ITUPublications

International Telecommunication Union

Standardization Sector

# ITU-T Technical Specification

(07/2020)

# ML5G-DEL3

# Machine learning sandbox for future networks including IMT-2020: requirements and architecture framework

## Technical Specification

# Machine learning sandbox for future networks including IMT-2020: requirements and architecture framework


### Summary

Use cases for integrating machine learning (ML) into future networks including IMT-2020 have been documented in Supplement 55 and an architecture framework for this integration was specified in Recommendation ITU-T Y.3172. However, network stakeholders are apprehensive about using ML-driven approaches directly in live networking systems because they can lead to unexpected situations that can degrade KPIs. This is mostly due to the apparent complexity of ML mechanisms (e.g., deep learning), the incompleteness of the available training data and the uncertainty produced by exploration-exploitation approaches (e.g., reinforcement learning), etc. In the face of such impediments, the ML Sandbox emerges as a potential solution that allows mobile network operators (MNOs) to improve the degree of confidence in ML solutions before their application to the network infrastructure. This Technical Specification deals with the requirements, architecture, and implementation examples for an ML sandbox in future networks including IMT-2020.



### Keywords

Architecture, requirements, sandbox, simulator.


### 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.


**Contact:** Slawomir Stanczak
Fraunhofer HHI Germany
Email: slawomir.stanczak@hhi.fraunhofer.de

## Table of Contents

# Technical Specification

## Machine learning sandbox for future networks including IMT-2020: requirements and architecture framework

## 1 Scope

This Technical Specification defines the architecture of the ML Sandbox to improve the level of confidence in future networks including IMT-2020. The architectural structure of the ML sandbox is derived from a list of requirements. In addition, this it includes several illustrative examples concerning the integration of the ML sandbox into use cases. Finally, practical implementation aspects are discussed for the fast adoption of the ML Sandbox specifications.

## 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.3173] Recommendation ITU-T Y.3173 (2020), *Framework for evaluating intelligence levels of 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. Sup55] Supplement ITU-T Y.3170-series (2019*), Machine learning in future networks including IMT-2020: Use cases*.

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

[ML5G-I-216] ITU-T FG ML5G specification ML5G-I-216, *Requirements, architecture and design for machine learning function orchestrator*, https://www.itu.int/en/ITU-T/focusgroups/ml5g/pages/default.aspx.

[ML5G-I-227] ITU-T FG ML5G specification ML5G-I-227, *Serving framework for ML models in future networks including IMT-2020*, https://www.itu.int/en/ITU-T/focusgroups/ml5g/pages/default.aspx.

## 3 Definitions

### 3.1 Terms defined elsewhere

This Technical Specification uses the following terms defined elsewhere:

**3.1.1 machine learning model** [ITU-T Y.3172]: Model created by applying machine learning techniques to data to learn from.

**3.1.2 machine learning pipeline** [ITU-T Y.3172]: A set of logical nodes, each with specific functionalities, that can be combined to form a machine learning application in a telecommunication network.

**3.1.3 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.

**3.1.4 machine learning function orchestrator (MLFO)** [ITU-T Y.3172]: A logical node with functionalities that manage and orchestrate the nodes in a machine learning pipeline.

**3.1.5 machine learning marketplace** [ITU-T Y.3176]: A repository of machine learning models.

**3.1.6 machine learning model metadata** [ITU-T Y.3176]: Data related to the characteristics of a machine learning model.

NOTE – Examples of machine learning model metadata include the author, license, description of the machine learning model, the name of the machine learning model, version of the machine learning model, description of the data inputs and outputs of the machine learning model and runtime environment of the ML model.

**3.1.7 network intelligence level** [ITU-T Y.3173]: Level of application of automation capabilities including those enabled by the integration of artificial intelligence techniques in the network.

**3.1.8 model serving** [ML5G-I-227]: The process by which an ML model becomes integrated with other ML pipeline nodes and is ready for being executed in the ML underlay.

## 3.2 Terms defined in this Technical Specification

This Technical Specification defines the following terms:

**3.2.1 capability exposure**: External exposure of capabilities of network functions (NFs), supported by the network exposure function (NEF), which can be categorized as monitoring capabilities, provisioning capabilities and policy/charging capabilities.

**3.2.2 evaluation block**: Chaining of pipeline nodes and simulated network functions (NFs) with served machine learning (ML) models whose goal is to evaluate a particular ML use case.

**3.2.3 machine learning output**: Policies or configurations to be applied in the network.

NOTE – Definition based on [ITU-T Y.3172].

**3.2.4 overfitting**: A phenomenon where a machine learning model fits well with the training dataset but lacks generalization in addressing unseen real-world situations.

NOTE – Overfitting mostly occurs when the machine learning model learns details from the training dataset, including the undesired phenomena that can potentially affect the accuracy of the model's output (e.g., noise, malicious perturbations, etc.).

**3.2.5 simulation component metadata**: Data describing the characteristics of a particular simulation component.

NOTE – Examples of simulation component metadata include the capabilities of simulated network functions (NFs), configurable parameters, performance indicators and monitored parameters, interfaces.

**3.2.6 simulation environment metadata**: Data describing the characteristics of a particular simulation environment.

NOTE 1 – Simulation environment metadata can contain information such as installation/execution requirements, simulation component metadata, performance indicators, connections, maturity indicators (alpha/beta versions), etc.

NOTE 2 – The format for representing simulation environment metadata can be JSON, CSV, or XML, to name a few formats.

**3.2.7 simulation profile**: A list of parameters and their values which describe the machine learning (ML) use case to be trained, evaluated, or tested at the ML sandbox.

NOTE – The list of parameters and their values may be derived from the machine learning (ML) profile and the simulation environment metadata.

# 4 Abbreviations and acronyms

This Technical Specification uses the following abbreviations and acronyms:

AI Artificial Intelligence

API Application Programming Interface

KPI Key Performance Indicator

| | |
|---|---|
| ML | Machine Learning |
| MLFO | Machine Learning Function Orchestrator |
| NF | Network Function |
| OAM | Operation Administration and Maintenance |

## 5 Conventions

The integration of AI/ML techniques has been identified as one of the key features of future networks. However, network operators have the unique challenge of maintaining the key performance indicators (KPIs) even during or after this integration. In addition, the introduction of machine learning (ML) techniques to fifth generation (5G) networks may raise concerns regarding transparency, reliability, and availability. Often, ML methods are seen as black boxes (especially for deep learning, where the internal operation of the model is unknown because it is too complex or even hidden) that can learn complex patterns from training datasets. However, such datasets may be limited and/or too complex, thus questioning the accuracy of the output of the ML mechanism. In particular, reducing the generalization error is the main concern when applying any kind of supervised learning (SL) approach, which can be high even if the test error is kept low (this phenomenon is commonly known as *overfitting*). Apart from SL methods, other branches of ML such as unsupervised learning (UL) and reinforcement learning (RL) deal with uncertainty in one way or another. Such uncertainty may entail the application of changes in the network leading to unacceptable performance. On the one hand, unsupervised learning aims to find patterns from data without any guidance (unlabelled data) and hence lacks validation. On the other hand, RL is based on the learning-by-experience paradigm. RL has been shown to be of great utility for single-agent approaches in controlled scenarios, however notable adverse effects can appear as a result of the competition raised by multiple systems sharing the same resources (e.g., while providing heterogeneous services using common network resources). Moreover, when multiple systems are competing for the same market of users, exploration may hurt a system's reputation in the near term, with adverse competitive effects.

All kinds of learning can, therefore, lead to unexpected and/or undesired behaviour. Even if the performance of networking systems can be improved by ML techniques in the long-term, it is safe to assume that the system will unavoidably experience certain performance degradation during a transitory regime.[1] In some situations, this degradation of KPIs may be unacceptable for network operators, especially for demanding requirements of certain network-oriented applications such as ultra-reliable low-latency communication (uRLLC). In other cases, the network may change quickly and may not reach a stable, long-term regime that is intended to optimize the network's performance.

Given the instability that ML methods can generate in networking systems (especially those based on exploration-exploitation), the **sandbox subsystem** [ITU-T Y.3172] emerges as a promising solution for training, testing and evaluating the performance of ML models before being deployed in live environments. The ML sandbox is an isolated environment in which machine learning models can be evaluated. The ML sandbox is therefore meant to reproduce the behaviour/operation of live networking systems, thus improving the robustness and resilience of future ML-enabled networking systems. The ML sandbox includes a managed test network (e.g., a testbed) or a software-based environment (e.g., using a simulator/emulator), including simulators, emulators, and models. In this regard, network simulators can be particularly useful for overcoming the limitations of limited training datasets and laboratory-based testbeds. Simulators can be used to frame cases that have not been noticed before (i.e., anomalies), which would contribute to enabling failure prediction, anomaly detection, and self-healing.

---

[1] The transitory regime precedes the stability phase of an ML model when applied to a network. Performance degradation can result from potential delays in serving models in the network, or from trying suboptimal configurations during exploration periods in online learning.

Through the management subsystem, network operators can manage the ML sandbox and thereby address the challenges posed by ML-driven solutions for networks. The interfaces between the MLFO and the ML Sandbox allow the manageability of the simulation, the execution of test cases, and the evaluation of ML models.

## 6 Requirements

High-level requirements for the ML sandbox are divided into the following broad classifications:

– Simulated ML underlay requirements;

– ML operations requirements;

– Communication requirements;

– Metadata requirements.

### 6.1 Simulated ML underlay requirements

**REQ-ML-SANDBOX-001**: ML sandbox is required to simulate heterogeneous sources of data and SINKs of ML output.

NOTE – SRCs and SINKs simulated in the ML sandbox include those within the IMT-2020 network as well as application functionalities hosted in network slices. Examples of application functionalities hosted in network slices are V2X applications, Industry 4.0 applications, emergency applications.

**REQ-ML-SANDBOX-002**: ML sandbox is required to support the dynamic instantiation of new simulated SRCs and/or SINKs.

NOTE – The instantiation of new simulated SRCs and SINKs is managed by MLFO.

**REQ-ML-SANDBOX-003**: ML sandbox is required to consider policy inputs from the operator to configure the simulated ML underlays.

NOTE – E.g., conflict resolution, resource management.

**REQ-ML-SANDBOX-004**: ML sandbox is required to provide coordinated time sync, stop & start, and restart functions to control the simulated ML underlays.

NOTE 1 – See [ITU-T Y.3172] for the MLFO interface.

NOTE 2 – Synchronization of all simulated NFs can be done before reporting to MLFO.

**REQ-ML-SANDBOX-005**: ML sandbox is recommended to include the quality of data of the simulated data to be used by ML models (training or testing).

**REQ-ML-SANDBOX-006**: ML sandbox is recommended to support demand mapping for configuring and updating the simulated ML underlays.

NOTE 1 – Demand mapping is achieved by looking at the configurations and requirements from the operator (ML profile) and mapping them to specific operation administration and maintenance (OAM) steps for implementing the use case.

NOTE 2 – Demand mapping can be complemented for new use cases through the analysis of data patterns and ML pipeline output.

**REQ-ML-SANDBOX-007**: ML sandbox is required to provide sanity checks to assess the correct operation of the simulated ML underlays.

### 6.2 ML operations requirements

**REQ-ML-SANDBOX-008**: ML sandbox is required to support multiple ML pipelines, which may be chained and which potentially are interfaced with simulators from different levels of the network.

NOTE – Network levels are defined in [ITU-T Y.3172].

**REQ-ML-SANDBOX-009**: ML sandbox is required to support the monitoring and evaluation of ML pipelines and simulation components according to specifications in the ML intent.

NOTE – This may include threshold-based asynchronous notifications from Sandbox (to MLFO), post-processing of ML output, metering, security threat analysis, etc.

**REQ-ML-SANDBOX-010**: ML sandbox is required to support testing and evaluation of multiple ML pipelines at the same time, including aggregated impacts on the network due to them.

NOTE 1 – For instance, two different models (e.g., based on RL and SL) are triggered simultaneously for addressing an ML use case. The output of both methods is compared to make a decision. Also, "fork" & "join" functionalities.

NOTE 2 – To allow parallelization, fork-join methods can be included.

**REQ-ML-SANDBOX-011**: ML sandbox is required to support training and testing ML models that combine simulated and real data from the network.

NOTE 1 – The choice of data to be used is managed by MLFO [ML5G-I-216].

NOTE 2 – The combination of simulated and real data may also include augmented data.

**REQ-ML-SANDBOX-012**: ML sandbox is required to support dynamic resource allocation for ML pipeline nodes.

NOTE – E.g., to support various request handling mechanisms like load balancing.

**REQ-ML-SANDBOX-013**: ML sandbox is required to handle granular evaluation of ML test cases.

NOTE – Batch jobs can be triggered by the ML sandbox.

**REQ-ML-SANDBOX-014**: ML sandbox is required to support monitoring and evaluating the network intelligence level.

NOTE – See [ITU-T Y.3173] for monitoring and evaluating the network intelligence level.

**REQ-ML-SANDBOX-015**: ML sandbox is required to support testing techniques to enhance the robustness of the system.

NOTE – Generative adversarial networks (GANs) can be employed to test the validity of the input data to be used for training an ML model.

**REQ-ML-SANDBOX-016**: ML sandbox is required to produce a machine-readable format for the output of simulations, tests, and evaluations.

### 6.3 Communication requirements

**REQ-ML-SANDBOX-017**: ML Sandbox is required to support data handling reference points towards technology-specific simulated ML underlays.

NOTE – Data handling reference points are defined in [ITU-T Y.3174].

**REQ-ML-SANDBOX-018**: ML Sandbox is required to support the transfer of trained models across the different ML pipelines in the Sandbox as well as to other ML overlays.

**REQ-ML-SANDBOX-019**: ML Sandbox is required to support the transfer of data for training or testing models across different ML pipelines in the Sandbox as well as to other ML overlays.

**REQ-ML-SANDBOX-020**: ML Sandbox is required to support interfaces with ML marketplaces to transfer ML models.

NOTE – See reference point 13 in [ITU-T Y.3176] for the interface between ML marketplaces and the ML sandbox. This interface serves both the downlink (e.g., download models) and the uplink (e.g., update models).

**REQ-ML-SANDBOX-021**: ML sandbox is required to support data handling mechanisms, including metadata storage, interfacing with DMs and ML underlay networks, and data storage.

NOTE – See [ITU-T Y.3174] for data handling mechanisms.

## 6.4 Metadata requirements

**REQ-ML-SANDBOX-022**: ML sandbox is recommended to reuse the ML metadata store for different ML underlays.

NOTE 1 – The application programming interface API-g is stored in the management subsystem to allow the training, testing, and evaluation of ML models in the simulated ML pipeline [ITU-T Y.3174].

NOTE 2 – DM and corresponding API-s used in the simulated ML underlay network are stored in the management subsystem to allow the interworking between the data broker (DBr) and the simulated NFs [ITU-T Y.3174].

**REQ-ML-SANDBOX-023**: ML sandbox is recommended to use ML profile events from MLFO to configure and update the simulated ML underlay networks.

NOTE – ML profile events can be provided offline, at runtime, or based on triggers.

**REQ-ML-SANDBOX-024**: ML sandbox is recommended to use ML model metadata from the Marketplace to adjust the simulated ML underlays and the evaluation scenarios.

NOTE – For instance, the limitations of algorithms in terms of the amount of data (e.g., unsupervised learning) should be input as the amount of data to be generated (e.g., the number of APs to be simulated, total simulation time, minimum number of events, etc.).

**REQ-ML-SANDBOX-025**: ML sandbox is required to support simulation environment metadata.

NOTE 1 – Simulation environment metadata can be provided to the serving framework for considering the deployment environment while creating a serving engine (REQ-ML-DEP-002 in [ML5G-I-227]).

NOTE 2 – Simulation environment metadata includes the data models used by simulated NFs and APIs to access these data.

NOTE 3 – Simulation environment metadata can be used by DH to select the type of storage for data (REQ-ML-DH-011 in [ITU-T Y.3174]).

**REQ-ML-SANDBOX-024**: ML sandbox is recommended to support isolation between different instances of evaluation ML pipelines (instantiated for different ML underlays), e.g., for security reasons, support slicing, etc.

# 7 High-level architecture of the ML sandbox

To simulate ML underlay networks, the ML Sandbox includes simulated NFs, AFs, and ML pipeline(s) whose elements are managed by the machine learning function orchestrator (MLFO) [ML5G-I-216]. The ML sandbox is particularly useful for addressing dynamic networking systems since it allows the validation the effect of ML-based optimizations before they are deployed in production environments. In addition, because of the potential limitations of data coming from live networks (insufficient amount, privacy issue, etc.), the ML sandbox can be used to generate synthetic data as a complement for a given training procedure. Figure 1 illustrates the main functional components and reference points of the ML sandbox.

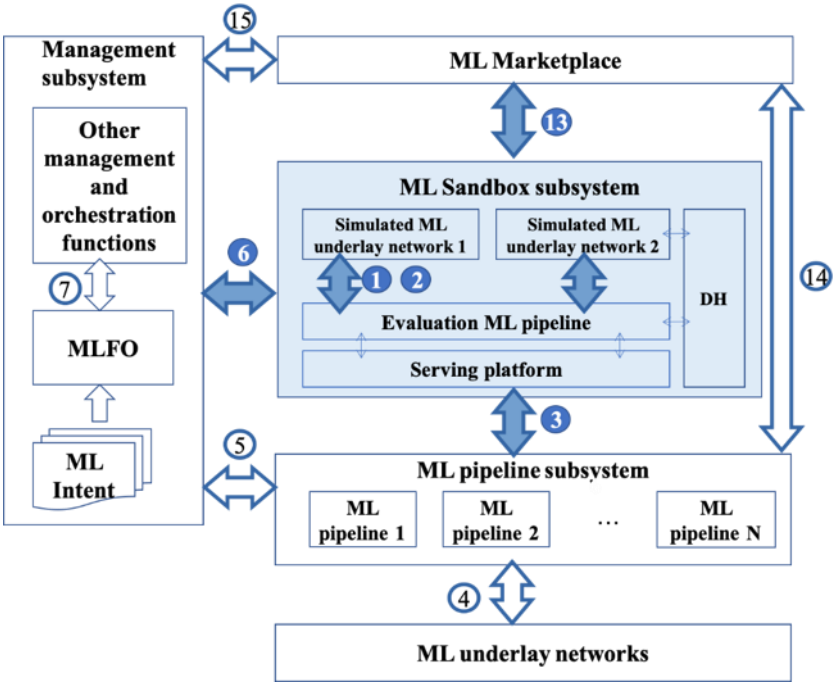


**Figure 1 – Overview of the main functional components and reference points of the ML Sandbox subsystem within the logical ML-aware architecture**

Data handling reference points **1** and **2** between the simulated ML underlay networks and the Evaluation ML pipeline (SINK and SRC) are used unmodified as defined in [ITU-T Y.3172] for training and update of ML models at the ML sandbox subsystem.

Reference point **3** is the interface between ML sandbox and ML pipeline subsystems, used unmodified as defined in [ITU-T Y.3172]. Interface 3.1 within reference point 3 allows the ML pipelines to interface with the ML sandbox subsystem for training and update of ML models. Data from the ML underlay networks and/or the simulated ML underlay networks may be used to train the ML models in the ML sandbox subsystem.

Reference point **4**, as defined in [ITU-T Y.3174], is the interface between the ML pipeline and the ML underlay network.

Reference point **5**, as defined in [ITU-T Y.3172], is the interface between the management subsystem and the ML pipeline subsystem.

Reference point **6** is used for the management subsystem to manage the models applied to the ML Sandbox [ITU-T ML5G-216], including monitoring and evaluating network intelligence levels [ITU-T Y.3173]. Reference point **6** has two parts:

– Reference point 6.1 [ITU-T Y.3174] is the interface between the management subsystem and the simulated ML underlay network. The management of the ML pipeline nodes in the ML Sandbox subsystem is controlled by the MLFO.

– Reference point 6.2 interfaces the management subsystem with the evaluation ML pipeline. Data from the ML underlay networks and/or the simulated ML underlay networks may be used to train ML models in the ML Sandbox subsystem.

Reference point **7** is the interface between MLFO and other management and orchestration functions of the management subsystem, used unmodified as defined in [ITU-T Y.3172].

Reference point **13** is the interface between the ML marketplace and the ML Sandbox subsystem, used unmodified as defined in [ITU-T Y.3176].

Reference point **14** is the interface between the ML marketplace and the ML pipeline subsystem, used unmodified as defined in [ITU-T Y.3176].

Reference point **15** is the interface between the management subsystem and the ML marketplace, used unmodified as defined in [ITU-T Y.3176].

## 7.1 Components of the ML sandbox

The ML sandbox contains the components defined in the following sub-clauses. The high-level architecture of the ML Sandbox subsystem is illustrated in Figure 2.

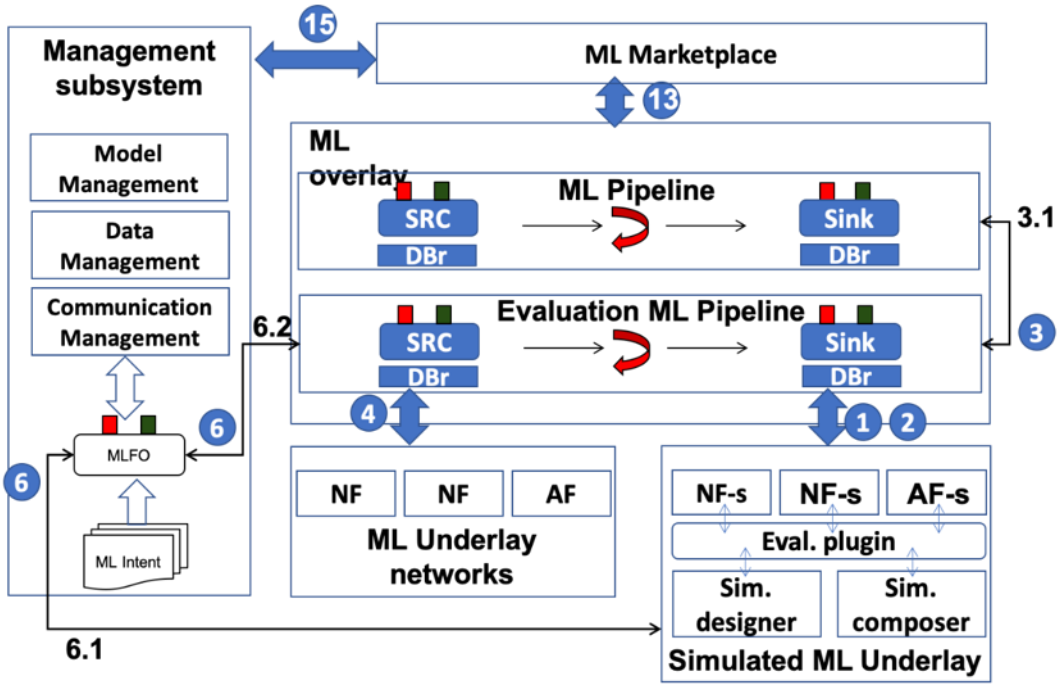


**Figure 2 – High-level architecture of the ML Sandbox components within the ML architecture**

### 7.1.1 Simulated ML underlay

Component defined in [ITU-T Y.3172], but modified for simulated ML underlay networks. It is used for testing, training, and evaluating ML models through simulated NFs.

NOTE – Simulated NFs can be third-party simulation tools such as ns-3.

### 7.1.2 Evaluation ML pipeline

Used for model evaluation in the ML sandbox environment, used unmodified as defined in [ITU-T Y.3172].

NOTE – The ML pipeline provides the configuration to be used in every simulation run (SRC), which at the same time provides feedback to be used by the ML model (SINK).

### 7.1.3 Data handling components

Set of components that serve for the storage of DMs and data, and interfacing with DMs and ML underlay networks, used unmodified as defined in [ITU-T Y.3174].

### 7.1.4 Monitoring agent

It monitors and evaluates the network intelligence level at the ML Sandbox. The following five dimensions are considered [ITU-T Y.3173]: demand mapping, data collection, analysis, decision, and action implementation.

Text from [ITU-T Y.3173]: The data collection process is done by source (SRC) nodes, the analysis and decision processes are performed by ML models and policy nodes, and the action implementation process is supported by SINK nodes. In addition, the demand mapping process is done using the ML intent provided as input to the MLFO.

### 7.1.5 Serving engine builder

This functional entity generates a serving engine for an ML model, used unmodified as defined in [ML5G-I-227].

### 7.1.6 Simulation designer

It prepares the set of simulation resources that compose the simulated ML underlay, and which are sought to be part of the evaluation block corresponding to the ML use case.

NOTE – The information to mimic/reproduce (setup/update) live ML underlays can come from the use case (ML intent) or data gathered by the live ML underlay (managed by MLFO). For example, as a result of network dynamics, some path-loss parameters used by the ML Sandbox subsystem may vary over time. To address this issue, the MLFO keeps track of those changes and provides feedback to the Simulation designer to update the necessary simulation parameters.

### 7.1.7 Simulation composer

It deploys, installs and runs the simulated ML underlay (simulators/emulators/models) to be used for characterizing different types of network underlays.

NOTE 1 – Based on the specified role and requirements of the ML sandbox (derived from use cases), the MLFO selects the best simulation tool (e.g., OMNET++ for PHY representation in LTE) and performs the deployment through the Simulation driver.

NOTE 2 – The deployment can be enabled by tools such as Docker and Kubernetes.

### 7.1.8 Evaluation plugin

It is responsible for interacting with the simulated ML underlay. For that purpose, it must provide technology-agnostic interfaces, which would enable interacting with third-party applications such as network simulators. In the simulation domain, for instance, the handler invokes operations such as *run*, *stop*, *pause*, etc. In addition, it manages the log files as provided by the simulator.

NOTE – For instance, when model evaluation is triggered, the handler runs the simulation by directly interacting with the simulator (e.g., ns-3, Komondor). Simulator-specific commands will be used for that purpose.

### 7.1.9 Simulation post-processor

It provides an interface whereby data from simulated ML underlays are post-processed and presented in a standard-compliant manner to the MLFO, which performs model evaluations and/or (re)training. This step is critical to handle heterogeneous sources of information.

NOTE – for instance, once the handler gets the raw logs generated by a simulator (e.g., a CSV file), the post-processor extracts the information to be used by the MLFO.

## 7.2 Sequence diagrams

### 7.2.1 Capability discovery/negotiation for third-party simulation components

Discover third-party simulation components that can be used to perform use-case-specific simulations in the ML sandbox. The sequence diagram is shown in Figure 3.

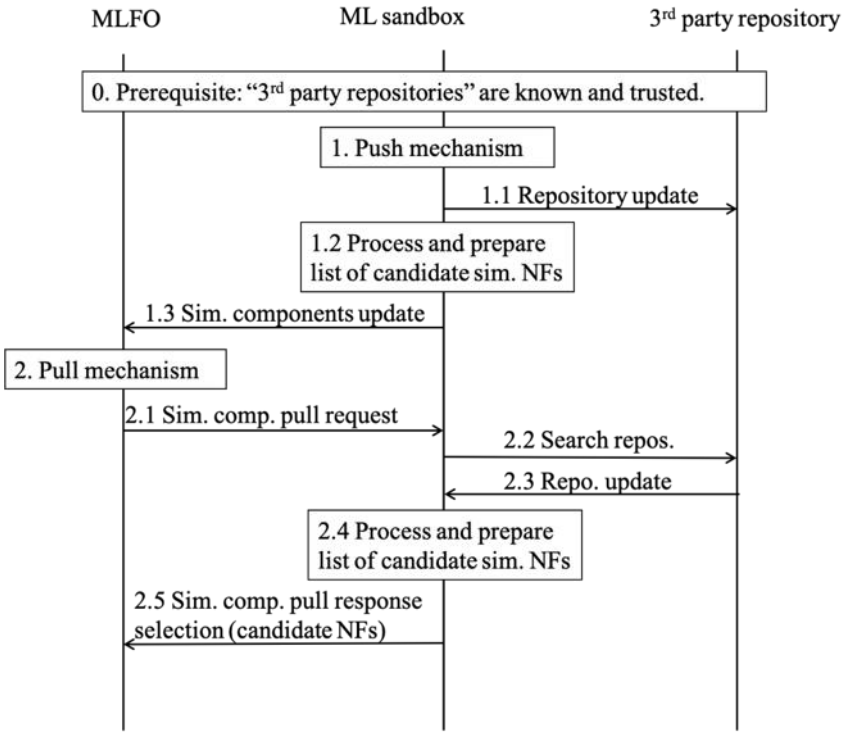


**Figure 3 – Sequence diagram of capability discovery/negotiation for third-party simulation components**

**Prerequisite**: MLFO knows the list of third-party repositories, offline configured, trusted, secure channels. Simulation components in the repositories are described using simulation environment metadata.

Two methods are considered according to the type of notification:

**1. Push mechanism:**

1.1 Capability updates from simulator repositories are performed through the evaluator plugin of the ML sandbox. New information can also be automatically provided as and when a simulation platform (e.g., ns-3) releases a particular capability (e.g., MIMO).

1.2 New/refreshed information is processed to provide a list of candidate simulated NFs.

1.3 The corresponding metadata is published to trusted MLFOs.

**2. Pull mechanism:**

2.1 MLFO queries the ML sandbox for simulation capabilities based on simulation environment metadata.

2.2 The evaluation plugin extends the query to trusted repositories.

2.3 Repositories respond with a list of components matching the simulation environment metadata.

2.4 New/refreshed information is processed to provide a list of candidate simulated NFs.

2.5 The corresponding metadata is published to trusted MLFOs.

For both push or pull mechanisms, the MLFO obtains candidate simulation environments (list of NFs, simulation components, corresponding configurations, connections, data handling adaptors) for the ML use case. The operator selects an optimal configuration and deploys it in the ML sandbox. In addition, data handling and other underlay changes are applied based on the selected configuration.

NOTE 1 – Simulation resources in the repositories are described using simulation environment metadata.

NOTE 2 – As an example of 3rd party NF, ns-3 can be selected to simulate a specific deployment of IEEE 802.11ac WLANs.

NOTE 3 – Based on the use case requirements, the list of potential NFs is narrowed. For instance, NFs can have associated information (via simulation environment metadata) such as "running time", "billing aspects", "accuracy", etc.

### 7.2.2 Monitor health

Health monitoring is meant to ensure the proper behaviour of the simulation components in the ML sandbox. The sequence diagram is shown in Figure 4.

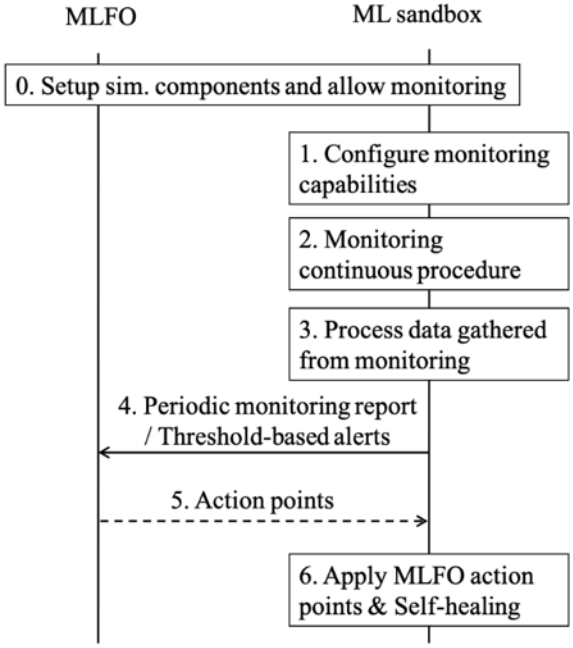


**Figure 4 – Sequence diagram of monitor health**

**[Prerequisite]** Simulation components have been set up, and the ML profile allows for monitoring of simulation components.

1. ML sandbox sets up the resources for monitoring the simulation components through the Simulation designer.
2. Monitoring is carried out based on the continuous flow of data generated by simulation components, including simulation data (SRC and SINK nodes), regression tests, reporting from simulation modules, etc.
3. Data gathered from monitoring is processed to be delivered in a standard manner.
4. Periodical reports are generated and sent to the MLFO, according to the ML intent specification. Alternatively, threshold-based alerts can be activated when undesired events occur, which are also reported to the MLFO.
5. MLFO may take action after processing periodical reports or threshold-based alerts (covered in "Manage simulated ML underlay").
6. ML sandbox applies action points suggested by MLFO and/or self-healing actions.

### 7.2.3 Validate input/output data

Validate input data (check that demand mapping can be fulfilled at the simulated ML underlay) and simulated output data (feasibility of trained models, accuracy of generated data, etc.). For instance, testing techniques such as equivalence partitioning or centroid positioning can be applied to validate the diversity and the quality of the data generated by simulators (e.g., for validating the synthetic data generated by GANs). The sequence diagram is shown in Figure 5.

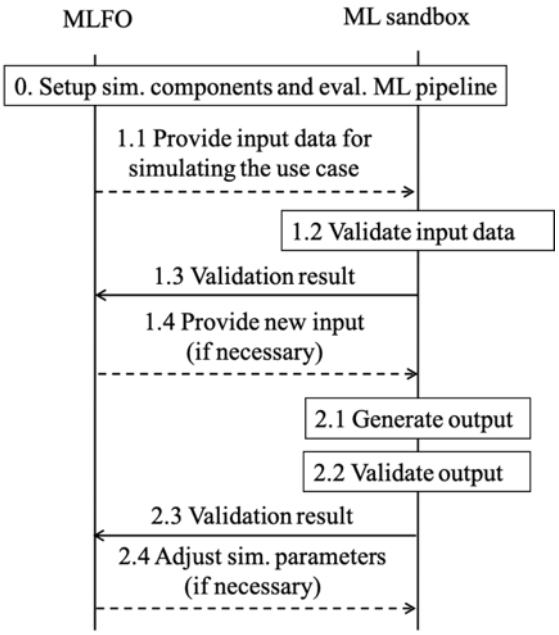


**Figure 5 – Sequence diagram of validate input/output data**

**[Prerequisite]** Simulation components and the evaluation ML pipeline have been set up.

**For input data:**

1.1 ML sandbox receives input data from the ML profile to configure the simulation environment.

1.2 Input data is validated by the simulation designer.

1.3 A response with the validation result is provided to the MLFO.

1.4 MLFO may provide a new ML profile or modify considerations.

**For simulated output data:**

2.1 ML sandbox receives output data from the ML use case (training dataset, trained ML model, evaluation of ML model).

2.2 Output data is validated by the simulation post-processor.

2.3 The result of the validation is sent to the MLFO

2.4 MLFO may provide a new ML profile or modify considerations.

### 7.2.4 MLFO-triggered operations

The MLFO-triggered operations are generally defined in Figure 6.

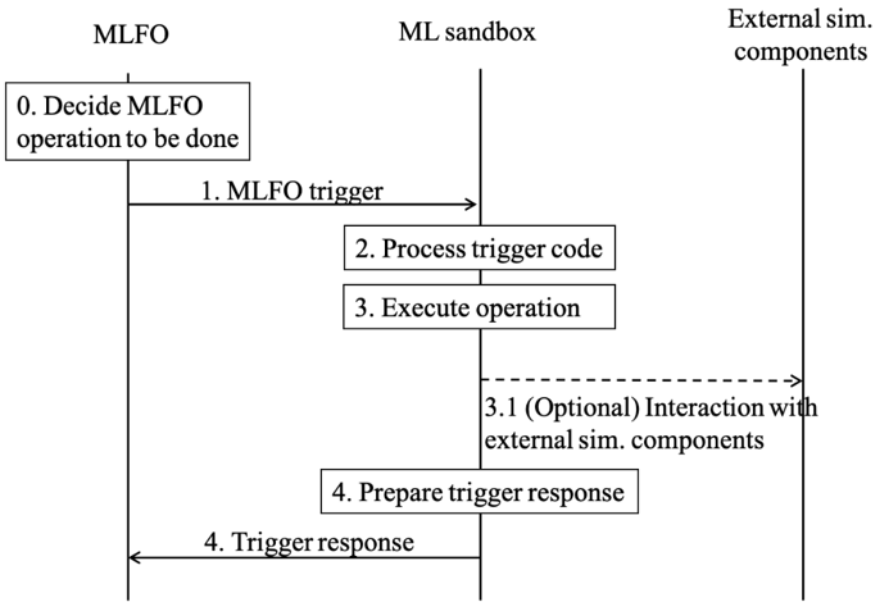


**Figure 6 – Sequence diagram of MLFO-triggered operations**

The set of MLFO-triggered operations are as follows:

**1) Setup environment for ML use case:**

**[Prerequisite]** Capability discovery is done (see clause 7.2.1).

1. The MLFO sends a request to the **Simulation designer** to set up the environment for the ML use case in the sandbox. This request should contain the selected candidate simulation environment, as a result of the procedure in clause 7.2.1.
2. The **Simulation designer** prepares the environment:
   a) May include downloading and installing simulation components.
   b) Data handling plumbing for the simulated ML underlay includes integration with brokers and data storage units [ITU-T Y.3174]. A **plugin** to interact with 3rd party applications can also be employed for the setup.
3. A response is provided by the **simulated ML underlay** to the **Simulation designer** (e.g., traces) through the **plugin**, which relays it to the MLFO after post-processing in the **post-processor** (e.g., OK/NOK).

   NOTE 1 – Environment setup may use the policies from MLFO and demand mapping (e.g., the MLFO ML profile may require isolation of resources). The ML underlay may also be configured based on the use case specification and based on features extracted from the live ML underlay to be reproduced.

   NOTE 2 – Setting up the evaluation ML pipeline may include the download of ML models from marketplaces [ML5G-I-203] and serving of models to create evaluation blocks [ML5G-I-227]. The ML model deployment in the evaluation ML pipeline is defined through reference point 3 of clause 8.3.1 (between MLFO and ML pipeline) in [ML5G-I-227] and the sequence diagram of clause 8.2.

   NOTE 3 – Environment setup may include the installation of third-party applications or setting up interfaces, establish connections via sockets, etc.

**2) Validate environment for ML use case:**

**[Prerequisite]** Setup environment for the ML use case is done (see clause 6.2.2).

1. MLFO requests the execution of sanity tests on a specific simulation environment for the ML use case.

2. Simulation designer selects and executes the test suite (interaction with 3rd party applications can be done through the evaluation plugin).
3. The output of the test suite is processed and output to the MLFO.

   NOTE – For instance, installing a third-party simulation tool may output a set of traces (or logs) indicating the result of installing submodules. This information should be post-processed to assess whether the final installation procedure was successfully accomplished regarding the use case requirement. Example: if the installation of the LTE module failed, but the use case was meant for Wi-Fi (which module was successfully installed), then the result of the installation is satisfactory.
4. The MLFO decides whether the validation results are acceptable or not (may include some basic tests and KPIs).

**3) Manage simulated ML underlay:**

1. The MLFO sends a trigger for modifying/updating the simulated ML underlay. Information on changes is included in the updated ML profile.
2. The Simulation driver configures the simulated ML underlay accordingly (e.g., adapt configuration parameters, specify desired output…). Besides, there is an information exchange between the live and simulated ML underlays (for mimicry purposes).
3. The ML Sandbox responds to the trigger with the information related to changes done in the simulated ML underlay (OK/NOK, changelog, etc.).

   NOTE – Example: as a result of network dynamics, some path-loss parameters used by the ML Sandbox subsystem may vary over time. To address this issue, the MLFO keeps track of those changes and provides feedback to the simulation designer to update the necessary simulation parameters.

**4) Evaluate output of ML model:**

**[Pre-requisite]** The environment has been set up in THE Sandbox for the ML use case.

1. The MLFO sends a request to the **Simulation composer** in the ML sandbox to run the simulated ML underlay.
2. The **Simulation composer** uses the "**evaluator plugin**" to input the output of the ML model. The plugin can also be used to interact with the 3rd party application (i.e., translate commands from the MLFO to simulator-oriented instructions).
3. Evaluate the output of an ML model: the evaluation platform provides a report, which is processed by the **post-processor**.
4. The processed report is sent back to the MLFO. The report can also include synthetic data for training.

   NOTE 1 – the information extracted from the simulated ML underlay needs to be post-processed according to the desired output specified in the use case and the characteristics of the simulated ML underlay. The evaluation result can be an OK/NOK message, a percentage of reliability, a set of KPIs gather from the evaluation procedure, etc.

   NOTE 2 – Post-processing for third-party simulation tools may include the conversion of raw data into meaningful information (e.g., average or deviation on KPIs).

**5) ML model training:**

**[Requirement]** ML model is served and evaluation blocks are ready.

1. MLFO sends a trigger for training an ML model at the sandbox.
2. Model training is performed at the simulated ML underlay through the **Simulator composer**.
3. The trained ML model is post-processed (e.g., compressed, pruned) according to the use case and the policies and capabilities (time constraints, link capacity for exchanging information, storage capabilities, etc.).
4. The trained model is included in the response sent to the MLFO.

Table 1 details the definition of MLFO-triggered operation codes.

**Table 1 – Definition of MLFO-triggered operation codes**

| Code | Parameters | Time sync / dependencies | Description |
|---|---|---|---|
| SANDBOX-TRIGGER-001 | Request type, ML profile | Time sync = yes (evaluation blocks) | Request to prepare the simulation environment (both simulated ML underlay and evaluation ML pipeline) to train, test, and evaluate ML models in the ML sandbox. |
| SANDBOX-TRIGGER-002 | Request type, validation type, acceptance criteria | Time sync = yes (sanity checklist, test suite, etc.) | Request to validate (sanity check) the deployed simulation environment. Used by the MLFO to determine whether to take action to fix potential deployment issues, proceed with ML model evaluation in the sandbox, etc. |
| SANDBOX-TRIGGER-003 | Request type, update type, updated ML profile | Time sync = yes (updated simulation components) | Request to modify/update the simulated ML underlay according to updates in policies, changes in the live ML underlay, potential failures of previous simulated functions, etc. |
| SANDBOX-TRIGGER-004 | Request type, ML profile, ML model output | Time sync = yes (evaluation results) | Request to evaluate the impact of the output of an ML model in the simulated ML underlay, so that some insights can be provided before applying that output to the live ML underlay. |
| SANDBOX-TRIGGER-005 | Request type, ML profile | Time sync = no | Request to train an ML model in the ML sandbox. |

### 7.2.5 Sandbox asynchronous messages

ML sandbox asynchronous messages are generally defined in Figure 7.

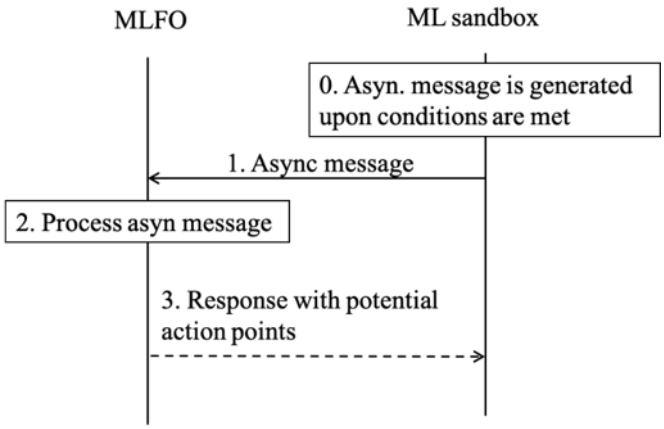


**Figure 7 – Sequence diagram of sandbox asynchronous messages**

**[Requirement]** Asynchronous messages are generated upon certain conditions being met, which are specific to the different types of events.

1. The ML sandbox sends an asynchronous message to the MLFO.
2. The message is processed and potential action points are defined.
3. MLFO sends a response to the ML sandbox.

The set of sandbox asynchronous messages are defined in Table 2.

**Table 2 – Definition of ML Sandbox asynchronous message codes**

| Code | Parameters | Time sync / dependencies | Description |
|---|---|---|---|
| SANDBOX-ASYNC-001 | Status code / error code / detailed report (conditional) | Threshold-based alert is fired / miss-behaviour is detected / keep-alive message | Report the health status of the simulation components |
| SANDBOX-ASYNC-002 | Update type / changelog / additional information on implications of update | Updated on simulation components is notified/discovered to/by the ML sandbox | Report an update on security, accounting, licensing requirements of simulation components |
| SANDBOX-ASYNC-003 | Alert type / forecasted results / additional information on potential failure points | Threshold-based alert based on trend analysis (e.g., service at risk) | Proactive behaviour trend identification of simulation components and sandbox resources |
| SANDBOX-ASYNC-004 | Report type / updated simulation component metadata | Change on simulation component is noticed | Report an update on simulation component metadata |
| SANDBOX-ASYNC-005 | Report type / updated level of intelligence of simulation components | Change on the intelligence level of a simulation component | Report the simulation environment intelligence level |

## 7.3 APIs

### 7.3.1 Capability discovery/negotiation request API

**API Description**: Discover third-party simulation components that can be used to perform use-case-specific simulations in the ML sandbox. A list of updated capabilities to be used by MLFO to find candidate simulation environments for the ML use case.

**Request (pull mechanism)**: MLFO → ML sandbox subsystem

| Information element | Type | Mandatory/Optional/ Conditional | Description |
|---|---|---|---|
| Request ID | Integer | Mandatory | Identifier of the request, indicating "capability discover/negotiation" |
| ML profile | Json | Mandatory | Includes metadata defining policies, requirements, constraints, etc. |

**Response (or push method)**: ML sandbox subsystem → MLFO

| Information element | Type | Mandatory / Optional / Conditional | Description |
|---|---|---|---|
| List of simulation components | Json | Mandatory | Updated capability list of the available simulation components |
| Simulation environment metadata | Json | Mandatory | Metadata can contain information such as installation/execution requirements, capabilities of simulated NFs, performance indicators, configurable parameters, maturity (alpha/beta), etc. |

NOTE 1 – As an example of 3rd party NF, ns-3 can be selected to simulate a specific deployment of IEEE 802.11ac WLANs.

NOTE 2 – Based on the use case requirements, the list of potential NFs is narrowed. For instance, NFs can have associated information (via simulation environment metadata) such as "running time", "billing aspects", "accuracy", etc.

### 7.3.2 Monitor health reporting API

**API Description**: Report the health status of the simulation components, so that the MLFO can consider taking healing actions.

**Request (periodical or responsive)**: ML sandbox subsystem → MLFO

| Information element | Type | Mandatory / Optional / Conditional | Description |
|---|---|---|---|
| Notification ID | Integer | Mandatory | Identifier of the request, indicating "monitoring report" |
| Health status | Integer | Mandatory | Code indicating the current status of the simulation components (green, yellow, orange, red) |
| Severity | Integer | Optional | Code indicating the severity of the potential anomalies identified (critical, major, minor, normal, or clear) |
| Monitoring logs | String list | Optional | Raw data resulting from monitoring |
| Alerts | Key-value pairs | Optional | Threshold-based alerts |
| Suggested action points | String list | Optional | List of suggested action points to fix the potential reported issues |

NOTE – Monitoring is carried out based on the continuous flow of data generated by simulation components, including simulation data (SRC & SINK nodes), regression tests, reporting from simulation modules, etc.

### 7.3.3 Input/Output validation reporting API

**API Description**: This API is used to report the status of input/output data used/generated at/by the ML Sandbox. This information can be used by the MLFO to generate new datasets, re-train ML models with different configurations, update simulation components, etc.

**Requests**: ML sandbox subsystem → MLFO

| Information element | Type | Mandatory / Optional / Conditional | Description |
|---|---|---|---|
| Notification ID | Integer | Mandatory | Identifier of the request, indicating "input/output validation result" |
| Validation type | Integer | Mandatory | Indicates the type of validation performed (e.g., input data to configure simulation parameters or output training data validity) |
| Result of validation | Integer | Mandatory | "Success" or "fail" |
| Warnings | String list | Optional | Detailed information regarding potential issues or misbehaviours observed from the current input/output data |
| Error details | String list | Conditional | Detailed information regarding the errors thrown during the validation procedure |

NOTE – Validate input data (check that demand mapping can be fulfilled at the simulated ML underlay) and simulation output data (feasibility of trained models, accuracy of generated data, etc.). For instance, testing techniques such as equivalence partitioning or centroid positioning can be applied to validate the diversity and the quality of the data generated by simulators (e.g., for validating the synthetic data generated by GANs).

### 7.3.4 MLFO-triggered operations API

**API Description**: This API is used for the MLFO-triggered operations defined in clause 7.2.4.

**Request (pull mechanism)**: MLFO → ML sandbox subsystem

| Information element | Type | Mandatory / Optional / Conditional | Description |
|---|---|---|---|
| Message ID | Integer | Mandatory | Identifier of the message, indicating "ML-triggered operation" |
| Operation code | Integer | Mandatory | Code of the operation to be performed |
| Policies & Requirements | Json | Conditional | Metadata including policies and requirements |
| Simulation environment metadata | Json | Conditional | Includes simulation configuration, available resources, time constraints, etc. |

**Response (or push method)**: ML sandbox subsystem → MLFO

| Information element | Type | Mandatory / Optional / Conditional | Description |
|---|---|---|---|
| Message ID | Integer | Mandatory | Identifier of the message, indicating "ML-triggered operation" |
| Response code | Integer | Mandatory | Code of the operation response (OK, bad request, error, etc.) |

| Information element | Type | Mandatory / Optional / Conditional | Description |
|---|---|---|---|
| Response data | (variable) | Conditional | Depending on the request type, different response data types can be provided (e.g., training dataset, trained ML model, validated ML model) |

### 7.3.5 Sandbox asynchronous messages API

**API Description**: This API is used for the Sandbox asynchronous messages defined in clause 7.2.5.

**Request**: ML sandbox subsystem → MLFO

| Information element | Type | Mandatory / Optional / Conditional | Description |
|---|---|---|---|
| Message ID | Integer | Mandatory | Identifier of the message, indicating "Sandbox asynchronous message" |
| Message code | Integer | Mandatory | Code of the asynchronous message type |
| Additional information | String list | Conditional | Additional information related to the message type |