline nodes and is ready for executing inference in the ML underlay.

3.2.3 inference optimization: Optimization that can be performed on a trained model for better performance during inference.

3.2.4 serving engine: The functionality that provides a runtime for the machine learning (ML) model and exposes corresponding model inference service.

4 Abbreviations and acronyms

This Technical Specification uses the following abbreviations and acronyms:

AN Access Network

API Application Programming Interface

AUC Area Under Curve

CN Core Network

MEC Mobile Edge Computing

ML Machine Learning

MLaaS Machine Learning as a Service

MLDB Machine Learning Database

MLFO Machine Learning Function Orchestrator

MSE Mean Squared Error

NF Network Function

5 Conventions

In this Technical Specification, requirements are classified as follows:

- The keywords "**is required to**" indicate a requirement which must be strictly followed and from which no deviation is permitted if conformance to this document is to be claimed.
- The keywords "**is recommended**" indicate a requirement which is recommended but which is not absolutely required. Thus, such requirements need not be present to claim conformance.
- The keywords "can optionally" and "**may**" indicate an optional requirement which is permissible, without implying any sense of being recommended. These terms are not intended to imply that the vendor's implementation must provide the option and the feature can be optionally enabled by the NOP/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformance with the specification.

6 Introduction

The use cases for machine learning (ML) in future networks including IMT-2020 were described in [b-ITU-T Sup.55]. An architectural framework for ML in future networks including IMT-2020 was presented in [ITU-T Y.3172]. Integration of ML marketplaces with future networks including IMT-2020 is described in [ITU-T Y.3176]. However, there are still some challenges to be addressed for model serving in future networks:

- How to enable interoperable optimization mechanisms for ML models for heterogeneous hardware environment?

Current ML marketplaces, such as the ones available in open-source [b-LF Acumos], provide unified containerized solutions which wrap the trained models directly without optimization considerations for the deployment environment. Performing inference with training framework lacks efficiency due to its high computational complexity and redundant parameters, etc. Furthermore, the diverse inputs and constraints from varied ML underlay networks may require specific optimization considerations.

- How to ensure flexible deployment of ML models for different use case scenarios?

Deployment requirements of ML models in network operators' networks may be specific to the platforms, levels used in the network, as well as the use case requirements. Although existing deployment methods consider general containerized deployment, e.g., deployment on Kubernetes [b-Kubernetes] platforms with container, these inbuilt mechanisms are not flexible and portable enough to include consideration of network operators' deployment preferences.

NOTE 1 – For example, network operator may prefer deployment of internal ML marketplace implemented using Acumos or external ML marketplace implemented by third party vendors. Each marketplace may support deployment types e.g., Kubernetes cluster. But network operator may need to deploy part of the optimized ML model in the access network (AN) for real time inferences, and the training part may be implemented in the core network (CN). These flexible choices are needed to help to migrate ML models to various hardware environment, e.g., to environment with limitations in computing power and strict latency requirements.

- How to provide effective interaction between the serving model and other components of the ML pipeline?

NOTE 2 – A serving model is a ML model which is deployed and ready for performing inference in the network.

For some ML pipelines where ML model is embedded locally in a ML application, e.g., in AN, the interaction is implementation specific. While in other cases, model inference is exposed to the network as web services (MLaaS). A standardized mechanism for serving these ML models on the centralized platform is needed to ensure the workflow of the ML pipeline.

This document specifies a serving framework for ML models in future networks including IMT-2020 to address the above challenges by considering three fundamental stages in model serving, which are inference optimization, model deployment and model inference.

A model optimizer is introduced in the serving framework for inference optimization, used to adjust the trained models for better performance when executing inference on a certain platform according to the requirements of the use case and the current state of the network. This is essential to achieve better performance, e.g., improved latency, throughput, power efficiency and memory consumption. Inference optimization may be achieved by both model-targeted optimization techniques, e.g., pruning, quantization, structural compression, graph conversion and layer fusion, as well as hardware specific ones, e.g., compiler acceleration, parallel computing acceleration.

NOTE 3 – For example: the metadata generated from the training and usage of ML models in the network operators' networks may be used for inference optimization of the ML models.

The inference engine builder, another entity in serving framework, deals with the model deployment stage, aiming to get the model ready to run on a specific platform.

After the deployment completes, the serving platform, i.e., the platform where the model is deployed, provides model inference service that can be consumed by ML pipelines.

Based on the requirements for model serving in these three stages, an architecture for such a serving framework including reference points and sequence diagrams is proposed in the document, aimed at providing an agile mechanism to serve a ML in future network including IMT-2020.

7 High-level requirements for Serving framework for ML models in future networks including IMT-2020

7.1 High-level requirements for Serving Framework in inference optimization stage

This clause describes high-level requirements for serving framework in inference optimization of ML models before models are deployed in ML pipeline subsystem.

REQ-ML-OPT-001: Inference optimization is required to be determined based on the deployment environment in the network.

NOTE 1 – Examples of deployment environment include real time platforms in AN and cloud environment in the CN.

REQ-ML-OPT-002: Inference optimization is required to be determined based on the characteristics and current status of the ML model.

NOTE 2 – Examples of related characteristics of the ML model are type of the learning, ML framework and structure of the neural network (NN).

NOTE 3 – Examples of current status of the ML model are current performance metrics.

NOTE 4 – Current monitored performance reports of the models may be correlated with the type of data handled by them to decide the type of optimization to perform (e.g., pruning of decision trees).

REQ-ML-OPT-003: Inference optimization is recommended to upload the optimized models along with the corresponding optimization information to the ML marketplace.

NOTE 5 – The optimized models can be reused in other ML pipeline subsystem with similar hardware environment. Optimization information helps in matching similar runtime environments.

NOTE 6 – Examples of optimization information are the target environment, changes to the original model and effects of the optimization.

7.2 High-level requirements for Serving Framework in model deployment stage

This clause describes high-level requirements for serving framework in model deployment, i.e., considerations while deploying a model on a serving platform.

REQ-ML-DEP-001: Model deployment, testing and evaluation are required to be done in the ML Sandbox before the deployment in the ML pipeline in the serving engine.

REQ-ML-DEP-002: Model deployment is required to select the runtime environment based on the resource requirements of the target models and the current status of the resources of the ML underlay.

NOTE 1 – For example, some models may require GPU to achieve its desired performance while some may be optimized for a specific platform e.g., CPU, FPGA.

NOTE 2 – An example of current status of resources of the underlay network is the current memory available for use in the ML underlay. The deployment should prepare a ML model with size less than the memory available for use in the ML underlay.

REQ-ML-DEP-003: Model deployment is required to consider the use case requirements as mentioned in the ML intent while selecting the deployment options.

NOTE 3 – The use-case specification may include latency and throughput requirements, performance requirements and location requirements. Model performance metrics may include precision, recall, area under curve (AUC) for classification models and mean squared error (MSE) for regression models. It may also include how the prediction service is provided, e.g., in the form of web service or real-time streaming analytics in which case the input data would be a stream of events triggering the prediction.

NOTE 4 – The deployment options include the level of network in which the ML model is deployed and the runtime environment for the ML model.

REQ-ML-DEP-004: Operations in model deployment is recommended to be time synchronized with other events in the underlying network.

NOTE 5 – Examples of other events in the network are upgrading, scaling and maintenance of network functions (NFs) in the underlying network.

REQ-ML-DEP-005: Model deployment is required to consider the state of the ML models and that of the underlying network along with network operator preferences and policies.

NOTE 6 – Examples of state of ML models are size of the model and performance metrics evaluated in the ML sandbox.

NOTE 7 – Example of network operator preferences and policies is the timing of update of ML models in the ML pipeline subsystem.

NOTE 8 – Examples of model deployment decision are distributed or centralized. E.g., certain ML models may be deployed as multi-layered, with certain parts in the AN and other parts in the CN, where as some ML models may be centrally deployed in the core network.

NOTE 9 – Examples of the state of underlying network are the capabilities of the NFs and the capability exposure to be supported by the NFs, e.g., UPF may need a capability for feature extraction and executing of ML models, e.g., the mobile edge computing (MEC) may need capability for updating ML models based on optimizations identified by the machine learning function orchestrator (MLFO).

REQ-ML-DEP-005: ML model deployment is required to consider the configurations of nodes in ML pipeline while preparing the deployment of ML models into the ML pipeline.

7.3 High-level requirements for serving framework in model inference stage

This clause describes requirements for model inference stage on MLaaS-based Serving Platforms.

REQ-ML-DEP-001: Model inference is required to execute in coordination with other components of ML pipeline.

NOTE 1 – Examples of the other components of ML pipeline are the data collector configured to collect input data, a pre-processor allocated to process the data before inputting the data to the deployed model and the sink determined to receive the ML output [ITU-T Y.3172].

REQ-ML-DEP-002: Model inference is required to support version management of serving models.

NOTE 2 – Example – reword later – m1v1 and m1v2, while inference, can we select m1v1 or m1v2 based on some criteria.

REQ-ML-DEP-003: Model inference is required to enable monitoring and evaluation of the performance of ML models on an ongoing basis.

NOTE 3 – Example of monitoring results is whether the models are experiencing performance degradation over time. Inputs, outputs and exceptions of each inference request may be stored and used for such analysis.

REQ-ML-DEP-004: Model inference is recommended to support sending notifications to authorized management services when the model performance deteriorates to a certain threshold indicated by each use case.

NOTE 4 – MLFO is an example of an authorized management service.

REQ-ML-DEP-005: Model inference is recommended to expose capabilities, e.g., to stop, restart, update and delete the serving models based on messages from authorized management services.

NOTE 5 – MLFO is an example of an authorized management service.

REQ-ML-DEP-006: Model inference is required to expose profiles of deployed models to the network so that models can be discovered by model consumers to compose an ML pipeline.

NOTE 6 – Profile of a deployed model may include its current status as well as metadata which are defined in [ITU-T Y.3176].

REQ-ML-DEP-007: Model inference is required to support sending any changes or updates in the nature of a model's input to an authorized management service to indicate the changed data collection requirements.

NOTE 7 – Examples of changes in the nature of a ML model input are updates in model input caused by model optimizations.

REQ-ML-DEP-008: Model inference is recommended to enable updates of models by online learning, i.e., to improve the performance of model continuously with the sequential arrival of data, see [b-Online learning], for ML use cases where data characteristics change dramatically over time.

REQ-ML-DEP-009: Model inference is recommended to enable scalability, i.e., enable auto/manual scale of the resources allocated for the model.

NOTE 8 – Examples of parameters for scaling are volume of input data and performance thresholds of the use case.

NOTE 9 – The decision to scale up/down may be taken by an authorized management function (e.g., MLFO).

REQ-ML-DEP-010: Model inference is recommended to enable load balancing capability between multiple instances of a model.

REQ-ML-DEP-011: Model inference is required to support request forwarding to the model instances deployed in the serving engine.

REQ-ML-DEP-012: Model inference is required to support different serving types according to the requirements of each use case.

NOTE 10 – Examples of serving types are online prediction and batch prediction. Online prediction is useful for use cases with high latency requirements, which refers to real-time prediction where data generated from the network is sent to the model for inference immediately, and batch prediction refers to the prediction for a high volume of data per inference request.

REQ-ML-DEP-013: Model inference is required to support fault management.

REQ-ML-DEP-014: Model inference is required to consider security, accounting/licensing requirements when it's hosted on public cloud environment, e.g., Google cloud.

8 Architecture and APIs for serving of ML models in telecommunication network

8.1 Functional components of serving framework

The functional components of the serving framework are shown in Figure 1. This is a simplified view of the main functional components along with external functionalities needed to serve a model in the network.

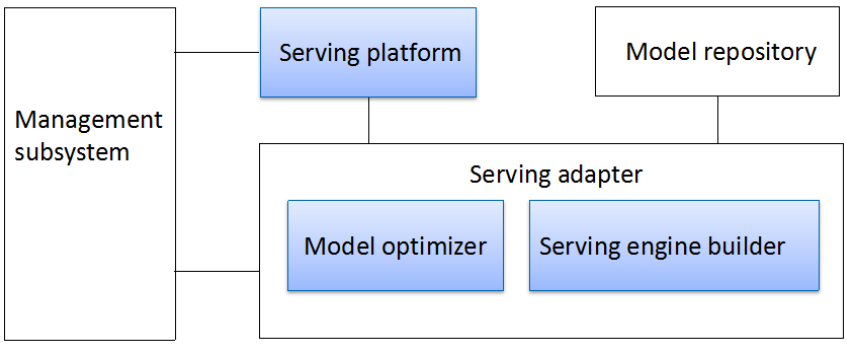

Figure 1 – Functional components of the serving framework

8.1.1 Model repository

This functional entity is a repository from which the serving framework gets ML models from.

NOTE – This figure treats model repository as a generic component inside the model marketplace.

8.1.2 Serving adapter

Serving adapter, composed of model optimizer and serving engine builder, is used for adapting the model to serve on a serving platform with specific hardware environment.

8.1.2.1 Model optimizer

This functional entity is responsible for inference optimization of a ML model that is ready to serve the ML underlay network from a model repository, based on the topology of the model and the deployment environment to achieve the optimization goals. It will export an optimized model, the format of which may differ from the original model.

8.1.2.2 Serving engine builder

This functional entity generates the serving engine for ML models, which will run on the SErving Platform with the following functionality:

1) Runtime provision

   The serving engine loads the ML model artifacts with a certain runtime so that the ML model can run on the target hardware and perform inference according to the model configurations.

2) Model scheduling

   The serving engine schedules the models based on scheduling policies so that multiple serving models can execute inference efficiently.

3) Version management

   The serving engine manages the versions of the models according to the model update policy.

4) Model inference service provision

   The serving engine exposes the model inference as well as monitoring capability as a service that can be consumed by ML pipeline, which may be an online or batch prediction service.

8.1.3 Serving platform

This functional entity refers to the platform where models are deployed. It allocates resources to run the serving models.

There are several options to deploy ML models to the serving platform.

1) Embedded deployment

   Embedded deployment means ML models are built and packaged inside an ML application, where the whole ML pipeline is managed within the application. In this approach, ML models along with their runtime libraries act as dependencies of the application and to the method of performing inference is implementation dependent. Such deployment reduces the latency to consume the model inference and is mostly used in resource constrained scenarios such as in IoT devices.

2) Monolithic MLaaS deployment

   In the monolithic MLaaS deployment approach, a ML model is wrapped in a serving engine image, which is then deployed to serving platform as a whole, to provide a model inference service that can be consumed by ML pipelines. Such deployment can be chosen for large, centralized serving platforms that serve ML pipelines at scale.

3) Model-standalone MLaaS deployment

   Model-standalone MLaaS deployment refers to the approach where the serving engine which is already hosted on serving platform provides model inference service by loading ML models from a file path on serving platform where the models are stored. Such deployment requires only the acquisition of ML models, which will share the same Serving Engine when performing inference.

Serving platforms where models are deployed with option 2 or option 3 provide general support for Serving engine, e.g., traffic routing, scaling and monitoring, serving model management.

8.2 High level architecture

The architecture for the serving of ML models in telecommunication networks aligns with the architecture for ML marketplace integration in network specified in [ITU-T Y.3176], which is shown in Figure 2. All the reference points align with their definitions in [ITU-T Y.3172] and [ITU-T Y.3176].

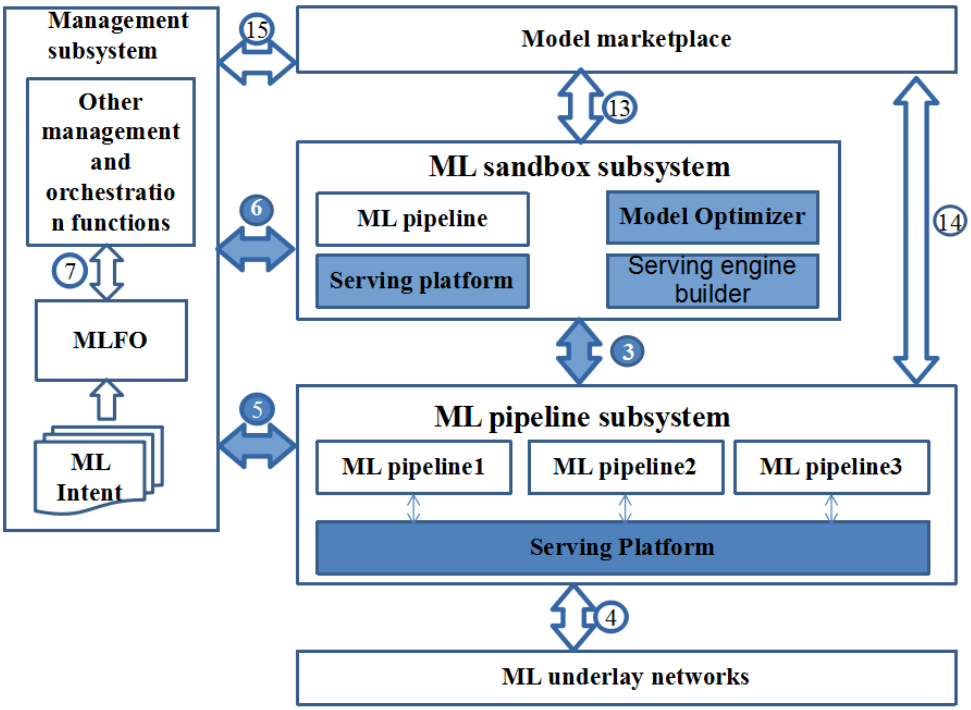

Figure 2 – Architecture for optimization and deployment of ML models in telecommunication network

NOTE – As defined in clause 3.1.2, the ML marketplace may include internal and external marketplaces.

As shown in Figure 2, model optimizer is added as a new enhanced function in the ML sandbox, which allows models obtained from the internal ML marketplace via reference point 13 to be optimized. The optimized models are deployed and evaluated on the Serving Platform in ML sandbox subsystem and then deployed as serving models on the serving platform in ML pipeline subsystem via reference point 3.

8.3 Reference points in serving framework of ML models in ML marketplace

Reference points 3, 5 and 6 which are used when serving ML models, are described in the following clauses.

8.3.1 Reference point 3 between ML sandbox subsystem and ML pipeline subsystem

This reference point is reused from [ITU-T Y.3172] and is used for deployment or update of models in the ML pipeline subsystem.

8.3.1.1 Serving request API

Application Programming Interface (API) description: When the serving platform receives the model deployment request from MLFO, it can pull ML models (for model-standalone deployment) or serving engine images (for monolithic) from ML sandbox (see Tables 1 and 2).

Direction: Serving platform → ML sandbox

Table 1 – Service request API (Serving platform → ML sandbox)

Information element	Type	Mandatory/Optional/ Conditional	Description
Deployment option	Enum	Mandatory	It can be monolithic/model standalone.
Model ID/Serving engine ID	String	Mandatory	ID of the serving engine.

Response: Serving response

Direction: ML sandbox → Serving platform

Table 2 – Service request API (ML sandbox → Serving platform)

Information element	Type	Mandatory/Optional/ Conditional	Description
Result	Enum	Mandatory	Indicates whether the serving request is successful.
Model ID/Serving engine ID	String	Optional	The ID of the target model/serving engine.

8.3.2 Reference point 5: Interface between management subsystem and ML pipeline subsystem

This reference point enables model management and monitoring functionality. After a model is pushed to the ML pipeline subsystem, MLFO triggers the monitoring operation in ML pipeline subsystem and receives the monitoring result reported from it. MLFO can also manage the registration and lifecycle of serving models.

8.3.2.1 Model deployment request API

API description: For a model that is evaluated in the sandbox, MLFO can trigger the deployment from sandbox to serving engine in ML pipeline subsystem (see Tables 3 and 4).

Direction: MLFO → Serving platform

Table 3 – Model deployment request API (MLFO → Serving platform)

Information element	Type	Mandatory/Optional/ Conditional	Description
Model ID/Serving engine ID	String	Mandatory	The ID of the target model /Serving engine.
Deployment configuration	Json	Mandatory	Configuration data according to the requirements of the use case, e.g., deployment option, memory and CPU requirements, scaling policy, update policy, etc.
Serving identifier	String	Mandatory	ID specified for the serving model.
Version identifier	String	Optional	Version ID specified for the serving model.

Response: Model deployment response

Direction: Serving platform → MLFO

Table 4 – Model deployment request API (Serving platform → MLFO)

Information element	Type	Mandatory/Optional/ Conditional	Description
Serving identifier	String	Mandatory	ID of the serving model
Result	Enum	Mandatory	Indicates whether the deployment was successful.

8.3.2.2 Model registration API

API description: After a model is deployed in the serving platform, its profile including the metadata, configuration, location and other basic information about the model is registered so that model consumers can discover it from the registration information. This registration can be done in MLFO via this API, i.e., MLFO will maintain the profile of the model. Furthermore, if the status of the deployed model is updated, its profile in MLFO is changed accordingly. Similarly, the serving engine can deregister a model from MLFO and MLFO will remove its profile (see Tables 5 and 6).

NOTE 1 – The update of the model profile in MLFO can be notified to the model consumer either by polling or a pub/sub pattern.

NOTE 2 – MLFO can deregister a serving model when it is no longer available, and then MLFO cannot receive the periodic heartbeat sent from the serving model.

Direction: ML pipeline subsystem → MLFO

Table 5 – Model registration API (ML pipeline subsystem → MLFO)

Information element	Type	Mandatory/Optional/ Conditional	Description
Serving version	String	Mandatory	Version of the serving model
Serving identifier	String	Mandatory	ID of the serving model
Model metadata	<Attribute, value> array	Mandatory	Contains the metadata of the model
API information	String	Mandatory	Information about how the inference service is provided so that consumers of the service can discover the model.
Model status	Enum	Mandatory	Indicates whether the service of the model is available

Response: Model registration response

Direction: MLFO → ML pipeline subsystem

Table 6 – Model registration API (MLFO → ML pipeline subsystem)

Information element	Type	Mandatory/Optional/ Conditional	Description
Result	Enum	Mandatory	Indicates whether the registration was successful
Serving identifier	String	Conditional	Indicates the id of the serving model

8.3.2.3 Serving model management API

API description: This API is exposed by the serving platform to manage the serving models, examples are start/stop/upgrade/scale/modify/delete operations, which may be used by the authorized management subsystem, e.g., MLFO (see Tables 7 and 8).

Direction: MLFO → Serving platform

Table 7 – Service model management API (MLFO → Serving platform)

Information element	Type	Mandatory/Optional/ Conditional	Description
Serving identifier	String	Mandatory	ID of the serving model
Operation	Enum	Mandatory	E.g., Start/stop/upgrade/scale/modify/delete operations
Additional information	Key-value pairs	Mandatory	Additional information or configuration for an operation, e.g., the modify operation may have memory setting information, model version information.
Authorization	String	Mandatory	This field is used for authentication and if the verification fails, the request will be rejected.

Response: Model management response

Direction: MLFO → Serving platform

Table 8 – Service model management API (Serving platform → MLFO)

Information element	Type	Mandatory/Optional/ Conditional	Description
Result	Enum	Mandatory	Indicates whether management operation was successful
Serving Identifier	String	Mandatory	ID of the serving model

8.3.2.4 Model monitoring subscription API

API description: After the ML model is deployed in the ML pipeline subsystem, the subscription can be exposed by this API, e.g., to MLFO (see Table 9).

Direction: MLFO → ML pipeline subsystem

Table 9 – Model monitoring subscription API (MLFO → ML pipeline subsystem)

Information element	Type	Mandatory/Optional/ Conditional	Description
Serving Identifier	String	Mandatory	ID of the serving model whose monitoring event is subscribed.
Notification Target Address	String	Mandatory	The address where the monitoring notification will be sent

Table 9 – Model monitoring subscription API (MLFO → ML pipeline subsystem)

Information element	Type	Mandatory/Optional/ Conditional	Description
Monitoring event information	String list	Mandatory	Contains monitoring information that MLFO is interested, e.g., model status, model performance (with metrics like Confusion matrix, AUC, MSE), exceptions, and triggering event e.g., periodicity or threshold.

8.3.2.5 Model monitoring event notification API

API description: When a monitoring event is triggered, a notification will be sent to the subscribers (see Table 10).

Direction: ML pipeline subsystem → MLFO

Table 10 – Model monitoring event notification API (ML pipeline subsystem → MLFO)

Information element	Type	Mandatory/Optional/ Conditional	Description
Monitoring event information	String	Mandatory	Results of some monitoring event

8.3.2.6 Health check request API

API description: This API is used to check whether the serving model on the serving engine is active and able to process inference requests (see Tables 11 and 12).

Direction: MLFO → Serving platform

Table 11 – Health check request API (MLFO → Serving platform)

Information element	Type	Mandatory/Optional/ Conditional	Description
Serving identifier	String	Mandatory	ID of the serving model

Response: Health check response

Direction: Serving platform → MLFO

Table 12 – Health check request API (Serving platform → MLFO)

Information element	Type	Mandatory/Optional/ Conditional	Description
Result	Enum	Mandatory	Indicates whether the serving service is active

8.3.3 Reference point 6: Interface between management subsystem and sandbox subsystem

This reference point is used by the management subsystem to manage the models pushed from the marketplace to the sandbox. For example, MLFO may trigger and monitor the training, optimization, chaining and deployment process of a ML model in sandbox.

8.3.3.1 Model optimization request API

API description: The newly trained model in the ML sandbox is triggered for optimization before it is deployed in the ML pipeline (see Tables 13 and 14).

Direction: MLFO → ML sandbox

Table 13 – Model optimization request API (MLFO → ML sandbox)

Information element	Type	Mandatory/Optional/ Conditional	Description
Model identifier	String	Mandatory	ML sandbox already knows the model id.
Optimization object	JSON	Mandatory	MLFO specifies the optimization information, e.g., target performance metrics, the backend the model will run on, what type of optimization, which parameters were used, and what framework was followed such as Adlik or TensorRT.

Response: Model optimization response

Direction: ML sandbox → MLFO

Table 14 – Model optimization request API (ML sandbox → MLFO)

Information element	Type	Mandatory/Optional/ Conditional	Description
Result	Enum	Mandatory	Indicates whether the optimization was successful.
Model identifier	String	Mandatory	Indicates the id of the model updated.
Optimization result parameters	JSON	Conditional	Some information from the result of optimization is passed here. E.g., performance metrics like accuracy, latency and throughout.

8.3.3.2 Model monitoring subscription API

API description: This API is similar to the model monitoring subscription API in clause 8.3.2.4. This API is used to monitor the model in the sandbox subsystem.

8.3.3.3 Model monitoring event notification API

API description: This API is similar to the model monitoring event notification API in clause 8.3.2.5. This API is used to send monitoring notification of the model in the sandbox subsystem.

8.3.3.4 Model deployment trigger

API description: For a model that is evaluated in the sandbox, MLFO can trigger model deployment in the ML sandbox for validation and evaluation (see Tables 15 and 16).

Direction: MLFO → Serving platform in the ML sandbox

Table 15 – Model deployment trigger (MLFO → Serving platform in the ML sandbox)

Information element	Type	Mandatory/Optional/ Conditional	Description
Model identifier	String	Mandatory	Model ID
Deployment configuration	Json	Mandatory	Additional configurations of the deployment
Inference configuration	Json	Mandatory	Configuration information for model inference, e.g., hardware information, performance requirements, model scheduling policy, batching policy, deployment option.

Response: Serving platform in the ML sandbox → MLFO

Table 16 – Model deployment trigger (Serving platform in the ML sandbox → MLFO)

Information element	Type	Mandatory/Optional/ Conditional	Description
Model identifier	String	Mandatory	Model ID
Serving engine ID	String	Mandatory	ID of the Serving engine image.
Serving Engine metadata	Json	Mandatory	The meta-data of the target serving engine indicating the nature of the Serving Engine.
Result	Enum	Mandatory	Indicates whether the deployment was successful.

8.3.4 Reference point sp

This reference point is used by the ML pipeline to utilize the inference ability supported by the serving model.

8.3.4.1 Inference request API

API description: This API is used to get inference service from a specific serving model (see Tables 17 and 18).

Direction: ML pipeline → Serving platform

Table 17 – Inference request API (ML pipeline → Serving platform)

Information element	Type	Mandatory/Optional/ Conditional	Description
Serving identifier	String	Mandatory	ID of the serving model
Metadata	Json	Mandatory	Meta-data of the inference request, indicating the nature of the input and expected output, e.g., name and data format of the input data.
Input data	bytes	Mandatory	The tensor data that needs to be passed to the serving model to get inference.

Response: Inference response

Direction: Serving platform → ML pipeline

Table 18 – Inference request API (Serving platform → ML pipeline)

Information element	Type	Mandatory/Optional/ Conditional	Description
Serving ID	String	Mandatory	ID of the serving model.
Result	Enum	Mandatory	Indicates whether the request was successful.
Metadata	Json	Mandatory	Meta-data of the response which can be used for the interpretation of the output data.
Output data	bytes	Mandatory	The output data of the inference request.

8.4 Sequence diagrams of serving of ML models

This clause describes the procedures for serving of ML models.

8.4.1 Model optimization in the ML sandbox

Figure 3 shows the model optimization in the ML sandbox.

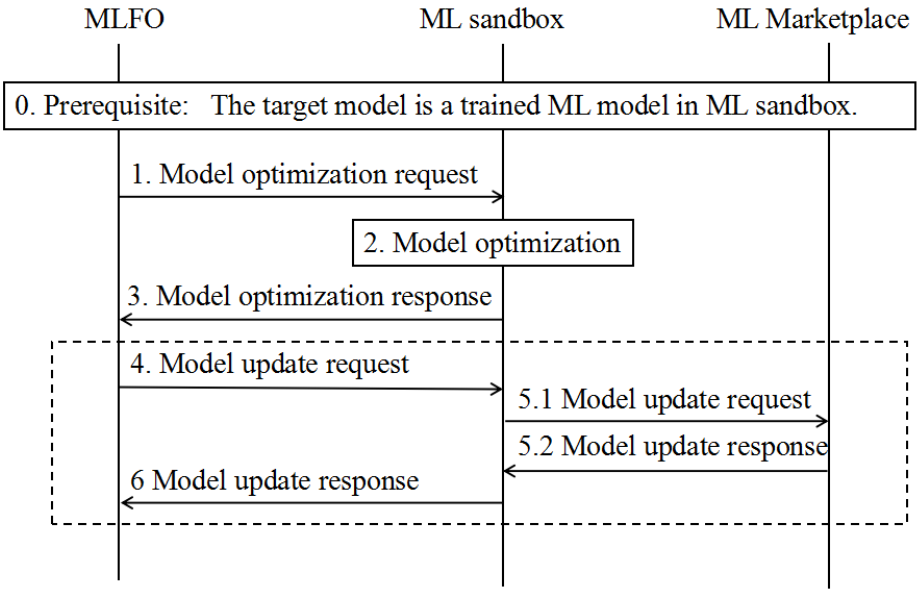

Figure 3 – Model optimization in the ML sandbox

Prerequisite: The model to be optimized is a trained model that is ready to serve the ML underlay network.

NOTE 1 – It can be either a model that was trained in the ML sandbox in the "model training" procedure or a well-trained model that was pushed from marketplace as indicated in the "Model selection and push" procedure defined in [ITU-T Y.3176].

1. MLFO sends a model optimization request to model optimizer in the ML sandbox to indicate which model needs to be optimized and to provide some other optimization information according to the requirements implied in the ML intent.

2. Model optimizer determines the optimization solutions for the target model, tunes the optimization parameters and exports the compiled output model, which is then evaluated in the ML sandbox to verify if the optimized performance meets the optimization goal.

3. The ML sandbox sends a model optimization response to MLFO to indicate the result of the optimization.

4. Steps 4-6 occur only when MLFO decides to update the optimized model as well as the optimization information to the marketplace based on the network operators' ML model update policy and the result of the optimization. MLFO triggers the model update procedure as indicated in clause 8.4.4 in [ITU-T Y.3176].

NOTE 2 – The difference between model update in model optimization and that in model training lies in the content of the model object. e.g., for model optimization, it may be an optimized model with sparse connections and compressed structure.

8.4.2 Model deployment in ML sandbox for evaluation

Figure 4 shows the model deployment in the ML sandbox for evaluation.

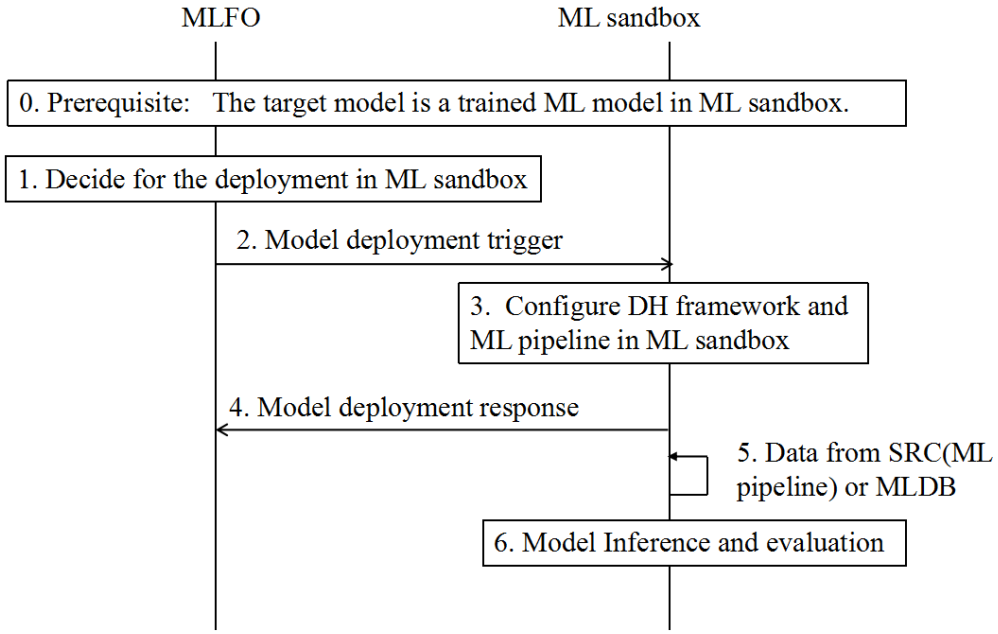

Figure 4 – Model deployment in the ML sandbox for evaluation

Prerequisites: The target model to be deployed is a trained ML model in the ML sandbox.

NOTE – The model may be an optimized model.

1. According to the ML intent, MLFO decides on model deployment in the ML sandbox.

2. MLFO sends a model deployment trigger message to the ML sandbox which contains the model ID and the model inference configuration information.

3. The serving engine builder in the sandbox prepares a serving engine image for validation according to the inference configuration. The ML pipeline for serving, the data handling framework (see clause 8.3.1 in [ITU-T Y.3174]) and simulator (ML5G 171) are configured in ML sandbox.

4. The ML sandbox sends a response of model deployment to MLFO to indicate the result of the deployment, including the serving engine ID if it is a monolithic deployment.

5. Data from the underlay or from the machine learning database (MLDB) is used as input for model inference and evaluation.

8.4.3 Model deployment from ML sandbox to Serving Platform in ML pipeline subsystem

Figure 5 shows the model deployment from the ML sandbox to the serving platform.

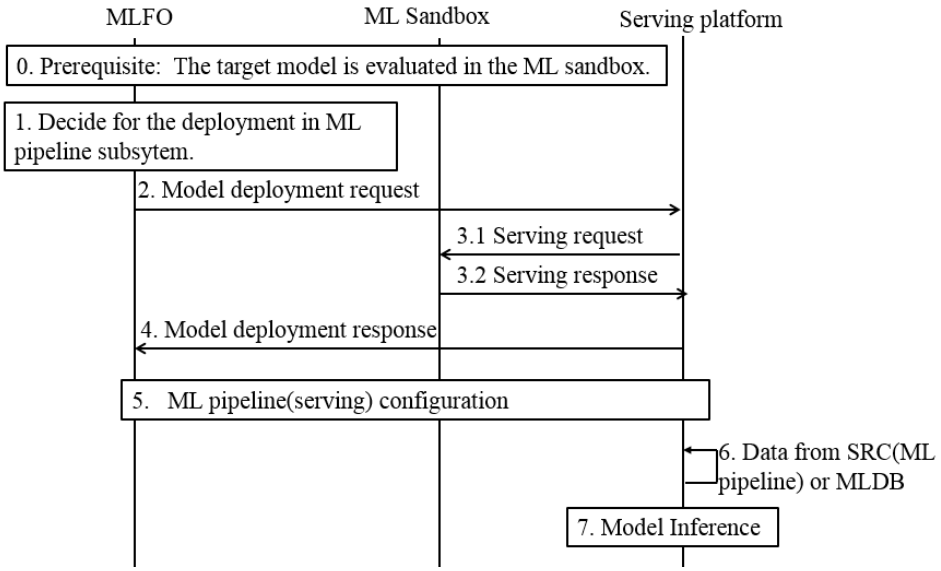

Figure 5 – Model deployment from the ML sandbox to the serving platform

Prerequisites: The target model is evaluated in the ML sandbox.

1. Based on the performance of the ML model in the ML sandbox, MLFO decides on deployment in the ML pipeline subsystem.

2. MLFO sends a model deployment request message to the serving platform, including the serving ID, version ID, deployment configuration, serving engine ID (for monolithic deployment) and model ID (for model-standalone deployment).

3. The serving platform sends a serving request to the serving engine builder with the information of the serving engine ID (for monolithic deployment) or model ID (for model-standalone deployment).

4. The serving platform creates the service instance for model inference by running the serving engine and sends an asynchronous model deployment response to MLFO to indicate whether the deployment is successful.

5. ML pipeline (serving) is configured and the detailed procedure is shown in clause 8.2.4.

6. Data from the underlay or MLDB is used as input for model inference.

7. Model inference starts in the ML pipeline subsystem.

8.4.4 ML pipeline (serving) configuration in ML pipeline subsystem

Figure 6 shows the ML pipeline (serving) configuration in the ML pipeline subsystem with the serving engine.

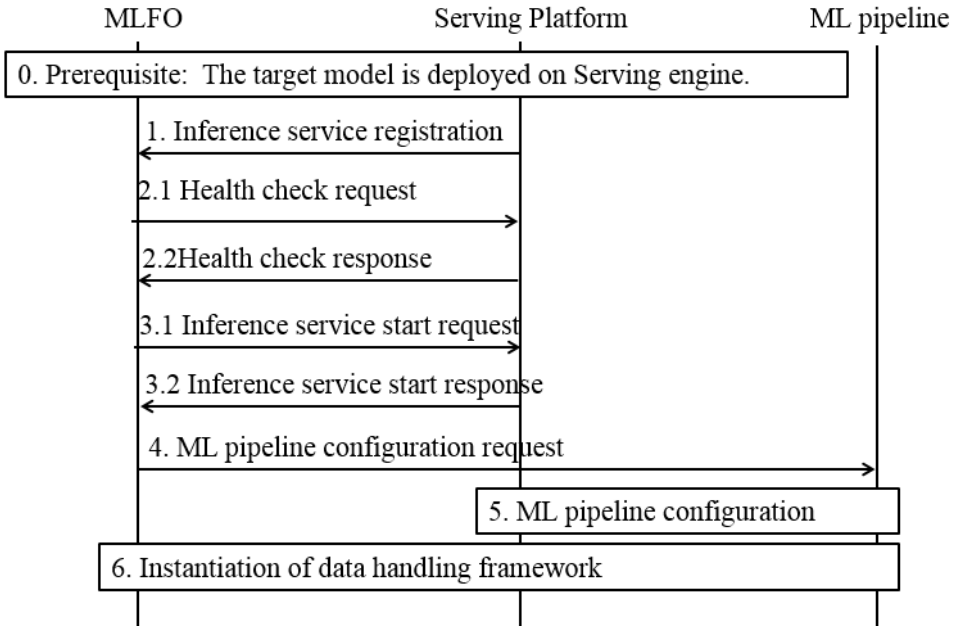

Figure 6 – ML pipeline (serving) configuration in the ML pipeline subsystem with the serving engine

Prerequisite: The target model is deployed on the serving platform.

1. The ML pipeline subsystem registers the new inference service to MLFO.

2. When MLFO needs to create a new ML pipeline with this inference service, it sends a health check request to check whether the inference service is running on the serving engine.

NOTE 1 – Based on the network operator's policy, MLFO can decide whether to stop an inference service when there's no ML pipeline configured to consume this service.

NOTE 2 – The health check may be done periodically as per the configuration in MLFO.

3. If the service is not running, it sends an inference service start request to the serving engine to get the inference service ready for getting data input and perform inference.

4. MLFO sends a ML pipeline configuration to the ML pipeline subsystem.

5. The inference service is configured as the model node in the pipeline, along with other nodes.

6. The data handling framework is instantiated as specified in [b-Data handling] and the ML pipeline starts to send the collected data to the inference service and perform further operations on the inference output.

8.4.5 Model monitoring

Figure 7 shows the model monitoring.

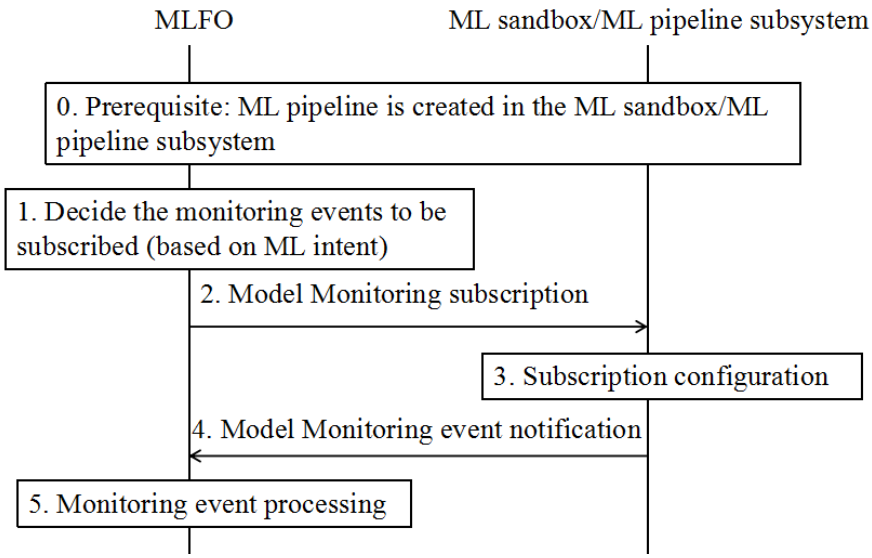

Figure 7 – Model monitoring

Prerequisites: ML pipeline (training/serving) is created in the ML sandbox/ML pipeline subsystem.

1. Based on the ML intent, MLFO decides the monitoring events to be subscribed to.

2. MLFO subscribes to the monitoring events of models in the ML sandbox/ ML pipeline subsystem.

3. Based on the event object, MLFO is configured as a subscriber of the monitoring event of the target model in the ML sandbox/ ML pipeline subsystem.

4. The ML sandbox/ ML pipeline subsystem responds to MLFO with monitoring event notification. This may be a periodic event.

5. The MLFO processes the monitoring notification, e.g., MLFO may trigger Model update, scaling of serving instances, etc.

8.4.6 Model update

Figure 8 shows the model update.

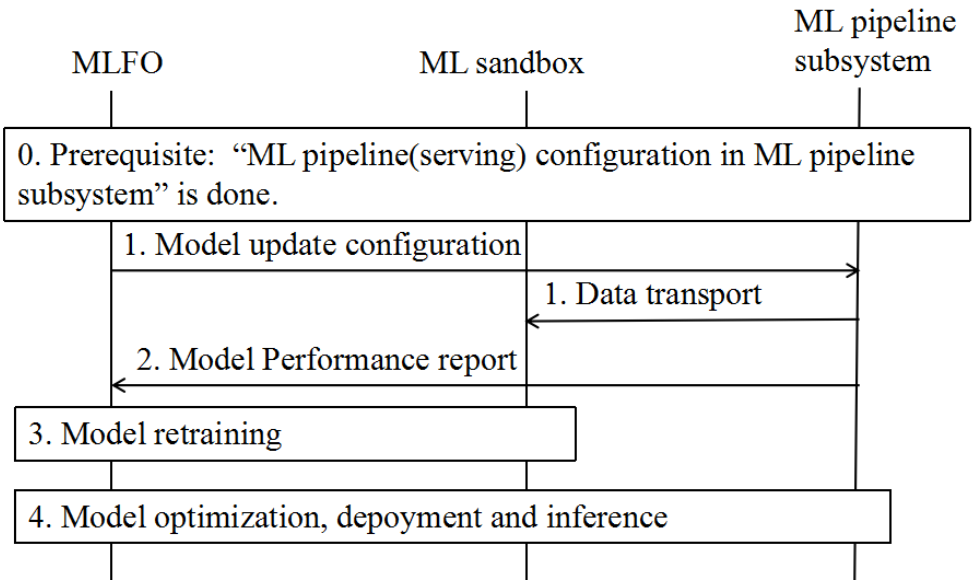

Figure 8 – Model update

Prerequisites: "ML pipeline (serving) configuration in ML pipeline subsystem" is done.

1. The inference data during the inference stage is continuously sent to the ML sandbox as new training data according to the model update configuration received from MLFO.

2. Model performance status is reported to MLFO via the model monitoring mechanism.

3. If MLFO decides to update the model in the ML pipeline subsystem according to the model update strategy and model performance status, the model is retrained in the ML sandbox.

4. The following steps are model optimization, deployment and inference based on the instruction of MLFO.

Bibliography

[b-ITU-T Sup.55] ITU-T Supplement 55 to Y.3170-series (2019), *Machine learning in future networks including IMT-2020: use cases*.

[b-Kubernetes] Concepts of Kubernetes https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[b-LF Acumos] Acumos AI Marketplace https://www.acumos.org

[b-Online learning] http://www.mit.edu/~rakhlin/6.883/