ETSI TR 126 927 V19.0.0 (2026-01)

TECHNICAL REPORT

LTE; 5G;
Codec for Immersive Voice and Audio Services (IVAS);
Performance characterization
(3GPP TR 26.997 version 19.0.0 Release 19)

Reference
DTR/TSGS-0426927vj00

Keywords
5G,LTE

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Legal Notice

This Technical Report (TR) has been produced by ETSI 3rd Generation Partnership Project (3GPP).

The present document may refer to technical specifications or reports using their 3GPP identities. These shall be interpreted as being references to the corresponding ETSI deliverables.

The cross reference between 3GPP and ETSI identities can be found at 3GPP to ETSI numbering cross-referencing.

Modal verbs terminology

In the present document "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Contents

Foreword

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

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

Version x.y.z

where:

- x the first digit:
  - 1 presented to TSG for information;
  - 2 presented to TSG for approval;
  - 3 or greater indicates TSG approved document under change control.
- y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.
- z the third digit is incremented when editorial only changes have been incorporated in the document.

In the present document, modal verbs have the following meanings:

shall indicates a mandatory requirement to do something

shall not indicates an interdiction (prohibition) to do something

The constructions "shall" and "shall not" are confined to the context of normative provisions, and do not appear in Technical Reports.

The constructions "must" and "must not" are not used as substitutes for "shall" and "shall not". Their use is avoided insofar as possible, and they are not used in a normative context except in a direct citation from an external, referenced, non-3GPP document, or so as to maintain continuity of style when extending or modifying the provisions of such a referenced document.

should indicates a recommendation to do something

should not indicates a recommendation not to do something

may indicates permission to do something

need not indicates permission not to do something

The construction "may not" is ambiguous and is not used in normative elements. The unambiguous constructions "might not" or "shall not" are used instead, depending upon the meaning intended.

can indicates that something is possible

cannot indicates that something is impossible

The constructions "can" and "cannot" are not substitutes for "may" and "need not".

will indicates that something is certain or expected to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

will not indicates that something is certain or expected not to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

might indicates a likelihood that something will happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

might not indicates a likelihood that something will not happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

In addition:

is (or any other verb in the indicative mood) indicates a statement of fact

is not (or any other negative verb in the indicative mood) indicates a statement of fact

The constructions "is" and "is not" do not indicate requirements.

Introduction

This clause is optional. If it exists, it shall be the second unnumbered clause.

1 Scope

Editor's note: to be defined based on the content of the TR

2 References

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

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

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

[2] 3GPP TR 22.874: "Study on traffic characteristics and performance requirements for AI/ML model transfer".

[3] Video-Based Sign Language Digit Recognition for the Thai Language: A New Dataset and Method Comparisons. https://www.scitepress.org/Papers/2023/116437/116437.pdf

[4] Cao, Junli, et al. "Real-time neural light field on mobile devices." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2023.

[5] 3GPP TS 22.261: "Service requirements for the 5G system (5GS)".

[6] 3GPP TR 23.700-80: "Study on 5G system support for AI/ML-based services".

[7] 3GPP TS 23.501: "System architecture for the 5G System (5GS)".

[8] 3GPP TS 23.502: "Procedures for the 5G System (5GS)".

[9] 3GPP TS 23.503: "Policy and charging control framework for the 5G System (5GS)".

[10] 3GPP TS 23.288: "Architecture enhancements for 5G System (5GS) to support network data analytics services".

[11] 3GPP TS 26.501: "5G Media Streaming (5GMS); General description and architecture".

[12] 3GPP TS 22.501: "System architecture for the 5G System (5GS)".

[13] 3GPP TS 23.228: "IP Multimedia Subsystem (IMS); Stage 2".

[14] Cunningham, P., Cord, M., Delany, S.J. (2008). Supervised Learning. In: Cord, M., Cunningham, P. (eds) Machine Learning Techniques for Multimedia. Cognitive Technologies. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-75171-7_2.

[15] Open Neural Network Exchange (ONNX), https://onnx.ai.

[16] The Khronos NNEF Working Group, "Neural Network Exchange Format", https://www.khronos.org/registry/NNEF/specs/1.0/nnef-1.0.5.html.

[17] "ISO/IEC 15938-17:2024 Information technology - Multimedia content description interface - Part 17: Compression of neural networks for multimedia content description and analysis", Edition 2, ISO/IEC, January 2024.

[18] Y. Matsubara, R. Yang, M. Levorato and S. Mandt, "Supervised Compression for Resource-Constrained Edge Computing Systems," 2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), Waikoloa, HI, USA, 2022, pp. 923-933, doi: 10.1109/WACV51458.2022.00100.

[19] Y. Matsubara, D. Callegaro, S. Singh, M. Levorato and F. Restuccia, "BottleFit: Learning Compressed Representations in Deep Neural Networks for Effective and Efficient Split Computing," 2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM), Belfast, United Kingdom, 2022, pp. 337-346, doi: 10.1109/WoWMoM54355.2022.00032.

[20] 3GPP TR 26.847: "Evaluation of Artificial Intelligence and Machine learning in 5G media services".

[21] TensorFlow. (2024). TensorFlow: An open-source machine learning platform. Retrieved from https://www.tensorflow.org/

[22] PyTorch. (2024). PyTorch: An open-source machine learning library. Retrieved from https://pytorch.org/

[23] 3GPP TS 26.511: "5G Media Streaming (5GMS); Profiles, codecs and formats".

[24] 3GPP TS 26.113: "Real-Time Media Communication; Protocols and APIs.

[25] 3GPP TS 26.114: "IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction".

[26] AI Model Efficiency Toolkit (AIMET), https://github.com/quic/aimet

[27] Fraunhofer NNCodec: https://github.com/fraunhoferhhi/nncodec.

[28] "Application and Verification of NNC in Different Use Cases", MPEG document N0450, MPEG Video Coding ISO/IEC JTC 1/SC 29/WG 04, January 2024. https://www.mpeg.org/wp-content/uploads/mpeg_meetings/145_OnLine/w23554.zip

[29] "White Paper on Neural Network Compression", MPEG document N139, ISO/IEC JTC 1/SC 29/AG 03, January 2024. https://www.mpeg.org/wp-content/uploads/mpeg_meetings/145_OnLine/w23564.zip

[30] ISO/IEC JTC 1/SC 29/WG 04 N0598, "Common Test and Training Conditions for FCM", 2024-12-13.

[31] 3GPP TS 26.506: "5G Real-time Media Communication Architecture (Stage 2)".

[32] ITU-T T.840.1| ISO/IEC 6048-1 Information technology — JPEG AI learning-based image coding system

[33] ISO/IEC 6048-3 Information technology — JPEG AI learning-based image coding system reference software

[34] JPEG AI Is Coming: What You Need to Know - Streaming Learning Center Available: https://streaminglearningcenter.com/codecs/jpeg-ai-is-coming-what-you-need-to-know.html

[35] ISO/IEC JTC 1/SC29/WG1 N100600, CPM "JPEG AI Common Training & Test Conditions v8.0", 100th Meeting, Covilhã, Portugal, July 2023.

3 Definitions of terms, symbols and abbreviations

3.1 Terms

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

Autoencoder: Type of artificial neural network used for unsupervised learning, primarily for dimensionality reduction, feature learning, and data reconstruction.

AI/ML model: A mathematical representation that has been trained to recognize patterns or make decisions based on data.

AI/ML model data: Internal components of a trained neural network, including its architecture, weights, biases, and parameters that together define how the inputs are processed to generate the outputs of the AI/ML model.

Model inference: Process by which a deployed machine learning model generates a result [5].

AI/ML task, AI/ML media processing: A software process that executes an ML algorithm to accomplish a specific function.

Inference engine: Functionality that provides runtime environment for a machine learning model and exposes corresponding machine learning model inference capability [5].

AI/ML model subset, AI/ML sub-model: A discrete component of a larger AI/ML model that can perform inference independently.

AI/ML model composition: The composition of an AI/ML model into one or more AI/ML model subsets.

AI/ML model split points: The points in an AI/ML model where the model can be split into multiple distinct AI/ML model subsets.

AI/ML inference endpoint: A UE or Network inference engine that executes an AI/ML model or a model subset

Split operation: Operation which consists in splitting an AI/ML model into AI/ML model subsets, a head subset that takes the same input as the full AI/ML model, and a tail subset that takes the same output as the full AI/ML model.

Split AI/ML model: An AI/ML model composed of subsets) (e.g. head and tail) that can be distributed across different endpoints.

Split AI/ML model inference: Distributed inference performed on subsets of an AI/ML model across different endpoints.

Tensor: A multi-dimensional array of numerical data used in AI/ML, defined by its shape (number and size of dimensions) and data type (e.g., float32, int64).

Input media data: Raw or preprocessed media data captured from a data source (e.g. camera, microphones), including images, video, text, audio and speech that serves as the input for AI/ML models or AI/ML model subsets.

Output media data: Inference output data that has been post-processed to meet the specific requirements of the targeted AI/ML model and application expectations.

Inference input data: Input media data pre-processed to meet the specific requirements of a target AI/ML model, used as input by the AI/ML model or its head subset to perform inference.

Inference output data: The results produced by the AI/ML model or its tail subset after performing inference.

Intermediate data: Output from the inference process of a head AI/ML model subset, which serves as input to the tail AI/ML model subset and is not the final inference result of the full AI/ML model.

Model update: A partial or full update of a trained model which may include changes to its internal structure and/or related parameters (e.g. weight, biases).

AI/ML data: Any of AI/ML model data, intermediate data, inference input data, inference output data, output media data, training input data, training results data, metadata.

Split point configuration: Settings comprising the necessary interoperable information to configure an endpoint for particular split point.

3.2 Abbreviations

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

5GMS	5G Media Streaming
5GS	5G system
AF	Application Function
AI	Artificial Intelligence
AS	Application Server
AIMET	AI Model Efficiency Toolkit
ANN	Artificial Neural Network
AVC	Advanced Video Coding
CPU	Central Processing Unit
DC	Data Channel
DCAR	DC Application Repository
DCSF	DC Signalling Function
DNN	Deep Neural Network
FCM	Feature Compression for Machines
FL	Federated Learning
GPU	Graphics Processing Unit
HDR	High Dynamic Range
HEVC	High Efficiency Video Coding
HTTP	HyperText Transport Protocol
IMS	IP Multimedia Subsystem
INR	Implicit Neural Representation
LSTM	Long-Short Term Memory
MF	Media Function
ML	Machine Learning
MNO	Mobile Network Operator
MPEG	Moving Picture Expert Group
NeRF	Neural Radiance Field
NLP	Natural Language Processing
NN	Neural Network
NNC	Neural Network Coding
NNEF	Neural Network Exchange Format
NNR	Neural Network Representation
NPU	Neural Processing Unit
ONNX	Open Neural Network eXchange
OTT	Over The Top
PDTQ	Planned Data Transfer with QoS
QoE	Quality of Experience
QoS	Quality of Service
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RTP	Real-time Transport Protocol
SDR	Standard Dynamic Range
SVD	Singular Value Decomposition
UE	User Equipment
UL	Up-Link
VVC	Versatile Video Coding
VTM	VVC Test Model

4 Introduction to AI/ML for media

4.1 General

Artificial Intelligence (AI) is a general concept defining the capability for a system to act based on 2 major conditions:

- The context in which a task is to be done, meaning the value or state of different input parameters.
- The past experience of achieving the same task with different parameter values and the record of potential success with each parameter value.

Machine Learning (ML) is often described as a subset of AI (referred as AI/ML in this document), in which an application has the capacity to learn from the past experience. This learning feature usually starts with an initial training phase to ensure a minimum level of performance when it is placed into service.

Recently, AI/ML has been introduced and generalized in media related applications, ranging from legacy applications such as image classification, speech/face recognition, to more recent ones such as video quality enhancement, natural language processing, and generative AI for media. As research into this field matures, more and more complex AI/ML-based applications requiring higher computational processing is to be expected; such processing involves dealing with significant amounts of data not only for the inputs and outputs into the AI/ML models, but also for the increasing data size and complexity of the AI/ML models themselves. This growing amount of AI/ML related data, together with a need for supporting processing intensive mobile applications (such as VR, AR/MR, gaming, and more), highlights the importance of handling certain aspects of AI/ML processing by the server over 5G system, in order to meet the required latency requirements of various applications.

4.2 Media-based AI/ML use cases and scenarios

4.2.1 Introduction

TR 22.874 [2] has identified a set of use cases for AI/ML with the following key operations:

- AI/ML operation splitting between AI/ML endpoints
  - The AI/ML operation/model is split into multiple parts according to the current task and environment. The intention is to offload the computation-intensive, energy-intensive parts to network endpoints, whereas leaving the privacy-sensitive and delay-sensitive parts at the end UE. The UE executes the operation/model up to a specific part/layer and then sends the intermediate data to the network endpoint, the network endpoint then executes the remaining parts/layers and feeds the inference results back to the UE. Alternatively, the network endpoint may firstly execute the operation/model up to a specific part/layer and then sends intermediate data to the UE, which then executes the remaining parts/layers before consuming the inference results.
- AI/ML model/data distribution and sharing over 5G system
  - Multi-functional mobile terminals might need to switch the AI/ML model in response to task and environment variations. The condition of adaptive model selection is that the models to be selected are available for the UE. However, given the fact that the AI/ML models are becoming increasingly diverse, and with the limited storage resource in a UE, it can be determined to not pre-load all candidate AI/ML models on-board. Online model distribution (i.e., new model downloading) is needed, in which an AI/ML model may be distributed from a network endpoint to the devices when they need it to adapt to the changed AI/ML tasks and environments. For this purpose, the model performance at the UE needs to be monitored constantly.
- Distributed/Federated Learning over 5G system
  - The cloud server trains a global model by aggregating local models partially trained by each end devices. Within each training iteration, a UE performs the training based on the model downloaded from the AI/ML server using the local training data. Then the UE reports the interim training results to the cloud server via 5G UL channels. The server aggregates the interim training results from the UEs and updates the global model. The updated global model is then distributed back to the UEs and the UEs may perform the training for the next iteration.

These operations have been identified as they require exchange of AI/ML and media data over 5G, and in some cases may have some requirements on the QoS for proper operation.

The use cases and scenarios listed in this technical report, which are described in this clause, are based on a selection of the media-based AI/ML use cases identified in TR 22.874 [2].

4.2.2 Object recognition in image and video

4.2.2.1 Introduction

Based on clauses 5.1 and 5.2 of TR 22.874 [2], in this set of use cases, images and video streams are processed to identify and recognize objects and extract some metadata, such as bounding boxes, object labels, movement counters, etc.

The uses cases are applicable for the different topologies described in clause 5.1, including UE inference only, network inference only and split inferences topologies.

The computationally intensive and memory and power consuming AI/ML inference used to perform this processing requires offloading some inference parts from the UE to the edge or a cloud data center.

Split inference of trained AI/ML model(s) for object recognition is distributed between multiple endpoints, typically between the network and UE. Split points may depend on various factors including UE capabilities, network conditions, model characteristics, and user/task specific requirements:

- UE capabilities on running whole or part of model such as the required memory, the processing capabilities, the energy consumption, and the inference latency.
- Network conditions for delivering media and/or the intermediate data. This may include, for example the amount of data to transfer in one shot for an image or at a specific frame rate for video, the required bandwidth in UL and/or DL with different impact on the network load and the related UL and DL network latencies. Network inference latency is also to be considered.
- Model characteristics include split inference with a task-specific model head running on the UE for object recognition. For example, in one UE, the task is to recognize pedestrians, whereas in another it is to recognize traffic signs. The core of the network model as well as the input image/video are the same, but the tasks (and their required task-specific models) in the UEs are different.
- User or task specific requirements. For example, it may be necessary to perform some processing tasks on end-device in order to preserve privacy or because they are delay sensitive operations.

Two main scenarios, both involving either image or video processing are proposed:

a) The UE captures images or video and first passes the input data to the UE inference engine to process the first part of the model (e.g., to preserve privacy). The UE inference engine generates intermediate data which is transmitted to the network inference engine, which completes the remaining part of the model (e.g., process the intensive computations). The final result is transmitted back to the UE or sent to other peer UEs.

b) Unlike the previous scenario, the UE uploads the captures image or video to the network where a network inference processes inputs video/image, then sends back the intermediate data to the UE inference engine executing the remaining layers of the model (e.g., task specific operations) and obtaining the final results.

These scenarios involve the key operation of AI/ML model/data distribution and require the delivery of trained ML model(s) for object recognition to the UE in 5GS, including the selection of models for different tasks or environments and the possible selection of the split points based on the various factors described above

These scenarios also involve the distribution of distributed online training of image and video recognition models based on input from different UEs. Depending on the configuration of the AI/ML training framework, different data may need to be delivered between the UEs and the network. Typically, a shared model in the network is calibrated continuously based on the training results from all UEs. This scenario involves all the three key operations related to AI/ML model distribution, splitting, and distributed/federated learning.

4.2.2.2 Scenario: Sign language translation in real-time communication

Hearing-speech impaired people unable to have a regular voice call with other people can use sign language instead. The sign language may be converted to audio or text in real-time and sent to the unimpaired people. Unimpaired people can still use voice as if they are talking to a hearing person, the voice of the hearing people may be transferred to an avatar's sign language or converted to text being displayed on the screen of the hearing-speech impaired people. This helps hearing-speech impaired people to easily communicate with unimpaired people.

However, sign language AI/ML models typically have several millions of parameters [3] and may require involvement of network AI/ML inference. For privacy reasons, a hearing-speech impaired user might not want to transmit his/her sign language video stream to the IMS network or to the peer user. Therefore, the AI/ML inference for sign language could be split between the UE and IMS.

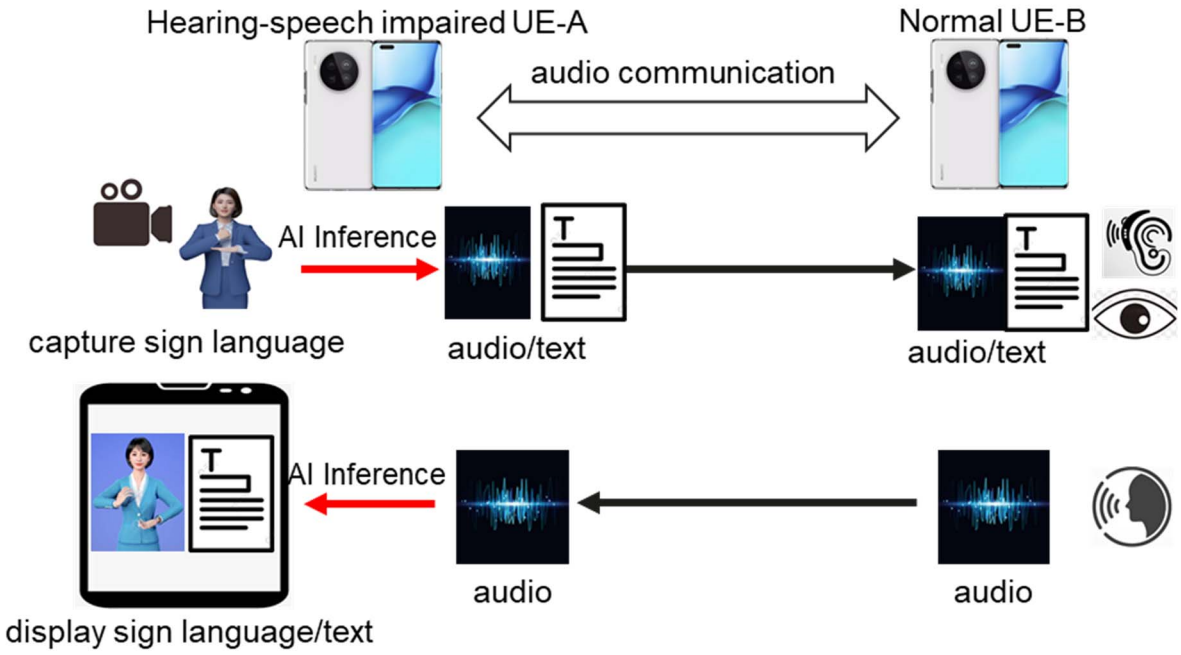


Figure 4.2.2-1: Graphical representation of sign language translation in real-time communication

As illustrated in figure 4.2.2-1, the hearing-speech impaired person UE-A uses a phone to have a voice call with UE-B. UE-A turns on his camera to capture his sign language video, AI/ML inference is performed to translate his sign language to voice or text, the translated voice or text is sent to the UE-B. On the other side, UE-B can use his microphone to talk, the voice of the UE-B is converted to an avatar's sign language video stream or text and sent to hearing-speech impaired person UE-A.

4.2.3 Video quality enhancement in streaming

4.2.3.1 Sender-receiver approaches

4.2.3.1.1 End-to-End neural network-based video coding

Based on clause 5.3 of TR 22.874 [2], in this use case, the sender and receiver apply parts of a DNN model (e.g. an autoencoder model) to enhance the quality of a video stream. An example of an autoencoder DNN is depicted in figure 4.2.3-1:

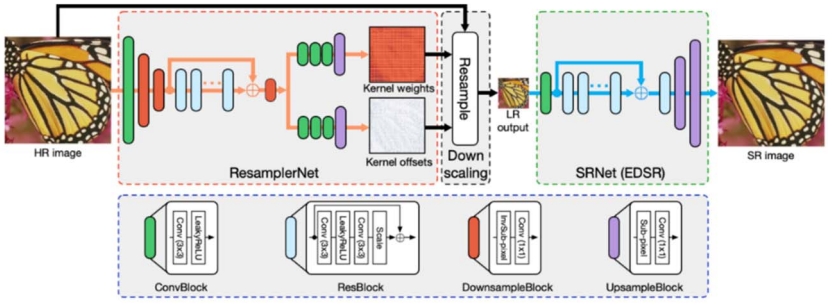


Figure 4.2.3-1: Example of DNN-based Down/Up-scaler

The sender is typically represented by various media functions in the network, which processes the high-fidelity video using the down-scaling part of a pre-trained DNN model to an intermediate data stream that is streamed together with a lower resolution encoding of the video. The receiver (UE) runs an inference algorithm (e.g. the up-scaling part of DNN model) on using the received intermediate data and video stream to produce a high-quality video for rendering.

The main scenario in this use case is about streaming intermediate data from the network for processing on the UE, involving AI/ML data distribution and operation splitting.

This use case covers all scenarios where an intermediate data stream needs to be sent to the receiver, in addition to a low-resolution video.

4.2.3.1.2 Neural network based post-processing for video coding

A neural network (NN) applies post-processing to a decoded video sequence to enhance the quality of the decoded frames. The post-processing is performed outside the coding loop and does not impact the decoding process of the video. Possible post-processing algorithms include:

- Post-filtering: where the output of the video decoder is provided as input to a NN to improve the quality of the decoded frames. Such improvements include removal of video coding artifacts, subjective quality enhancement, etc.
- Super resolution: where a NN is used to increase the resolution of the output video sequence when the resolution of the display is greater than the resolution of the decoded frames. The use of NN-based approaches in super resolution resampling process increases the quality of the resulting resampled frames.
- NN-based HDR enhancement: a NN is applied for example to enhance a SDR video into an HDR-looking video.

In contrast to 4.2.3.1.1, this approach does not use an intermediate data stream.

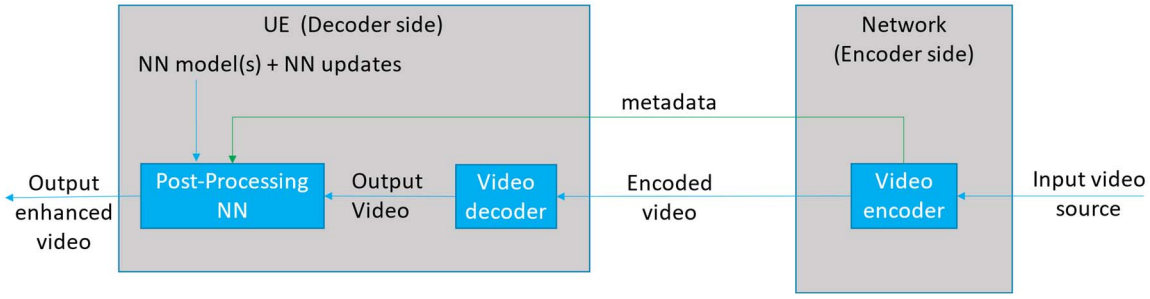


Figure 4.2.3-2: Neural network based post-processing for video coding use-case

Figure 4.2.3-2 depicts a neural-network-based post-processing use-case where pre-trained NN models are used at the receiver to post-process the decoded video to improve the quality. The video encoder processes the input video source to produce and send content-related metadata to the receiver, based on video/image or block, for example. The content-related metadata can be used to select a pre-trained NN model to be applied to a piece of content and to activate or not the selected NN model on it.

4.2.4 Crowd-sourcing media capture

4.2.4.1 Introduction

This use case and its corresponding scenarios are based on clause 6.2 of TR 22.874 [2]. A set of users attending a live concert and capturing the event on their UEs, use a shared (or a set of shared) DNN model(s) to process and improve their respective captured video and/or audio. Audio and video data may be captured in a noisy environment or an environment with poor lighting conditions. Multiple tasks may then be performed on the processed video and/or audio for media content analysis, e.g. to extract lyrics, annotate the video, improve audio and video quality, translate language, anonymize a face, etc. One such task may be creating Implicit Neural Representations (INR) of the event. The UEs share image or video data with a model training server in the network which trains and generates an implicit neural representation, such as a Neural Radiance Field (NeRF), of the event. The UEs request novel or enhanced views from the NeRF. The network may also train a set of NeRFs for the event suitable for different UEs which vary in their inference capabilities and platform support. Device appropriate NeRF models may be downloaded by the UEs and used for view synthesis locally. Developments in INR research have shown the feasibility of running NeRF inference on modern UEs [4].

This use case involves two different scenarios based on either a device inference or a network inference.

4.2.4.2 UE-based inference

The main scenario is to improve the media capture of each UE by using an up-to-date model adapted to the context event.

This scenario may involve the distribution of multiple models to many UEs in a short period of time. The UEs are heterogeneous, running with different types of operating systems, supporting different AI/ML engines/frameworks or having different GPU/CPU/NPU and RAM capabilities available for running the AI/ML service on the UE. This will need the distribution of a huge amount of various AI/ML models adapted to the different UE capabilities. Depending on each user's UE, the UE may request the download of a set of DNN models for device inference.

Moving or changing the environment (localization, energy, processing unit, memory, etc.) may need AI/ML model updates, where the DNN models stored in the network may be adapted or updated during the service.

The AI/ML application may optimize the end-to-end latency (e.g., to achieve latency below 1s) or the expected accuracy level of the inference result (e.g., to achieve image recognition precision of 99%) by modifying the model. The desired latency and/or accuracy level can therefore impact the size of the AI/ML model to be distributed. This can be done by:

- optimizing the model accuracy and latency for on-UE execution. The model accuracy and execution latency are known, and the optimization may result in bandwidth saving.
- compressing the model for reducing the bandwidth usage and improving the delivery latency. This may affect the accuracy of the model.

If an uncompressed model is sent, accuracy is not affected but delivery latency would depend on the size of the model and the network bandwidth.

The distribution of the AI/ML models for many UEs at the same time may also need to serve the models from different endpoints (e.g., cloud, edge, or other UEs), and may use several or different communication links (e.g. unicast, multicast or broadcast).

4.2.4.3 Network-based inference

The main scenario may be the sharing of the input media from multiple sources for network inference, as well as the selection of suitable DNN models according to the UE and/or task.

This scenario requests the UE to upload the media data for network inference. Similarly, to the UE inference, DNN models stored in the network may be adapted or updated during the service for network inferences.

4.2.4.4 Scenario: Crowdsourced Implicit Neural Representation

4.2.4.4.1 Crowdsourcing content

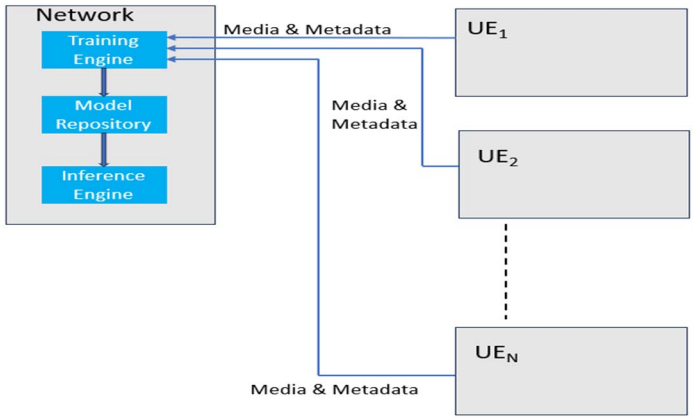


Figure 4.2.4-1: Crowdsourced NeRF creation

As shown on the figure 4.2.4-1 above, in a crowdsourced NeRF generation scenario, an application provider creates NeRF(s) of an event, such as a music concert or a sports match. To train the NeRF(s), the service provider uses training engine(s) in the network. The training data, for example, images or videos with associated metadata such as location and pose of the UE, is received from UEs present in the event (UE_1, UE_2,….UE_N). The training data is pre-processed and used for training the NeRF models. Multiple NeRFs, targeting different UE or Server hardware's, inference capabilities and platforms, may be trained.

4.2.4.4.2 Content Synthesis

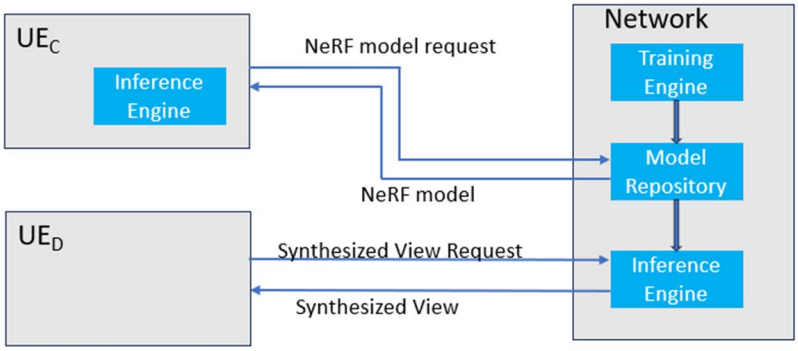


Figure 4.2.4-2 NeRF and synthesized view distribution

In the UE inference scenario of clause 4.2.4.2, a UE with on-device inference capabilities (UEC) requests NeRF model appropriate for its inference capabilities from the network, downloads the model and runs the inference engine to synthesize novel views locally on the device.

In the network inference scenario of clause 4.2.4.3, a resource-constrained UE (UED) may request a particular view of the event from the network, the network inference engine renders the view and it is delivered to the UE.

Those cases are illustrated in the figure 4.2.4-2.

4.2.5 Natural Language Processing (NLP) on speech

4.2.5.1 Introduction

Based on clause 6.3 of TR 22.874 [2], this set of use cases covers a wide range of speech processing use cases, e.g. to perform automatic speech recognition, voice translation, voice commands, speech synthesis, etc.

The AI/ML models for NLP are improved with distributed/federated training using multiple UEs. As more users make use of the service, the quality and accuracy of the models improves. The results of the local training of the models by the UEs are shared with the network.

The main scenario here is about UE downloading a partially trained model identified with its training state for local training and then sharing the results with the network for distributed/federated learning.

4.2.5.2 NLP on Speech in real-time communication

NLP on speech in real-time communication may be done on both UE and network, or fully on the network. A use case which is fully completed on network is described as below and illustrated in figure 4.2.5-1.

UE-B has subscribed to an intelligent translation service. UE-A initiates an audio/video call and establishes a connection between UE-A and UE-B through the IMS network. When it is detected that UE-B has subscribed to an intelligent translation service, the IMS network serving UE-B decodes the audio stream from UE-A, performs speech recognition and translates it into the required language using AI. Then, it sends the translated text along with the video stream or through a data channel.

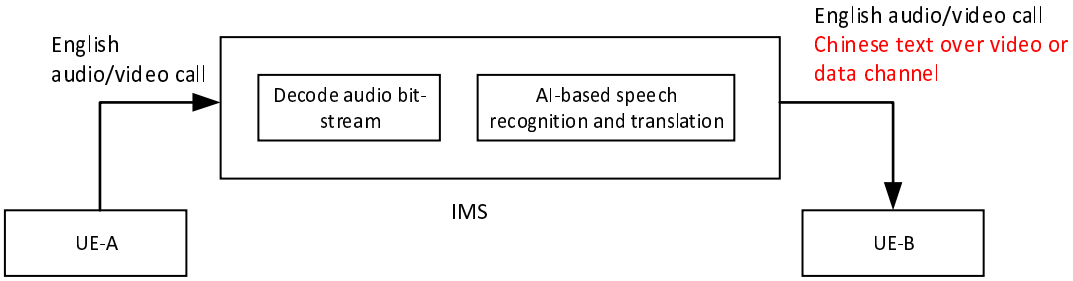


Figure 4.2.5-1: workflow for NLP on speech

Editor's note: Change «chinese text» to 'from language A tp language B'

4.3 Related work

4.3.1 Traffic characteristics and performance requirements for AI/ML model transfer in 5GS

The 3GPP TR 22.874 [2] captures the study of the use cases and the potential performance requirements for 5G system support of AI/ML model distribution and transfer (download, upload, updates, etc.), and identifies traffic characteristics of AI/ML model distribution, transfer, and training for various applications, e.g. video/speech recognition, robot control, automotive, other verticals.

The media related use cases described in 3GPP TR 22.874 [2] are used as a basis for those listed and described in clause 4.2 of this technical report.

The 3GPP TS 22.261 [5] specifies requirements for the 5G system under clause 6.40 as well as KPIs for AI/ML model transfer in 5GS in clause 7.10 for split inference, AI/ML model downloading and federated learning between the UE and the Network Server/Application.

4.3.2 5G System Support for AI/ML-based Services

The 3GPP TR 23.700-80 [6] documents, based on requirements as specified in clauses 6.40 and 7.10 of 3GPP TS 22.261 [5], 5GS assistance to support AI/ML model distribution, transfer, training for various applications, e.g. video/speech recognition, robot control, automotive, etc.

Assistance to AI/ML operations in the application layer is defined in clause 5.46 of 3GPP TS 23.501 [7], with specific improvements also impacting 3GPP TR 23.502 [8], 3GPP TR 23.503 [9] and 3GPP TR 23.288 [10], related to:

- Subscriptions, in particular NEF monitoring events, QoS monitoring and network data analytics.
- Member UE selection assistance functionality, hosted by the NEF to assist the AF to select member UEs that can be used in AI/ML based applications (e.g. Federated Learning).
- Other enhancements related to QoS and parameter provisioning related to expected UE behaviour.

4.3.3 MPEG Feature Compression for Machines

MPEG related work, FCM (Feature Compression for Machines) addresses features compression. FCM is based on existing video compression standards. At the encoder, feature tensors are reduced, converted, and mapped onto packed video frames that may be encoded using encoders such as VVC, HEVC, or AVC, e.g., monochrome 10 bits video frames where the tensor channels are spatially packed. The video decoder outputs the packed video frames which are then processed to restore the feature tensors in their original shape, where the conversion, unpacking and feature restoration may use additional metadata transmitted along with the video bitstream. This standard is under development.

Figure 4.3.3-1 shows a FCM encoder and decoder framework where the original selected trained model is split into two parts NN part1 and NN part 2. The current FCM framework under study includes a trained bottleneck, also called NN feature reduction at the encoder side and NN feature restoration at the decoder side. The parameters need to be trained for each split model and each split point, based on the end accuracy or the quality of reconstructed intermediate data after decoding, e.g. with an MSE-based metric. The NN feature reduction and restoration can be trained while keeping the parameters of the split AI/ML model frozen. The FCM encoder and FCM decoder includes a Feature conversion function mapping features data onto packed video frames and vice versa.

The training of the feature reduction and feature restoration models may not require access to the original training set nor access to the same level of computational power. Current training strategies involve different training sets and using a loss based on metrics such as Mean Square Error to compare reconstructed features with the original intermediate features. Light trained codecs may be designed to fit existing large pre-trained split models without having to perform heavy computations involving the backpropagation of gradient from the final prediction of the tail of the model back to the original intermediate features. These strategies lead to the promising compression the performances detailed below.

The MPEG-FCM group is also considering untrained feature reduction methods that are computed online, enabling a more versatile codec that does not require retraining for each split model, but at the cost of lower compression efficiency.

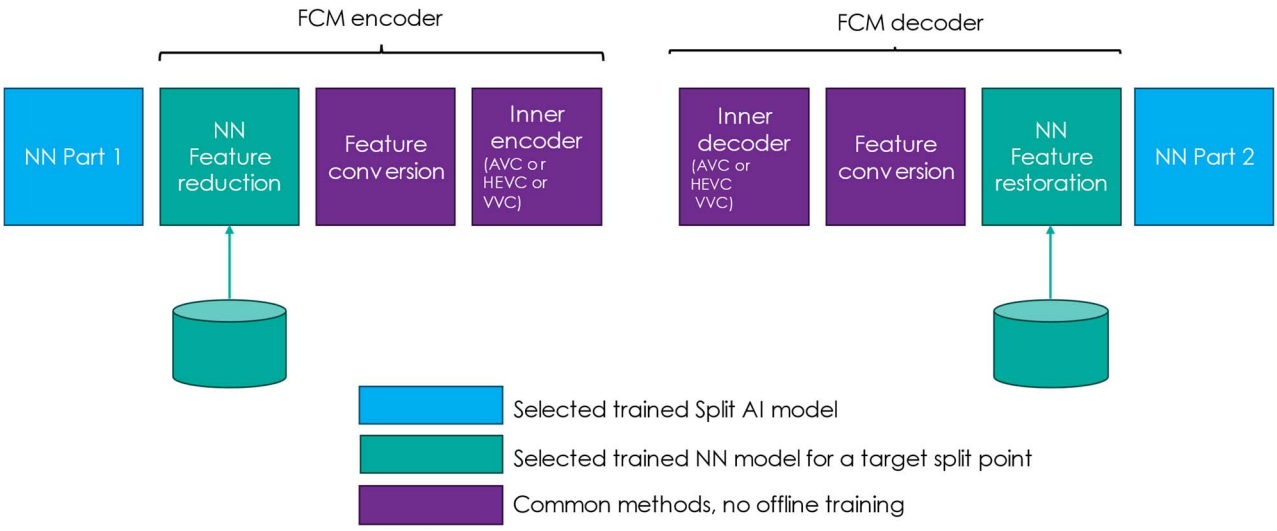


Figure 4.3.3-1: FCM framework

The current performance of FCM comparing to a remote inferencing anchor is 75% overall bitrate reduction over the same range of task performance as the remote inferencing anchor. The tasks include instance segmentation, object detection and object tracking. Remote inferencing refers to the compression of the input content using VVC reference software VTM-12.0 and the inference of the task model at the receiver on the decoded content. The compression ratio of the uncompressed feature size verses the compressed feature size in near lossless setting ranges from 6000:1 to 40000:1 on instance segmentation, object detection and object tracking. The obtained compression ratio of intermediate data while preserving near lossless accuracy is defined within a tolerance of 1% drop in task accuracy, relative to the performance achieved by the original task model operating directly on the input data.

4.4 Additional related work for distributed AI/ML processing and coding

4.4.1 JPEG-AI: learning-based image coding system

The scope of JPEG-AI is the creation of an interoperable learning-based image coding system offering a single-stream, compressed domain representation with significant compression efficiency improvements over image coding standards commonly used today at equivalent subjective quality. The JPEG-AI image compression standard is based on neural networks. It is a multi-task standard; this means that in addition to compression it also targets computer vision tasks such as enhancement or detection of objects in the compressed domain. The standard was completed in January 2025 under ISO/IEC 6048-1/ITU-T T.840.1 [32].

NOTE: The corresponding software is currently also available on https://gitlab.com/wg1/jpeg-ai/jpeg-ai-reference-software

Figure 4.4.1-1 shows a high-level architecture of a JPEG-AI encoder, highlighting key pipelines. In a JPEG-AI encoder, the input image chroma and luma are separated and processed with a transform, which aims to de-correlate the input image information typically using convolutional layers of a neural network, each one followed by nonlinear activation layers; each convolutional layer consists of learnable filters where some of them also perform spatial down-sampling. At this stage, the so-called latent representation is obtained, which can be understood as a compact representation of the input image based on the de-correlation using convolutional layers in the neural network. Hence, the latent representation refers to the representation of the image after this processing of analysis. The statistical redundancy present in the latent representation is exploited by the entropy coding engine to produce the final bitstream to be transmitted or stored.

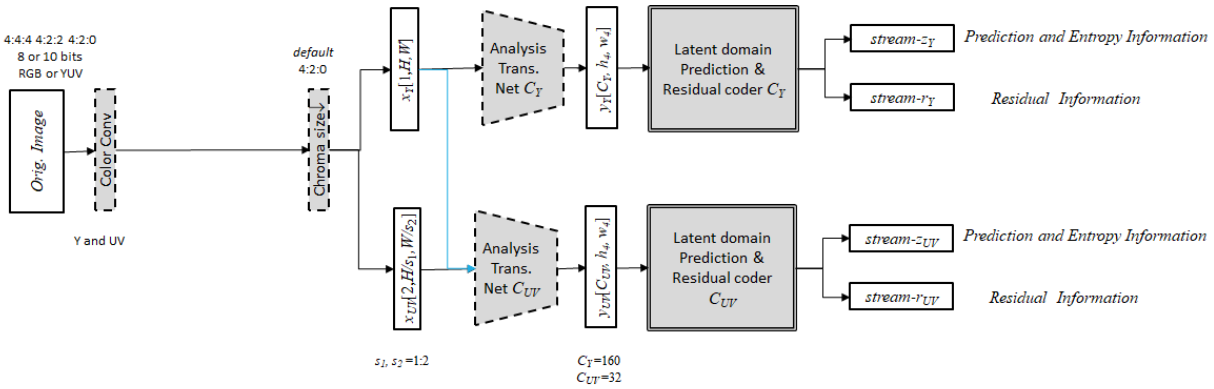


Figure 4.4.1-1: High level encoder diagram for JPEG-AI

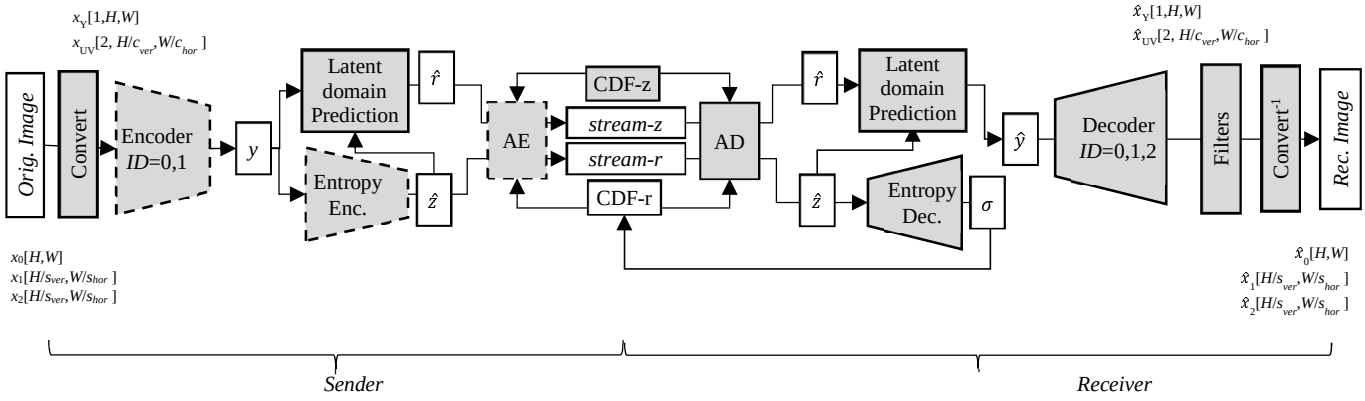


Figure 4.4.1-2: Encoder and decoder diagram of JPEG-AI

Figure 4.4.1-2 shows the combined encoder and decoder diagram of JPEG-AI [32] (like other coding specifications JPEG-AI only defines the decoder part normatively). The encoder receives a source image and converts it to the coding format. This includes colour separation to primary and secondary component, which are compressed using the same sequence of steps: analysis transform, encoder, latent prediction, entropy and residual coding.

The analysis transform (marked as *EncoderID* on Figure 4.4.1-2) produces latent representation of the image y. This latent representation is further compressed to tensor z, which carries information about latent domain prediction and entropy parameters. The tensor z is quantized to zˆ and compressed by arithmetic coder AE with entropy parameters which are part of trained model (known to both encoder and decoder). The latent domain prediction is subtracted from y, and the residual r obtained is quantized to rˆand encoded by arithmetic coder with entropy parameters produced by the decoder from the tensor.

The decoder performs same steps for primary and secondary components, including: a) parsing tensor zˆ(entropy parameters are part of the trained model), b) processing by the decoder to produce entropy parameters for residual decoding, c) processing, by the decoder to produce explicitly signalled components of prediction and latent domain prediction, d) forming reconstructed latent representation of the image yˆ. The synthesis transform for the primary component operates independently of the secondary component synthesis transform.

The synthesis transform (marked as *DecoderID* on Figure 4.4.1-2) processes the latent image representation to create the reconstructed image, which is later processed by series of enhancement filters and converted from coded picture format to output picture format. The latent domain prediction and entropy decoder model are identical for all analysis transform (*EncoderID*=0,1) and synthesis transforms (*DecoderID*=0,1,2).

In addition to the specification, the JPEG consortium has made reference and conformance software available. At the time of this report the software is available on https://gitlab.com/wg1/jpeg-ai/jpeg-ai-reference-software, the software is

also planned as ISO publication ISO/IEC 6048-3 (at time of writing as draft international standard DIS) [33]. In addition, early implementations of JPEG AI are already emerging on mobile devices, with demos shown running on modern smartphones, showcasing high-resolution 4K image decoding, tiling, and arbitrary resolution decoding [34]. Such codec implementation can benefit from GPU hardware acceleration via libraries that implement the neural network processing.

JPEG-AI shows improved image coding results compared to state-of-the-art anchors according to the JPEG group (on ds.jpeg.org reports and results can be found).

In addition, JPEG-AI is going to start a subsequent activity, to support parallel processing of the latent representation (in the network) for different inference tasks. To demonstrate its feasibility, a first evaluation was performed on using the compressed JPEG AI images for computer vision tasks. The setup is shown in Figure 4.4.1-3. In this experiment, resnet-50 was used for the image classification task on either the original image (after resize and crop), the decoded images (after resize and crop) and the latent space (using another classifier for the compressed domain inference). At 0.7 bit per pixel both top-1 and top-5 results under 2 and under 1 percent less accurate compared to inference on the uncompressed image respectively (these results are based on independent evaluation following a JPEG Call for Proposal on this topic of computer vision task performance, for the evaluation procedure see [35]).

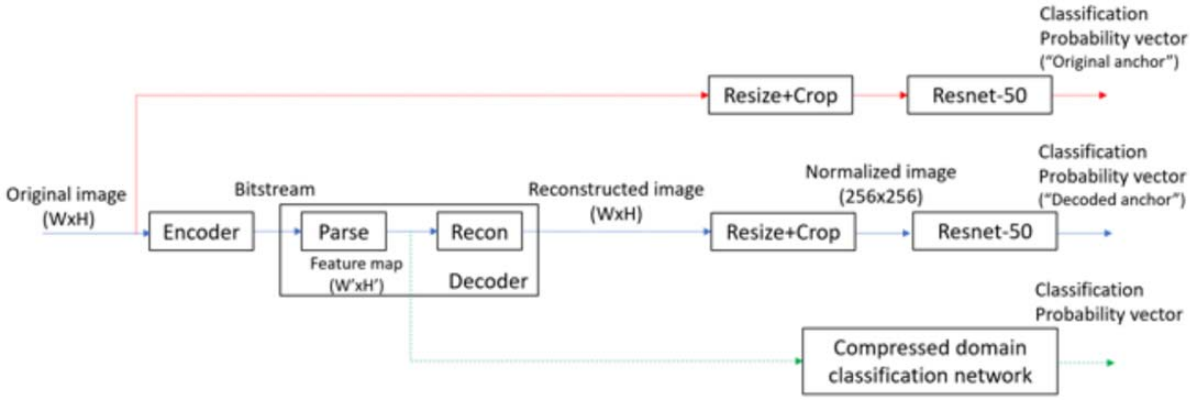


Figure 4.4.1-3 Evaluation of JPEG-AI for parallel image reconstruction and network computer vision task from a single entropy decoded latent representation.

The usage of "latent domain" representation for computer vision task is another claimed feature of the JPEG-AI standard. This still needs more exploration, and additional results may be added when they become available. Latent space-based classification could become more relevant when more JPEG-AI images are available.

5 Media service architecture for AI/ML

5.1 AI/ML Split configurations

5.1.1 AI/ML model composition

An AI/ML model may be splitable, meaning that it may be theoretically represented by several sequential sub-models separated by split points as illustrated for a fully connected artificial neural network (ANN) example in figure 5.1.1-1.

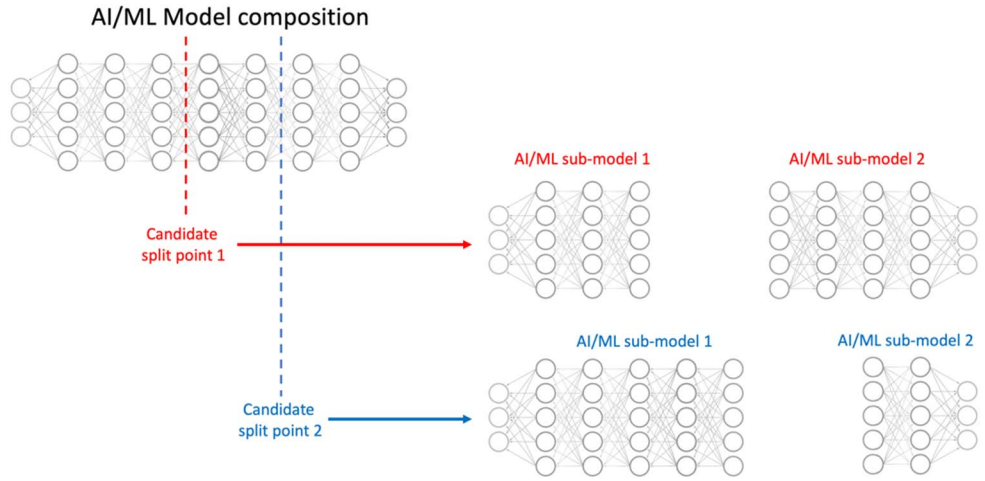


Figure 5.1.1-1: AI/ML model composition examples with a fully connected ANN

In a general case, illustrated in figure 5.1.1-2, several compositions of the same AI/ML model are represented by the AI/ML subsets (M0, M1), (M'0, M'1), or (M "0, M "1, M "2) with split points highlighted in red. The same AI/ML sub-model may be used in different compositions depending on the configurations of the model composition (e.g. M'0 and M"0 according to figure 5.1-1).

In figure 5.1.1-2, (a) and (b) are examples of AI/ML inference endpoints running an AI/ML model composed of two sub-models M0 and M1.

Examples (c) and (d) demonstrate AI/ML split models where M0, M'0 run on the UE while M1, M1' run on the network respectively.

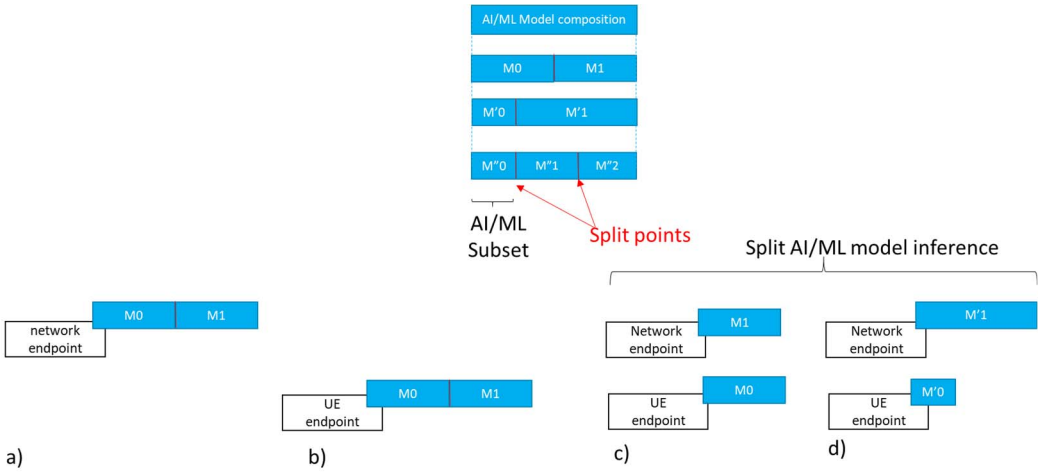


Figure 5.1.1-2: General AI/ML model composition examples

In this document the following working assumptions are made:

- Each sub-model describes a unique part of the inference process.
- The combination of the inference of all sub-models is equivalent to the inference of the entire AI/ML model.
- Several split points, identifying the frontier between AI/ML sub-models, may be identified within an AI/ML model.
- Those split points are predefined and may be selected or re-selected dynamically to adapt to the changing conditions.
- In this report, only service configurations limited to a predefined set of split-points are addressed in this report.

5.1.2 Topologies of split AI/ML model inference

5.1.2.1 Introduction

In the context of split AI/ML models, for which one AI/ML sub-model is run in the UE and the other sub-model in the network, there may be different orders of operations and consequently different media flows depending on where the process is initiated and where the media source is processed.

This clause introduces the different topologies of AI/ML split operations with the media source being in the UE (Clause 5.1.2.2) and in the network (Clause 5.1.2.3).

5.1.2.2 UE as the media source

In this scenario, the media data to be processed by the AI/ML model is in the UE. Two distinct cases are defined:

- The first AI/ML sub-model, also called the head sub-model, (M0 on the figure 5.1.2-1) runs a partial inference in the UE. The intermedia data is then sent to the network and used by the second AI/ML sub-model, also called the tail sub-model, (M1 on the figure 5.1.2-1) that completes the inference process. The result is finally sent back to the UE. The configuration is illustrated in figure 5.1.2-1.

- The media source is sent to the network where the head AI/ML sub-model (M0 on the figure 5.1.2-2) runs a partial inference. The intermediate data is then sent to the UE and used locally by the tail AI/ML sub-model (M1 on the figure 5.1.2-2) that completes the inference process. The result of the inference is available directly in the UE. This configuration is illustrated in figure 5.1.2-2.

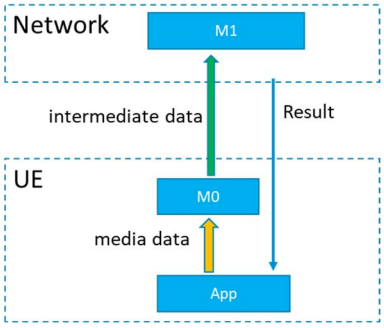


Figure 5.1.2-1: Split AI/ML model inference where the UE is the media data source with first inference endpoint on the UE

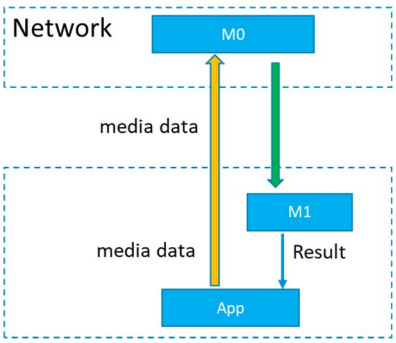


Figure 5.1.2-2: Split AI/ML model inference where the UE is the media data source with first inference endpoint on the network

5.1.2.3 Network as the media source

In this scenario, the media data to be processed by the AI/ML model is in the network. There, the head AI/ML sub-model (M0 on the figure 5.1.2-3) runs a partial inference. The intermediate data is sent to the UE that already has the tail AI/ML sub-model available. This tail AI/ML sub-model (M1 on the figure 5.1.2-3) uses the intermediate data to

complete the inference process. The result of the inference is available directly in the UE. This configuration is illustrated in figure 5.1.2-3.

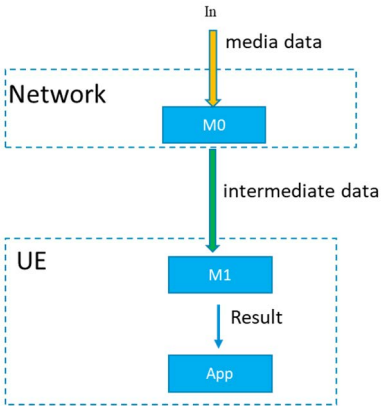


Figure 5.1.2-3: Split AI/ML Model inference where the network is the media source

5.2 Architectures and service flows

5.2.1 Introduction

Considering the related use cases as documented in TR 22.874 [2] and in clause 4.2, basic architectures and corresponding workflows for each scenario are presented.

The basic scenarios are:

1) Delivery of a pre-trained AI/ML model from network to UE, typically at the start of an AI/ML media service, but may also require updates during the service. At the most basic level AI/ML models can be delivered as a file (e.g. TensorFlow SavedModel, PDF5, ONNX file, NNEF file etc.) containing all the necessary information required for the UE to perform on device inference using the delivered model. For split scenarios, a (partial) AI/ML model to be used in the UE may be delivered.

2) Split inference of a pre-trained AI/ML model(s) with two further sub-scenarios:

   a) Basic scenario with an inference in the network or in the UE.

   b) Split scenario with inferences between the network and the UE, where the intermediate data output from the network inference (resp. UE inference) is transferred to the UE (resp. network) to be used as the input for UE inference (resp. network inference). Depending on the characteristics of the intermediate data, it may be practical to consider 5GMS architectures, procedures and/or protocols for the streaming delivery of such intermediate data.

3) Distributed/federated learning using multiple UEs with local training sets, and a central server in the network. Typically, a central model is distributed to UEs for local training. UEs use available local data for local training, and training result updates are sent back to the central server, which aggregates and updates the central model. Global updates on the central model are then distributed to the UEs for continuous training.

NOTE: Compression aspects will be addressed once the digital representation of AI/ML models will be identified together with their associated service requirements (e.g. traffic flow characteristics, latency, bitrate…).

5.2.2 Complete/basic AI/ML model distribution

5.2.2.1 Basic architectures

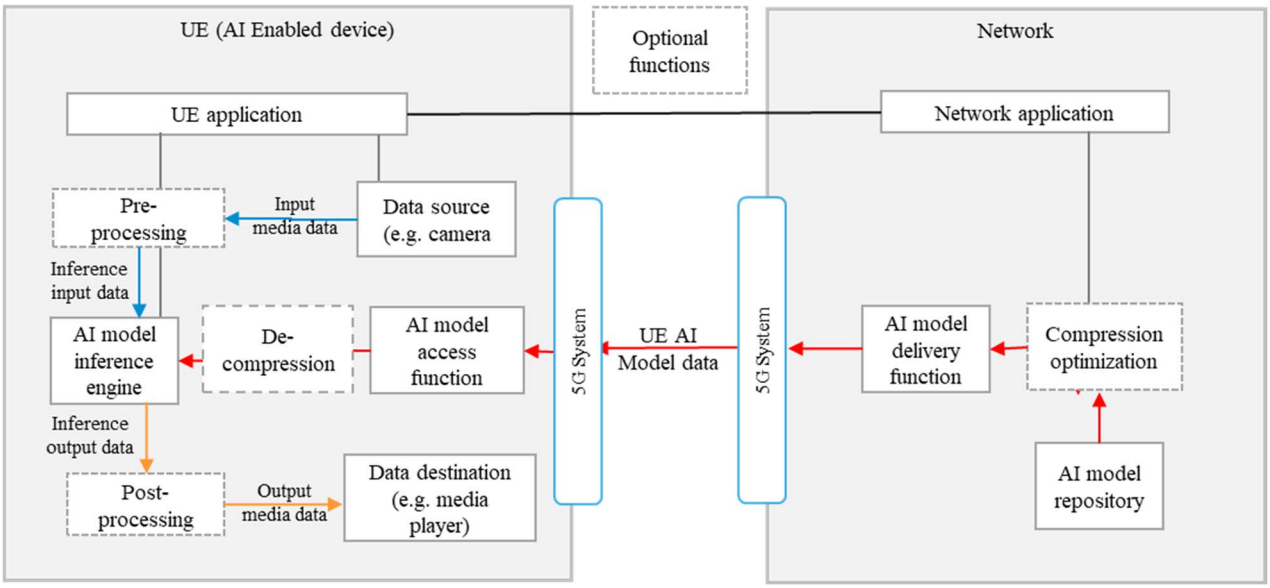


Figure 5.2.2-1: Basic architecture for AI/ML model delivery with inference in the UE

Figure 5.2.2-1 shows a simple basic architecture for AI/ML model delivery, as described in scenario 1) of clause 5.2.1, with an inference of a pre-trained AI/ML model in the UE, as described in scenario 2a) of clause 5.2.1. Input media data may be pre-processed as inference input data before being passed to the inference engine. Output media data may be post-processed before being consumed.

In the network:

- An AI/ML model in the repository is selected for the AI/ML media service by the network application and sent to the delivery function for delivery to the UE. Selection of an AI/ML model could depend on UE and network characteristics, such as the memory and CPU capability/availability, as well as current network load and performance status.
- The AI/ML model delivery function sends the AI/ML model data to the UE via the 5GS. This delivery function may also contain functionalities related to QoS requests and monitoring, as well as those related to the optimization or compression of AI/ML model data.

In the UE:

- A UE application provides an AI/ML media service using the AI/ML model inference engine and AI/ML model access function.
- The AI/ML model access function receives the AI/ML model data via the 5G system, and sends it to the AI/ML model inference engine. Receiver side optimization or decompression techniques for AI/ML model data may be included.
- The AI/ML model inference engine performs inference on the received model by using input media data from data sources (e.g. cameras or other media devices). The AI model inference passes inference output data to the UE Data destination for consumption (e.g. for a media player).

Depending on the exact service scenario, AI/ML model updates may be necessary during the service, and different AI/ML model data delivery pipelines may be considered for such purposes. An AI/ML model update may consist of a change in the AI/ML model structure without disrupting the AI/ML media service. If the AI/ML model has requirements on UE memory, processing/computing capabilities or if running the AI/ML model will increase the UE's power consumption dramatically which will also influence the user experience of other services, it may actively request the update of the AI/ML Model. For example, when the memory usage of the UE processing the AI/ML Model exceeds a certain threshold, or if UE performance deteriorates, the UE can actively send a request to the network for an AI/ML Model update. Alternatively, the network may also trigger the AI/ML model update itself, where an interaction between

the UE and network side might be needed to help the network collect current UE status information, e.g. Memory, CPU, current load, terminal location, current power consumption, current battery storage, etc.

5.2.2.2 Basic workflows

5.2.2.2.1 Generic model delivery

Figure 5.2.2-3 shows a basic workflow for AI/ML model delivery with inference in the UE. Steps for the procedures shown are described below.

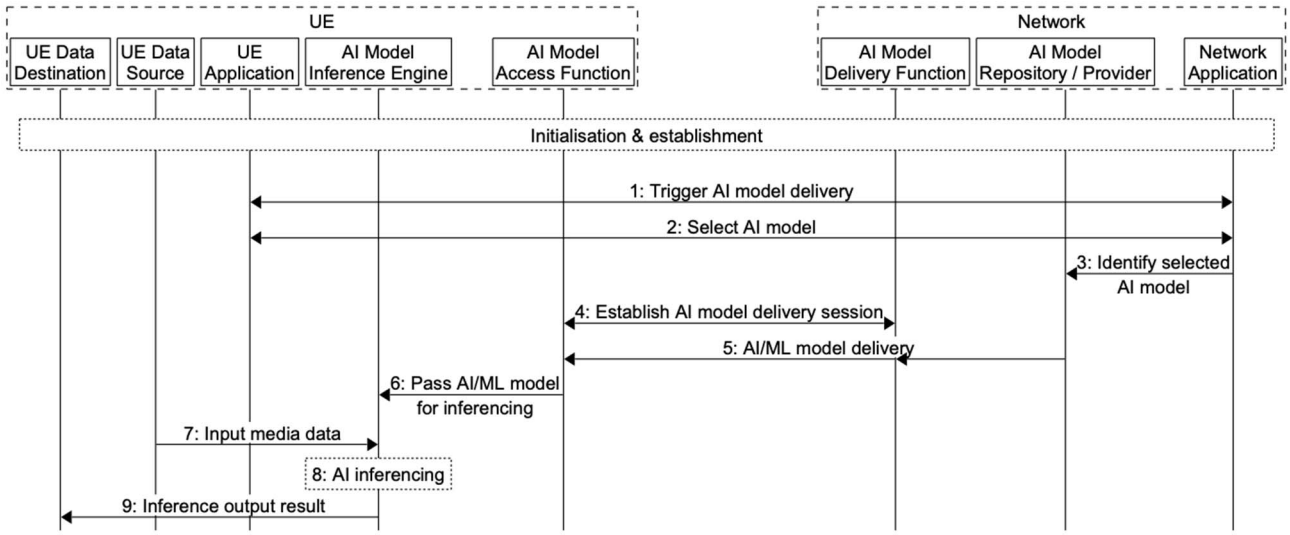


Figure 5.2.2-3: Basic workflow for AI/ML model delivery with inference in the UE

During the initialization and establishment step, it is assumed that information related to the required features and detailed configurations are exchanged and negotiated between the network and UE. Information may include those related to UE device and network capabilities, AI/ML service information (e.g. service requirements, AI/ML model descriptions), and delivery methods. Such information may be used for the selection of a suitable AI/ML model for the service.

1. The UE Application and Network Application communicate to trigger AI/ML model delivery, using the information from the initialization and establishment step.
2. An AI/ML model is selected between the UE Application and Network Application.
3. The Network Application identifies the selected AI/ML model in the AI/ML model Repository/Provider.
4. The AI/ML Model Access Function establishes an AI/ML model delivery session with the AI/ML Model Delivery Function.
5. The AI/ML Model Access Function receives the AI/ML model.
6. The AI/ML Model Access Function passes the AI/ML model to the AI/ML model Inference Engine in the UE.
7. The Data Source passes media data to the AI/ML model Inference Engine.
8. The AI/ML Model inference engine performs AI/ML inferencing.
9. The AI/ML Model inference engine passes the inference output result to the UE Data Destination for consumption.

5.2.2.2.2 Adaptive model delivery

Adaptive model delivery refers to a model delivery paradigm wherein a smaller size but lower precision model is delivered to a UE first to speed up the inference at the UE and to improve QoE. Subsequent model updates are delivered to the UE and the model at the UE is updated to a higher precision. In this context, an adaptive model refers to a model

which can be used for inference as it is by the UE, but subsequent updates can be applied to it to improve its accuracy. The update may be applied in different ways to compose the model, depending on how the low precision model is built. For example, 1) additive composition such as addition of bits for a bit-incremental model (e.g. quantized model 8, 16, 32 bits) or addition of neurons for a pruned model. 2) consecutive composition by appending model data to the previously received model data. (e.g. model with different subsets including early exits).

Figure 5.2.2-4 and text below shows a basic workflow for adaptive model delivery update. Steps for the procedures shown are described below.

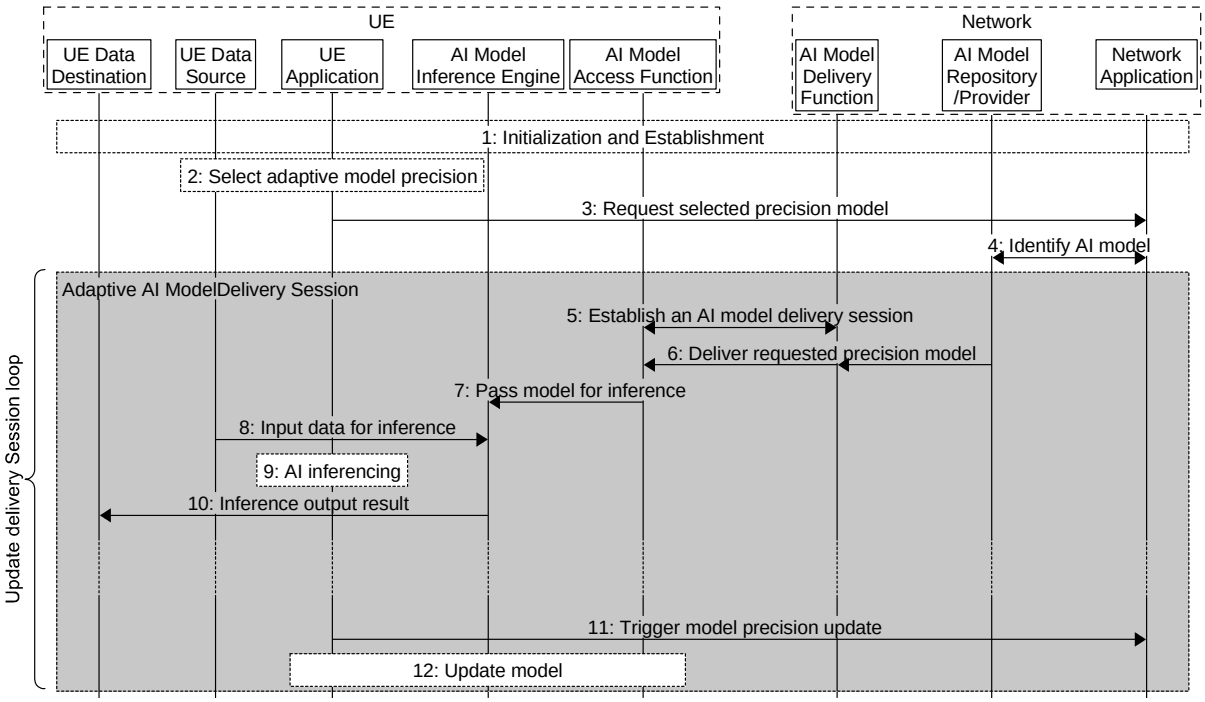


Figure 5.2.2-4: Basic workflow for adaptive model delivery update

1. During the initialization and establishment step, the UE Application and Network Application communicate to establish adaptive model delivery. The UE Application may receive Service Access information to learn about available services and configurations, including available AI/ML models, precisions and possible updates. This information may be in a 3GPP URI of/or model manifest file(s). The model manifest file contains size, complexity information etc. of the different versions. The available model list may comprise full models (as for 5.2.2-3), or adaptive models.

2. An adaptive model is selected by the UE Application, based on, e.g. model size and currently available network capacity.

3. The UE application requests the adaptive model of selected precision from the Network Application

4. The Network Application identifies the selected AI/ML model in the AI/ML model Repository/Provider.

Adaptive AI/ML model delivery session loop

5. The AI/ML Model Access Function establishes an AI/ML model delivery session with the AI/ML Model Delivery Function.

6. The AI/ML Model Access Function receives the AI/ML model of the precision requested by the UE.

7. The AI/ML Model Access Function passes the AI/ML model to the AI/ML model Inference Engine in the UE.

8. The Data Source passes data to the AI/ML model Inference Engine, AI/ML Model Inference Engine performs AI/ML inferencing,

9. The AI/ML Model inference engine performs AI/ML inferencing.

10. AI/ML Model inference engine passes the inference output result to the UE Data Destination for consumption.

Mode delivery update.

11. The UE application triggers a model precision update for updating the AI/ML model to a higher precision.

AI/ML Model delivery session is reused or established according to step 5-10. These steps may be repeated depending upon number of precision levels and corresponding model updates.

12. The update is applied to the low precision model.

The inference loop of step 9 continues.

5.2.3 Split AI/ML operation

5.2.3.1 Basic architectures

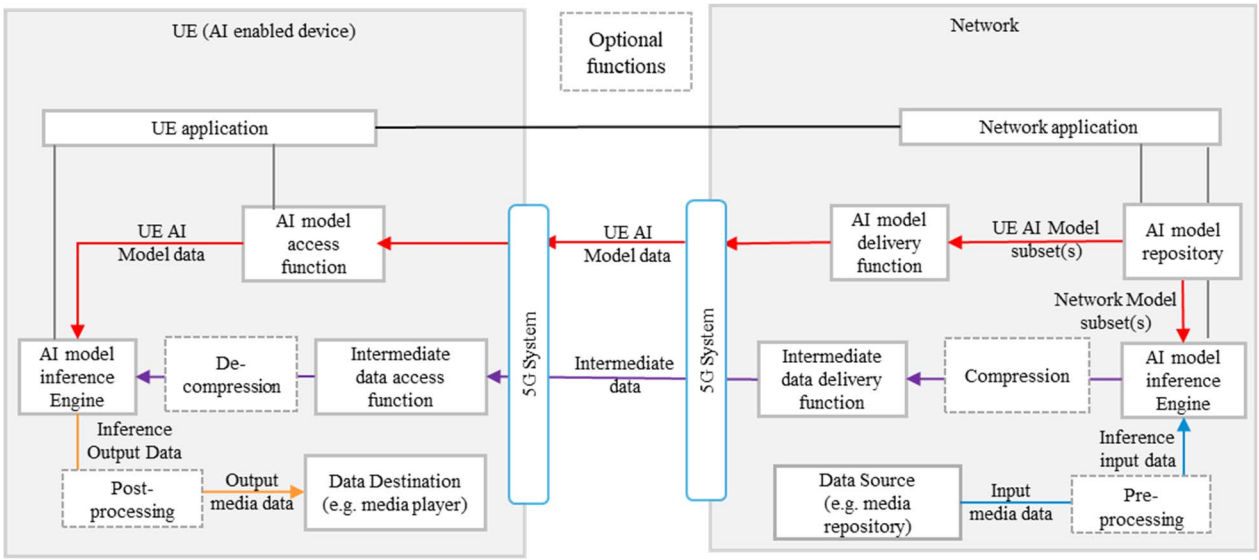


Figure 5.2.3-1: Basic architecture for split inference between the network and UE, with media data source in the network or from the UE via the network

Figure 5.2.3-1 shows a simple basic architecture for split inferences between the network and the UE, as described in scenario 2b) of clause 5.2.1, where the media data source comes from the network, or from the network via the UE. The first part of the AI/ML model is executed on the network side and the second part on the UE. Input media data may be pre-processed as inference input data before being passed to the inference engine. Output media data may be post-processed before being consumed.

For the split inference (network-UE) scenario, additional components are required:

In the network:

- An AI/ML model inference engine that receives both the network AI/ML model subset(s), and input data, for network inference. The input data may originate from the UE via the network. An intermediate data delivery function receives the partial inference output (intermediate data) from the network inference engine and sends it to the UE via the 5GS. This delivery function may also contain functionalities related to QoS requests and monitoring, as well as those related to the optimization or compression of intermediate data.

In the UE:

- An intermediate data access function receives the intermediate data from the network via the 5GS and sends it to the UE inference engine for UE inference. If the intermediate data delivery function performs optimization or compression on intermediate data, this function may apply the corresponding reconstruction or decompression techniques.

- The inference engine passes the inference output data to the data destination (e.g. a media player).

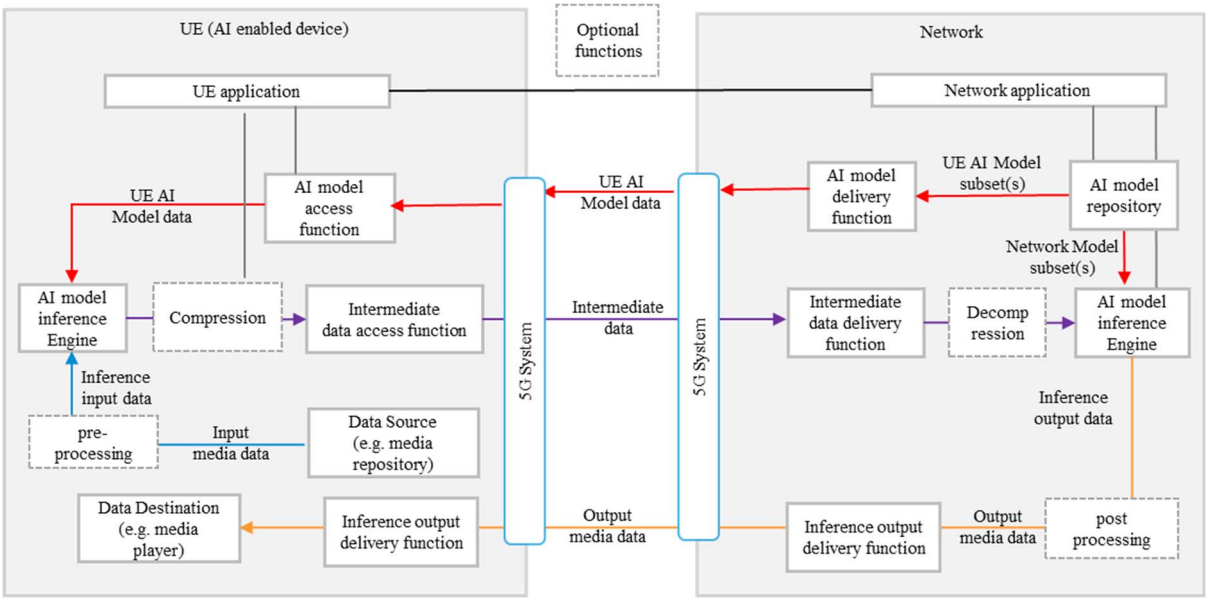


Figure 5.2.3-2: Basic architecture for split inference between the UE and network, with media data source in the UE

Figure 5.2.3-2 shows a basic architecture for split inferences between the UE and the network, as described in scenario 2b) of clause 5.2, where the media data source originates from the UE, the first part of the inference is performed in the UE, the second part in the network. The resulting output data is finally sent back to the UE. Input media data may be pre-processed as inference input data before being passed to the inference engine. Output media data may be post-processed before being consumed.

For the split inference (UE - network) scenario, additional components are required:

In the UE:

- An AI/ML model inference engine that receives both the network AI/ML model subset(s), and input media data (from a UE data source), for UE inference.
- An intermediate data delivery function receives the intermediate data from the UE inference engine and sends it to the network via the 5GS. This delivery function may also contain functionalities related to QoS requests and monitoring. If the intermediate data delivery function performs optimization or compression on intermediate data, this function may apply corresponding optimization or decompression techniques.
- An inference output access function receives the post-processed inference output data from the network via the 5GS and sends it to the relevant data destination according to the AI/ML media service.

In the network:

- An intermediate data access function receives the intermediate data from the UE via the 5GS and sends it to network inference engine for network inference. If the intermediate data delivery function applies optimization or compression on intermediate data, this function may apply corresponding optimization or decompression techniques.
- The inference engine passes the inference output data to the UE via the 5GS, through the inference output delivery function.

For both split inference scenarios, extra factors may be considered, including those such as:

- Configuration of the split inference between the network and UE. (e.g. definition and selection of the AI/ML model composition into “UE AI/ML model subset” and “network AI/ML model subset”)
- Resource allocation and management for network inference, including ingestion of network AI/ML model data and media data

- Intermediate data delivery pipelines between the network and UE, in particular considering the use of 5GMS or RTC defined pipelines to stream intermediate data that is media content data.
- The functionalities of certain components in figure 5.2.1-1 and figure 5.2.2-1 may overlap, and depending on the use case a combined architecture may also be considered for further study.
- Certain components may also overlap with functions defined in 5GMS or RTC architectures, clarifications are for further study.

5.2.3.2 Basic workflows

Figure 5.2.3-3 shows a basic workflow for split inference between the network and UE. Steps for the procedures shown are described below to illustrate the use-cases clause 4.2.

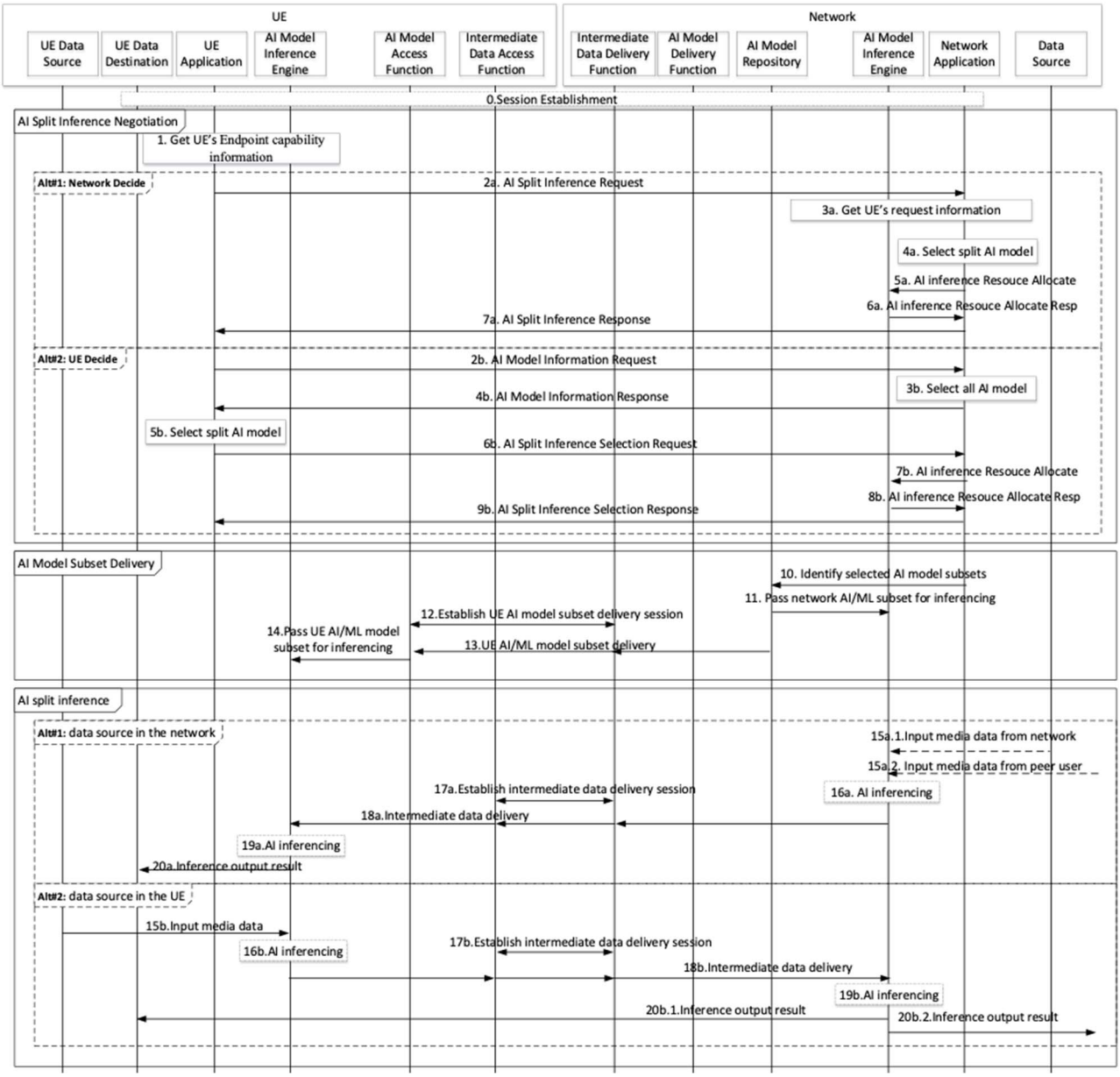


Figure 5.2.3-3: Basic workflow for split inference between the network and UE

0. The session is established between the UE and the network.

AI/ML Split Inference Negotiation (This step may be performed at the beginning or during the session when the UE or network status has changed):

1. The UE Application gets the UE's capability information such as memory and AI/ML inference processing capabilities available for the AI/ML service, supported AI/ML framework information, connection capabilities, etc.

Alternative Case#1: Network decides the split inference

2a. The UE Application sends an AI/ML split inference Request to the Network Application. The request may include AI/ML service information (e.g. service requirements), UE's capability such as available AI/ML inference processing capabilities, supported AI/ML framework for the AI/ML Inference Engine, available bandwidth (uplink/downlink) and preferred delivery method. The UE may also indicate the optimization selection choice regarding latency, energy consumption, network bandwidth to let the Network Application to find the best compromise.

3a. The Network Application gets the UE's request information.

4a The Network Application selects a proper AI/ML model (including the UE AI/ML model subset and the network AI/ML model subset) for split inference from all AI/ML models (with different candidate split points configurations) matching the service requirement information, the UE's capability information and the network's capability information.

5a. The Network Application sends an AI/ML Inference Resource Allocation request to the AI/ML model inference engine with the selected network AI/ML model subset information (including the split point and the intermediate data information).

6a. The AI/ML model inference engine responds with a successful result to the Network Application.

7a. The Network Application sends the AI/ML Split Inference Response with the selected UE AI/ML model subset information (including the split point and the intermediate data information) to the UE Application.

Alternative Case#2: UE decides the split inference

2b. The UE Application sends an AI/ML Model Information Request to the Network Application. The request may include AI/ML service information (e.g. service requirements), UE's capability such as the available AI/ML inference processing capabilities, supported AI/ML framework for the AI/ML Inference Engine, available bandwidth (uplink/downlink) and preferred delivery method.

3b. The Network Application collects all AI/ML models with different candidate split points configurations matching the service requirement information, the UE's capability information and the network's capability information. Candidate split point configuration may also include model accuracy, Intermediate data characteristics, latencies (e.g. UE application, uplink/downlink connections, Network Application), UE profile required to infer the UE subset (Memory or computing resources), distribution mode (e.g. RTP/DASH/Progressive and multicast support).

4b. The Network Application sends the AI/ML Model Information Response with all matched candidates split points configurations to the UE Application. The information response may also include information such as the range/list of supported split points, a preferred split point configuration.

5b. The UE Application selects a split point configuration based on the received information in the AI/ML Model Information Response.

6b. The UE Application sends an AI/ML Split Inference Selection Request to the Network Application with the selected split point configuration.

7b. The Network Application sends an AI/ML Inference Resource Allocation request to the AI/ML model inference engine with the network model subset information corresponding to the AI/ML model selected by the UE Application.

8b. The AI/ML model inference engine responds with a successful result to the Network Application.

9b. The Network Application sends the AI/ML Split Inference Selection Response to the UE Application.

AI/ML Model Subset Delivery:

10. The Network Application identifies the selected UE and network AI/ML model subsets in the AI/ML model Repository.

11. The AI/ML Model inference engine in the network receives the network AI/ML model subset.

12. The AI/ML Model Access Function establishes a UE AI/ML model subset delivery session with the AI/ML Model Delivery Function.

13. The AI/ML Model Access Function receives the UE AI/ML model subset.

14. In the UE, the AI/ML Model Access Function passes the UE AI/ML model subset to the AI/ML model Inference Engine.

AI/ML split inference:

Alternative case#1: data source in the network

15a. The network AI/ML model Inference Engine receives media data from the network Data Source or a peer user.

16a. The network AI/ML model Inference Engine performs network AI/ML inferencing.

17a. The Intermediate Data Access Function establishes an intermediate data delivery session with the Intermediate Data Delivery Function.

18a. In the UE, the Intermediate Data Access Function receives intermediate data and passes it to the AI/ML model inference engine. If the intermediate data delivery function performs optimization or compression on intermediate data, the intermediate data access function may apply corresponding optimization or decompression techniques.

19a. The AI/ML model inference engine in the UE performs AI/ML inferencing.

20a. The AI/ML model inference engine passes the inference output result to the UE Data Destination for consumption.

Alternative case#2: data source in the UE

15b. In the UE, the Data Source passes media data to the AI/ML model Inference Engine.

16b. The UE AI/ML model Inference Engine performs UE AI/ML inferencing.

17b. The Intermediate Data Access Function establishes an intermediate data delivery session with the Intermediate Data Delivery Function.

18b. In the network, the Intermediate Data Access Function receives intermediate data and passes it to the AI/ML model inference engine. If the intermediate data delivery function performs optimization or compression on intermediate data, the intermediate data access function may apply corresponding optimization or decompression techniques.

19b. In the network, the AI/ML model inference engine performs network AI/ML inferencing.

20b. The network AI/ML model inference engine sends the inference output result to the UE Data Destination or a peer user.

5.2.4 Distributed/federated learning

5.2.4.1 Basic architecture

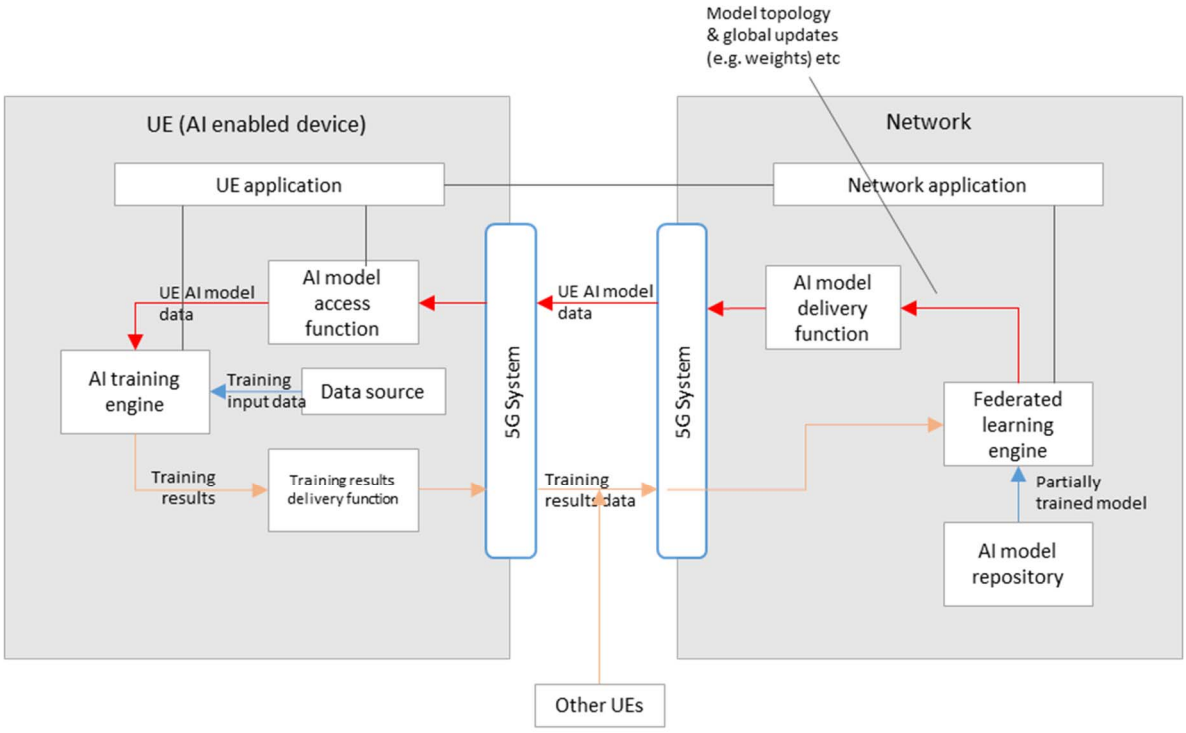


Figure 5.2.4-1: Basic architecture for distributed/federated learning between the network and multiple UEs

Figure 5.2.4-1 shows a basic architecture for distributed/federated learning between the network and UE(s), as described in scenario 3) of clause 5.2.1.

In the network:

- A federated learning engine receives a partially trained model from the AI/ML model repository, that is passed to the AI/ML model delivery function for delivery to multiple UEs via the 5GS.
- Training results data from multiple UEs is also received by the federated learning engine via the 5GS, which is then aggregated for the continuous training of the global model.
- Updates to the global model (e.g. in terms of topology or weights) are delivered to the UEs during the learning process.

In the UE(s):

- AI/ML model data is received by an AI/ML model access function via the 5GS, which then passes the data to the AI/ML training engine.
- An AI/ML training engine in the UE trains the AI/ML model using local device data as the training input.
- Training results (e.g. in the form of updated weights) are delivered to the network via the training results delivery function.

5.2.4.2 Basic workflows

Figure 5.2.4-2 shows a basic workflow for distributed/federated learning with training in the UE, the results of which are aggregated in the network. Steps for the procedures shown are described below.

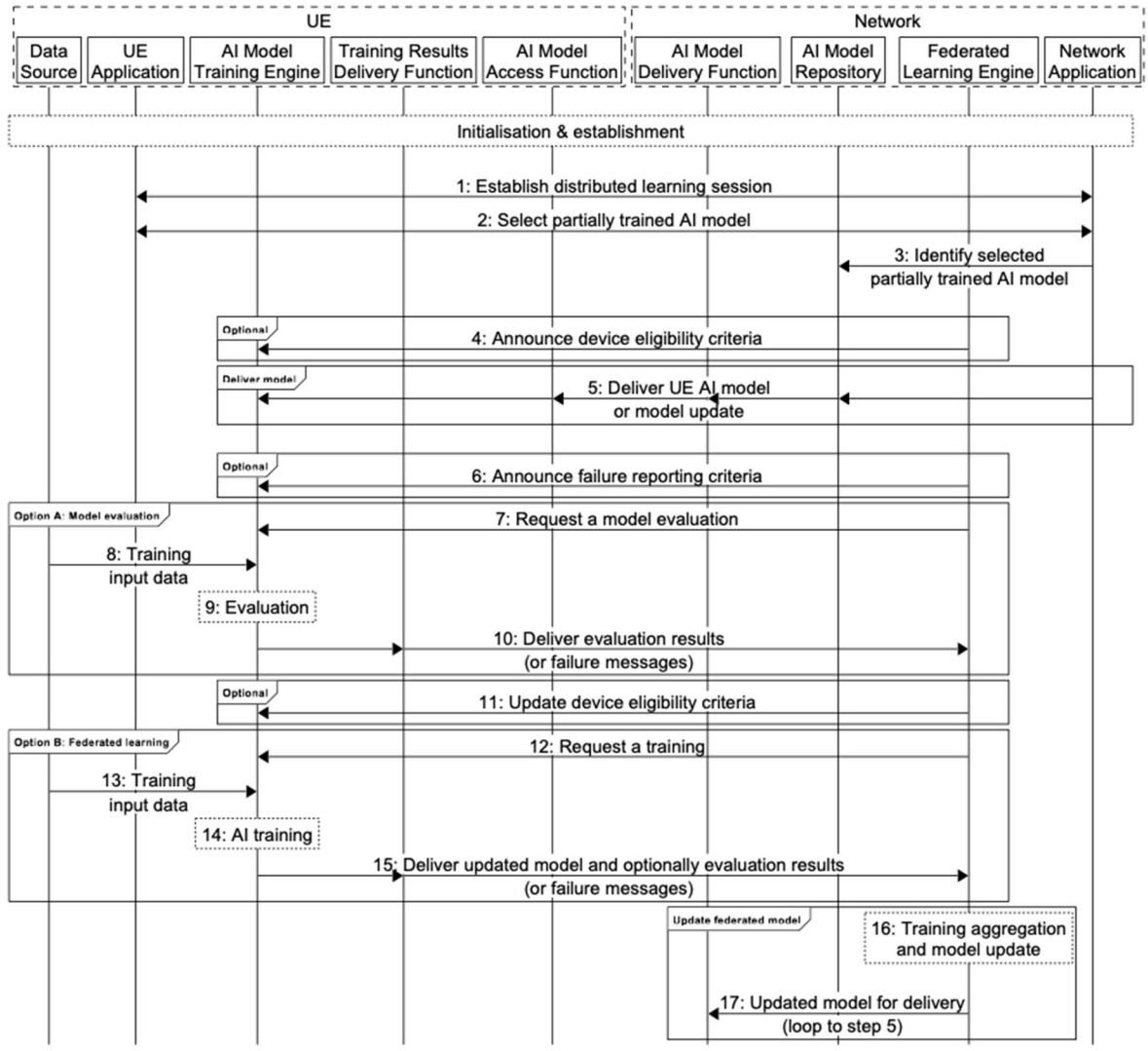


Figure 5.2.4-2: Basic workflow for distributed/federated learning between a UE and the network

During the initialization and establishment step, it is assumed that information related to the required features and detailed configurations are exchanged and negotiated between the network and UE. Information may include those related to UE device and network capabilities, AI/ML service information (e.g. service requirements, AI/ML model descriptions), and delivery methods. Such information may be used for the selection of a suitable partially trained AI/ML model for the service.

1. The UE Application and Network Application communicate to trigger distributed/federated learning, using the information from the initialization and establishment step.

2. A partially trained AI/ML model is selected between the UE Application and Network Application.

3. The Network Application identifies the selected partially trained AI/ML model in the AI/ML model Repository/Provider.

4. The Federated Learning Engine optionally announces the eligibility criteria for participating in the federated evaluation/learning to the device. The criteria could contain various information such as the device's operating system, processor speed, available memory, characteristics of the data library, geographical location of the device, language setting, and other attributes.

5. The AI/ML Model Access Function of an eligible device receives the partially trained AI/ML model or its updated version

6. The Federated Learning Engine optionally announces the failure reporting criteria for the participating devices.

Option A: Model evaluation:

7. The Federated Learning Engine requests the UE to start the model evaluation. The evaluation mechanism and criteria are defined by the Federated learning Engine.

Note: Whether a user wants its device to participate in the evaluation, depends on the business agreement between the user and the network.

8. The Data Source passes the training input data to the AI/ML model Training Engine.

9. The AI/ML Model Training Engine performs the evaluation.

10. The evaluation results (or the failure information, in the case of a failure) are delivered to the Federated Learning Engine.

11. Optionally, the device eligibility criteria may get updated depending on the evaluation results.

Option B: Federated training:

12. The Federated Learning Engine requests the UE to start the training.

Note: Whether a user wants its device to participate in the training, depends on the business agreement between the user and the network.

13. The Data Source passes the training input data to the AI/ML model Training Engine.

14. The AI/ML Model Training Engine performs the retraining of the model.

15. The updated model (or the failure information, in the case of a failure) is delivered to the Federated Learning Engine.

16. The Federated Learning Engine performs training aggregation of training results from multiple UEs and updates the partially trained AI/ML model.

17. The updated partially trained AI/ML model is delivered to the UE as from step 5.

Note: As shown in the above call flow, the model evaluation and the federated learning may also occur in a sequence.

5.3 Architecture for AI/ML data delivery

5.3.1 AI/ML data components

AI/ML user plane data includes:

- AI/ML model data (see clauses 3.1 and 6.2), including data describing the topology/structure of the AI/ML model, data related to the data nodes of the model, i.e. tensors, and other data which may be dependent on the format used for the AI/ML model.

- Intermediate data (see clauses 3.1 and 6.3), defined as the output data from the first step of a split inference process of an AI/ML model that is not considered the final inference result (depending on the service and output layer of the split AI/ML model, certain intermediate data may have media characteristics, or even be media data). Intermediate data is typically required to be delivered to a second device or entity, as the input to a subsequent second split inference process.

- Inference output data (see clause 3.1), which is the data corresponding to the output result of the final AI/ML inference process for the service. Depending on the nature of the AI/ML data inferencing for the given AI/ML data service, this inference output data may include: labels for identifying recognition like tasks from media, actual media data such as video and/or audio, or perhaps XR related data such as 3D models.

- Inference input data (see clause 3.1), corresponds to all inputs feeding the AI/ML inference. In case of a split inference, input data feeds the first inference starting the inference at the input of the trained model. For AI/ML for media use-cases, input is media data (image, video, audio, etc.)

- Training input data, corresponds to all inputs feeding the AI/ML training process on a device for federated learning. Such data is typically created on, or exists in UE devices.

- Training results data, which is the data corresponding to the output result of the AI/ML training process. Such data is typically delivered by a UE device to a federated learning entity which aggregates data from multiple UE devices to update and train a model.

- User-plane metadata (see clause 6.8), corresponds to contextual and additional information to the data payload being transmitted.

5.3.2 Media-related AI/ML data logical functions

The identified User plane logical functions supporting the scenarios include:

- AI/ML data delivery function

- AI/ML data access function

- AI/ML model inference engine

- AI/ML training engine

- AI/ML federated learning engine

Control plane functions in both the UE and network are needed for configuration, capability exchange and reporting:

- AI/ML capability manager

5.3.3 Mapping AI/ML functions to the generalized 5G media delivery architecture

Using the architecture in 3GPP TS 26.501 [11] as a reference architecture, instead of defining new 5G AI/ML functions at the same level as the generalized functions, it is also possible to directly map specific logical AI/ML functions into the generalized functions to support AI/ML media services as shown in the table 5.3-1 below:

Table 5.3-1: Logical AI/ML functions

Generalized media architecture function	Logical AI/ML function
Media AF	AI/ML Capability Manager
Media AS	AI/ML Data Access/Delivery, AI/ML Inference Engine Federated Learning Engine
Media Client	~
Media Session Handler	AI/ML Capability Manager
Media Access Function	AI/ML Data Access/Delivery, AI/ML Inference Engine, AI/ML Training Engine
Media Application Provider	AI/ML Enabled Application Provider
Media-aware Application	AI-aware Application

5.3.4 Architecture and components for AI/ML data delivery over 5G

5.3.4.1 Introduction

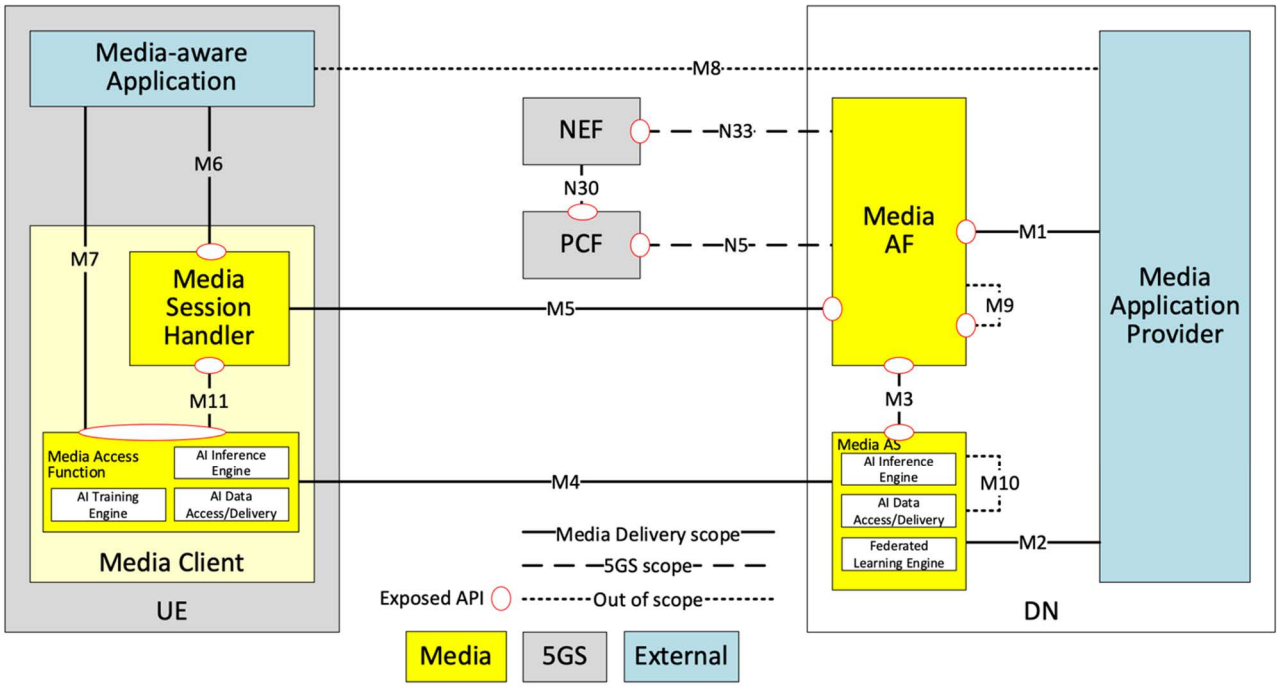


Figure 5.3.4-1: AI/ML data delivery general architecture

A generalized 5G Media Delivery architecture supporting AI/ML media functionality is shown in figure 5.3.4-1. Depending on the service scenario and/or use case, certain dedicated AI/ML logical subfunctions may be mapped to, or instantiated by the generalized media architecture functions.

5.3.4.2 Network functions and UE entities

In addition to the media related definitions described in TS 26.501 [11], additional definitions for AI/ML data related functions include:

- **Media Client** running on the UE contains two subfunctions:
  - **Media Session Handler**: A function on the UE that communicates with the network side 5G AI/ML Application Function (AF) to establish and control the configuration of an AI/ML data session. The function may include:
    - Features that monitors, shares and/or reports UE capabilities with/to Media AF. This may be used for the selection of the model for a UE inference or for the selection of the subset of a UE model for a split inference topology between the UE and the network. It may also be used for supporting federated learning data exchange.
  - **Media Access Function**: A function on the UE that communicates with the Media AS and the Media Session Handler to establish an AI/ML data delivery session. The function contains:
    - An AI/ML inference engine, which has the capability to perform the inferencing of received (split) AI/ML models.
    - An AI/ML data access and delivery function, which handles the access and delivery of user plane AI/ML data, as well as conventional media data including
      - Download the AI/ML model data for inference process, or the partially trained AI/ML model for on device local training. This includes instantiating an AI/ML data access client to access and retrieve AI/ML models or AI/ML model subsets from local files or over the network (e.g., by streaming or

downloading the model from a remote server). The inference engine may comprise format decapsulation and model decoding functions as well as a runtime engine that executes the model from the memory.

- Access/deliver intermediate data when a inference is split between the UE and the network.

- Encode data to deliver with serialization and/or compression technique or conversely decode the received AI/ML data with deserialization or decompression technique.

- An AI/ML training engine, which has the capability to perform training of a partially trained model, using training data as the input

- **Media-Aware Application**: An external function controlled by the external 5G AI/ML application provider implementing the AI/ML application logic, which includes triggering the delivery of an AI/ML model to be used by the inference engine and obtaining inference results from the inference engine.

- **Media AS (Application Server)**: An Application Server that hosts 5G AI/ML data functions. It includes:

  - An *AI/ML data access and delivery function*, which handles the access and delivery of user plane AI/ML data, as well as conventional media data.

  - An *AI/ML inference engine*, which has the capability to perform the inferencing of (split) AI/ML models.

  - A *federated learning engine*, which aggregates training results data and updates the model to be trained accordingly.

- **Media AF (Application Function)**: An Application Function that provides various control and configuration functions to the Media Session Handler on the UE and/or to the Media Application Provider. It may relay or initiate a request for different Policy or Charging Function (PCF) treatment or interact with other network functions via the NEF (Network Exposure Function). The Application function can include for example:

  - Supporting features such as monitoring, sharing and/or reporting network capabilities to the Media Session Handler, as well as configuring content consumption measurement, logging, collection and reporting; configuring QoE metrics measurement, logging, collection and reporting; requesting different policy and charging treatments; or Media AF-based Network Assistance. This may be used for the selection of the model for a UE inference or for the selection of the UE model subset part for a split inference topology between the UE and the network via the Media Access Function. These features may also be used for the configuration of federated learning.

5.3.5 Procedure for Split AI/ML operation

Figure 5.3.5-1 shows a procedure for split AI/ML operation, including three main parts:

- AI/ML split inference management, and

- AI/ML data delivery session

- Split inference processing

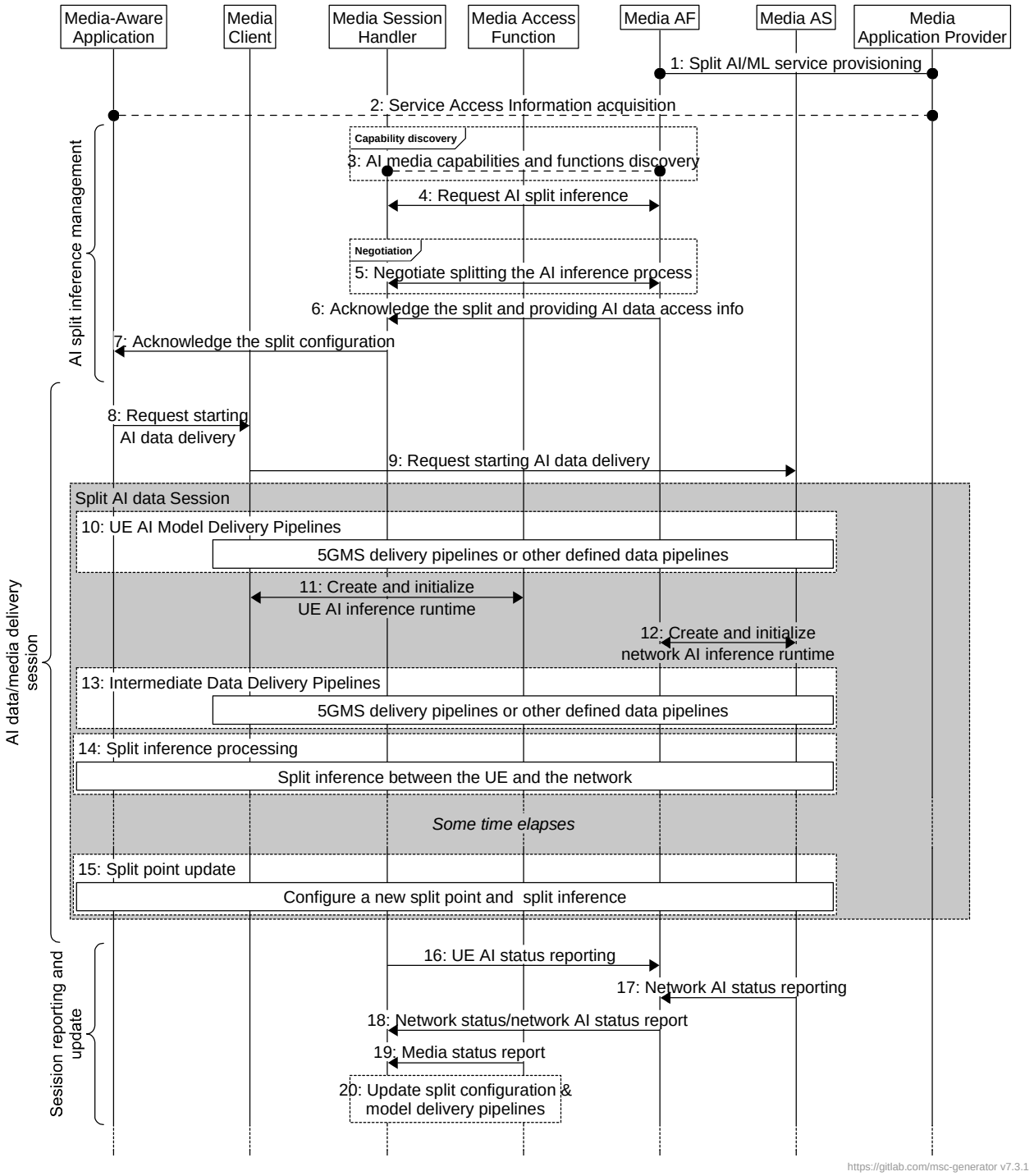


Figure 5.3.5-1: Procedures for split AI/ML operation

Editor's note: need to update the above figure to change AI into AI/ML

1. Service provisioning and announcement of AI/ML data service on the network side, in particular between the Media AF (application function) and the Media application provider.

2. Service access information acquisition. During this step, the available or required AI/ML model(s) for the service can be made known to the UE, by means of information made available via a URL link pointing to a file or manifest which may list such available AI/ML models. Such additional information may contain a list of features available from each AI/ML model, including its variants, including specific information such as the inferencing accuracy, the size, the number of nodes, structure, complexity and latency requirements of the AI/ML model.

AI/ML split inference management:

3. Discovering AI/ML data inferencing capabilities and functions in both the UE and network. In this step, the AI/ML capability manager functions in the UE and in the network may use its capabilities to calculate the range of inference latencies for the AI/ML model to be used for the split AI/ML inference service

4. Requesting AI/ML split inference. Either the UE or the network requests the other endpoint for an AI/ML split inference service. If information describing the AI/ML model was not made known via the service access information in step 2, then such information may also be shared during this step.

5. Negotiate splitting the AI/ML inference process. A split point is negotiated between the UE and the network, using information from steps 2, 3 and 4, in order to satisfy the service, capability and AI/ML model inference latency requirements. The decision of whether the split point is static or whether it can be updated dynamically during the service may be negotiated. Related metadata (see clause 6.6) may be shared between the network and UE depending on the configuration and a set of split points can be negotiated.

NOTE: The negotiation may include inference input data and/or inference output data as defined in clause 5.3.1 to be transmitted from one endpoint to another. For example, input media data captured by a first endpoint can be inference input data for the second part of the model. In another example, part of inference output data may be produced from the first or second part of the model. Not all scenario and split points would be suitable depending on various conditions and this needs to be considered when selecting the split points.

6. Acknowledge the split and provide the AI/ML data split inferencing access info. In this step, the network (Media AF) and UE (AI/ML data session handler) both acknowledge the decided split point, and access information for the AI/ML data is provided to the UE.

7. The split configuration outcome is notified to the Media-aware application.

Split AI/ML data session

8. Request the start of intermediate data delivery. On confirmation, the Media aware Application triggers the Media Client to request the start of AI/ML data delivery using the AI/ML intermediate data access information provided in step 7.

9. The Media client request the intermediate data to be delivered from the Media AS.

10. Pipelines for the delivery of AI/ML model data from the Media AS to the Media Client are setup, and suitable delivery sessions are established and initiated. Delivery may be in the manner of streaming delivery, or download delivery (such as that defined in TS 26.501 [11], or any other form of delivery mechanism required by the AI/ML data service.

11. Start inference process in the UE. In this step, the Media client triggers the inference process (the AI/ML inference engine function), namely the UE side of the split inferencing as decided by the result of step 5.

12. Start inference process in the server. In this step, the Media AF triggers the inference process in the 5GAI/ML AS (the AI/ML inference engine function), namely the network side of the split inferencing as decided by the result of step 5.

13. Pipelines for the delivery of intermediate data from the Media AS to the Media Client are setup, and suitable delivery sessions are established and initiated. Delivery may be in the manner of streaming delivery, such as that defined in TS 26.501 [11], or any other form of delivery mechanism required by the AI/ML data service.

Split inference processing

14. The split inference runs between the UE and the network. Depending on the specific split inference scenario, the UE and the network may deliver and/or access Intermediate data, Inference output data and/or metadata (see clause 6.8.1) using the pipelines defined in the AI/ML data delivery session. If the intermediate data delivery function performs optimization or compression on intermediate data, the intermediate data access function may apply corresponding optimization or decompression techniques.

NOTE: The UE and the network may transmit inference input data with intermediate data and/or inference output data depending on the negotiated split point configuration step 5.

Split point update and inference processing

15. A split point update is triggered, for example from the media aware application to adapt to the new conditions (e.g. UE capabilities or network capacity has changed). The new split point metadata information is either negotiated between the UE and the network or pass alongside the delivery pipeline from the UE to the network side.

Session reporting and update

16. The Media Session Handler may collect and send status reports regarding the UE's AI/ML media service status (for example AI/ML inference status, latency, resource status, capability status, dynamic media properties etc.) to the 5GAI/ML AF.

17. The Media AS may send status reports regarding the network's AI/ML media service status to the Media AF.

18. The Media Session Handler may receive network status, or network AI/ML status reports from the Media AF, as collected in step 16.

19. The Media Session Handler may receive media status reports either from the network or internally from the UE.

20. Depending on the configurations negotiated in step 5, as well as related information from the status reports in steps 16, 17 and 18, updates of the AI/ML model selection, split point configuration or the AI/ML data delivery pipelines for the session may take place between the UE and network.

5.3.6 Procedure for AI/ML model distribution and operation

Figure 5.3.6-1 shows a procedure for AI/ML model distribution and operation.

Similar to the procedures for 5GMS downlink Media Streaming, assuming that the network operator provides such an AI/ML model distribution service, as well as the availability of AI/ML models from the Media Application Provider, the procedure consists of an ingest session (where AI/ML models are uploaded to the Media AS), and a provisioning session, during which the Media Client can access the AI/ML models and the Media Application Provider can control and monitor the AI/ML models and its delivery.

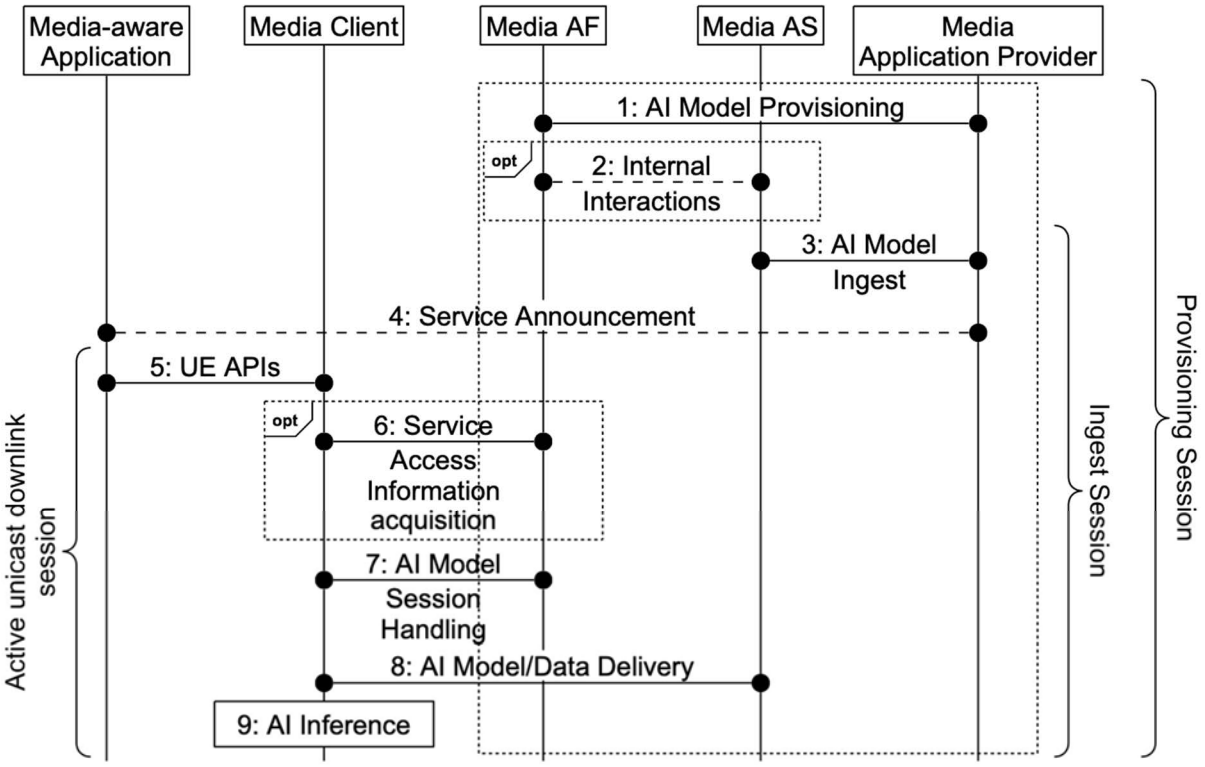


Figure 5.3.6-1: Procedure for AI/ML model distribution and operation

Steps 1 to 8 assume the same steps as defined in TS 26.501 [11] for downlink media streaming, but for AI/ML distribution; the Service Announcement Information (whether acquired in whole in step 4 or through a reference and later in whole in step 6) contains information allowing the Media Client to activate the reception of one or more AI/ML

models. In step 9, the Media Client performs AI/ML inferencing using media as an input into the AI/ML model delivered and received in step 8.

Depending on the distribution use case, media delivery features (such as dynamic policy, network assistance, metrics reporting etc.) may also be applicable to AI/ML distribution.

5.3.7 Procedure for distributed/federated learning

Figure 5.3.7-1 shows a procedure for distributed/federated learning.

Assuming that the network operator provides a distributed/federated learning service using multiple UEs for training, as well as the availability of partially trained AI/ML models from the Media Application Provider, the procedure consists of an ingest session (where partially trained AI/ML models are uploaded to the Media AS), a provisioning session, and an active FL (federated learning) training session, during which the Media Client can access the partially trained AI/ML model for local AI/ML training, after which the training results and report are sent back to the Media AS for training aggregation.

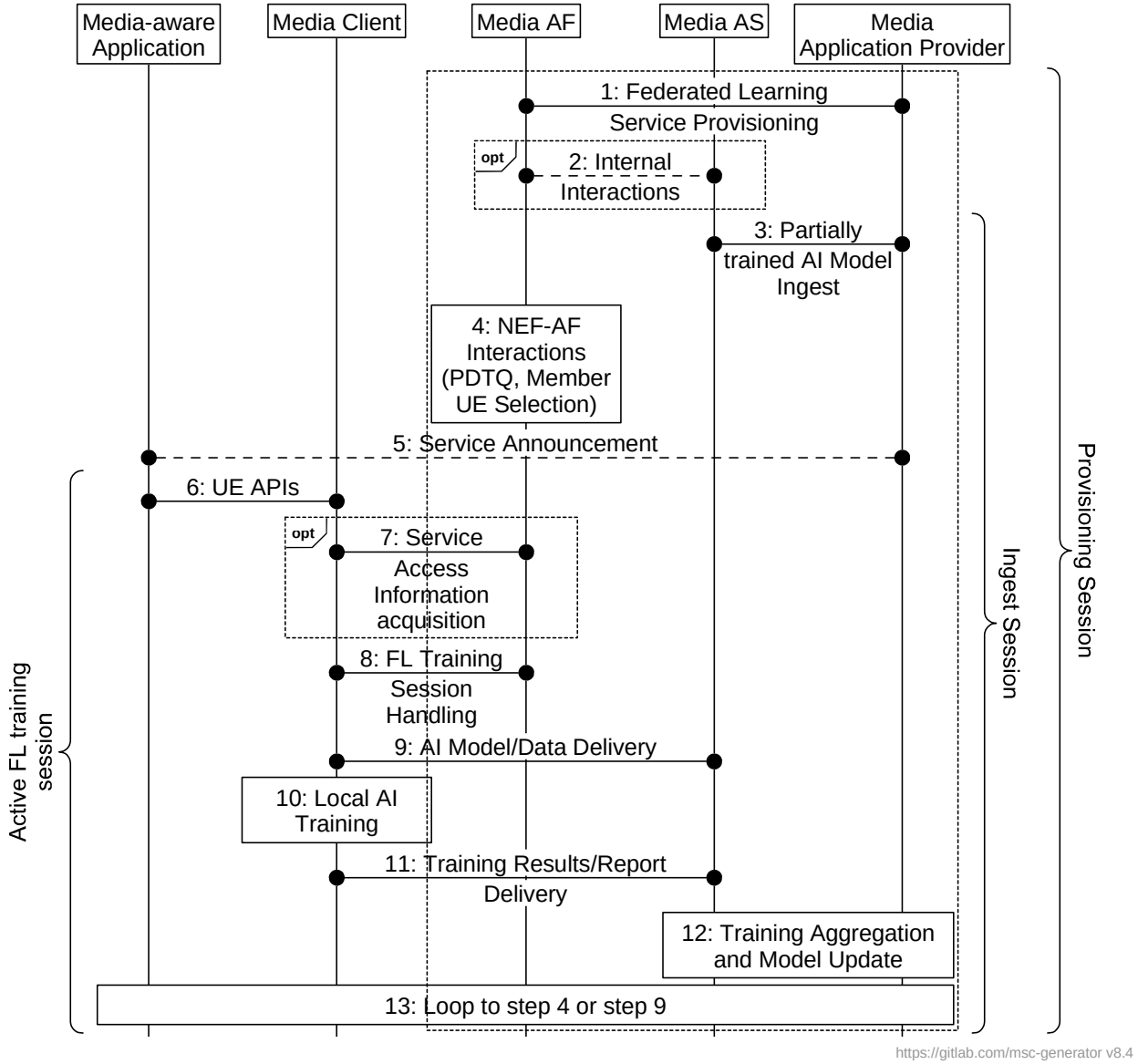


Figure 5.3.7-1: Procedure for distributed/federated learning

1. Service provisioning and announcement of federated learning service on the network side, in particular between the Media AF (application function) and the Media application provider.

2. When content hosting is offered and selected there may be interactions between the Media AF and the Media AS e.g., to configure Server Certificates and/or Content Preparation Templates and to allocate content ingest and distribution resources by providing a Content Hosting Configuration. The Media AS provides resource identifiers for the allocated resources to the Media AF, which then provides the information to the Media Application Provider. For AI/MLs services, content hosting is equivalent to AI/ML model hosting.

3. The Media Application Provider starts the Ingest Session by ingesting the partially trained AI/ML model.

4. NEF - Media AF interactions for the negotiation of assistance to AI/ML operations features as defined in 3GPP TS 22.501 [12]. An example of one feature is Member UE Selection Assistance, where the Media AF may be notified about changes in the subset list of UE(s) that fulfil certain filtering criteria. Another feature is that related to QoS, where the Media AF may request the network to provide a recommended time window for the active FL training session using the Planned Data Transfer with QoS (PDTQ) requirements and procedures.

5. The Media Application Provider provides the Service Announcement Information to the Media-Aware Application. The service announcement includes either the whole Service Access Information (i.e. details for AI/ML Model Session Handling or a reference to the Service Access Information or pre-configured information. When only a reference is included, the Media Client fetches (in step 7) the Services Access Information when needed.

6. When the Media-Aware Application decides to begin the federated learning, the Service Access Information (all or a reference) is provided to the Media Client.

7. (Optional) In case the Media Client received only a reference to the Service Access Information, then it acquires the Service Access Information from the Media AF.

8. The Media Client uses the Media Session Handling API exposed by the Media AF (at M5) for federated learning training session handling, in particular for dynamic policy invocation. The Media Session Handling API is used for configuring content consumption measurement, logging, collection and reporting; configuring QoE metrics measurement, logging, collection and reporting; requesting different policy and charging treatments; or Media AF-based Network Assistance.

9. The Media Client receives the partially trained AI/ML model/data from the Media AS.

10. The Media Client performs local training of the AI/ML model.

11. The Media Client sends the training results/report to the Media AS.

12. The Media AS aggregates training results data from multiple UEs and updates the partially trained model. The partially trained model may also be egested to the Media Application Provider.

13. Further iterations of FL training sessions for the same UE and base AI/ML model may occur from step 4 or step 9, depending on the service configuration.

5.4 Possible architecture and procedures for AI/ML data delivery over IMS

5.4.1 Architecture and components

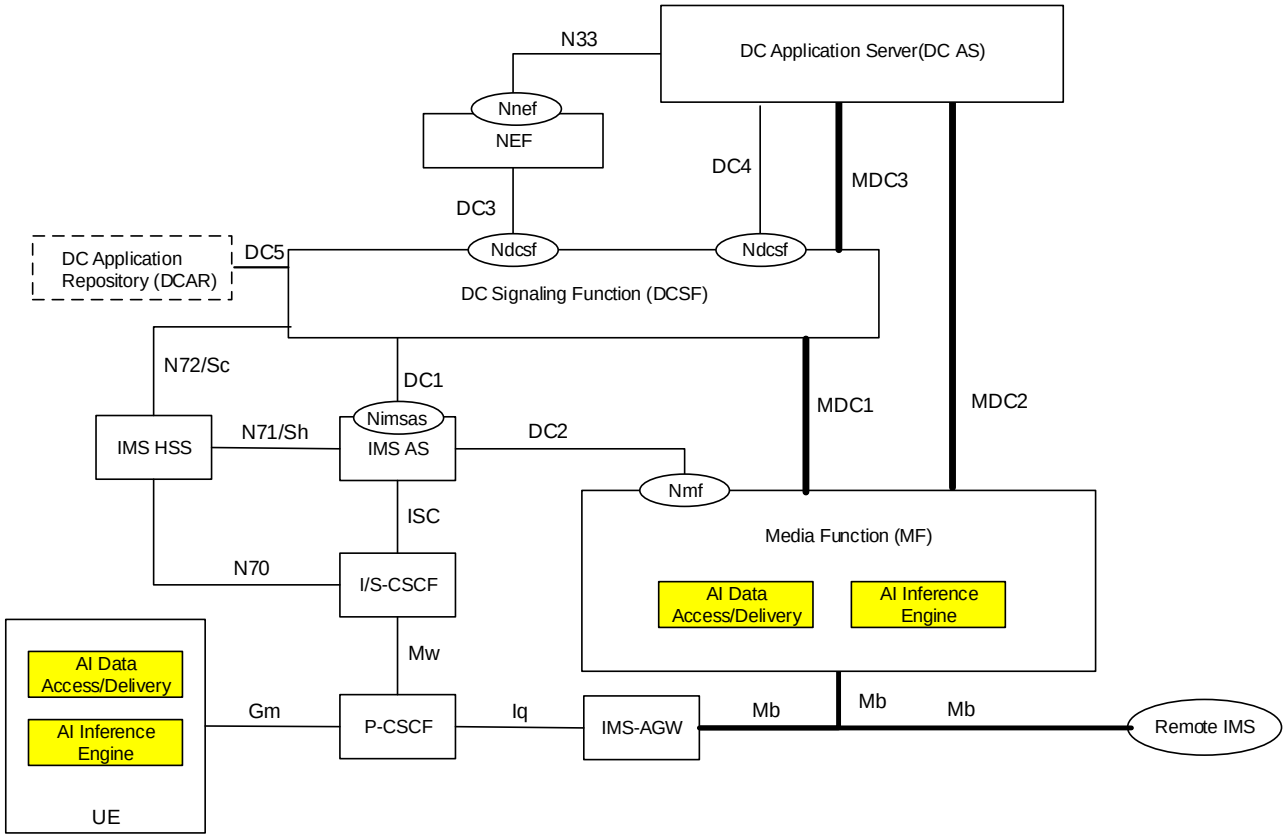


Figure 5.4.1-1 AI/ML data delivery over IMS architecture

Figure 5.4.1-1 shows a mapping of AI/ML media functionalities to the IMS data channel architecture which is defined in clause AC.2 of 3GPP TS 23.228 [13]. The mapped AI/ML media functionalities over IMS are the following:

UE:

- AI/ML Inference Engine: It has the capability to perform the inferencing of received (split) AI/ML models.
- AI/ML Data Access/Delivery: It handles the access and delivery of AI/ML data including
  - Download the AI/ML model data for inference process. This includes instantiating an AI/ML data access client to access and retrieve AI/ML models or AI/ML model subsets over the network (e.g., by streaming or downloading the model from a remote server).
  - Access/deliver intermediate data when an inference is split between the UE and the network.
  - Encode data to deliver with serialization and optionally compression techniques. Or conversely decode the received data with deserialization or optionally decompression techniques.

MF:

- AI/ML Data Access/Delivery: It handles the access and delivery of AI/ML data as described above.
- AI/ML Media Inference Engine: It has the capability to perform the inferencing of (split) AI/ML models.

DC AS:

- It monitors, shares and/or reports network capabilities with/to the UE. This may be used for the selection of the model for a UE inference or for the selection of the UE model subset part for a split inference topology between the UE and the network. It may also provide AI/ML model (including subset) storage and downloading function.

5.4.2 Procedures for AI/ML model distribution

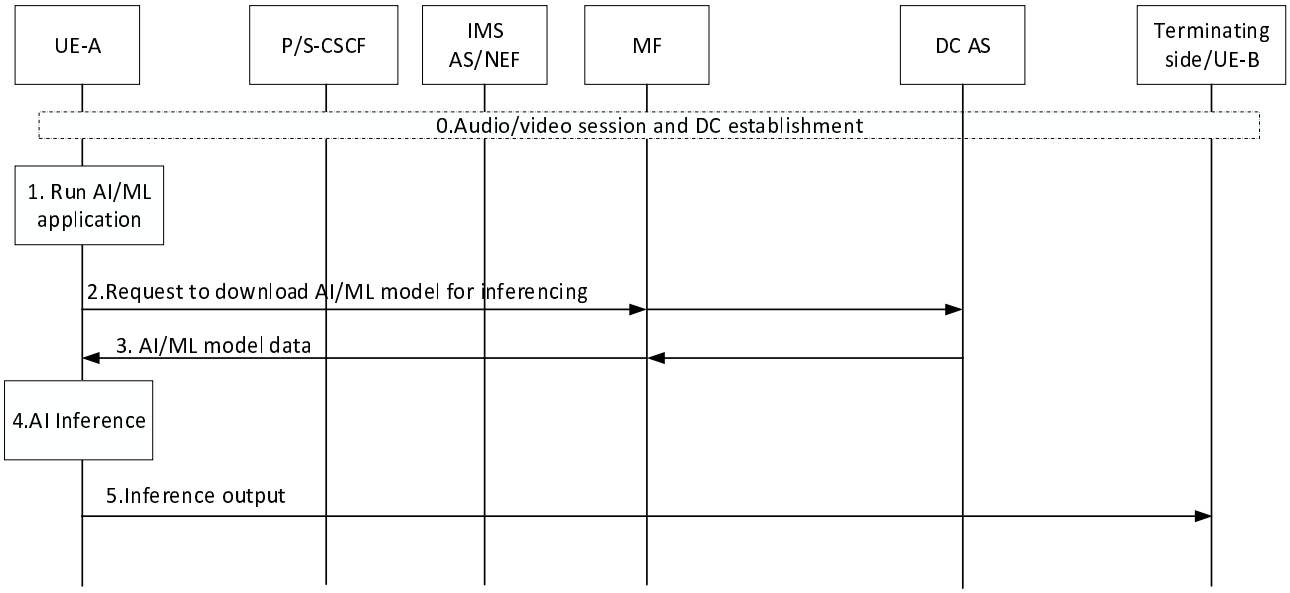


Figure 5.4.2-1 Procedures for AI/ML model distribution

0. The audio/video and data channel sessions are established between the UE-A and the UE-B.
1. UE-A runs AI/ML application.
2. The application requires AI/ML model for inference, the UE-A sends a request for downloading AI/ML model to MF through application data channel, the MF forward the request to DC AS.
3. The DC AS responses with AI/ML model data.
4. The UE-A performs AI/ML inferencing.
5. The UE-A may send the inference output result to the UE-B through application data channel or RTP.

5.4.3 Procedures for Split AI/ML operation

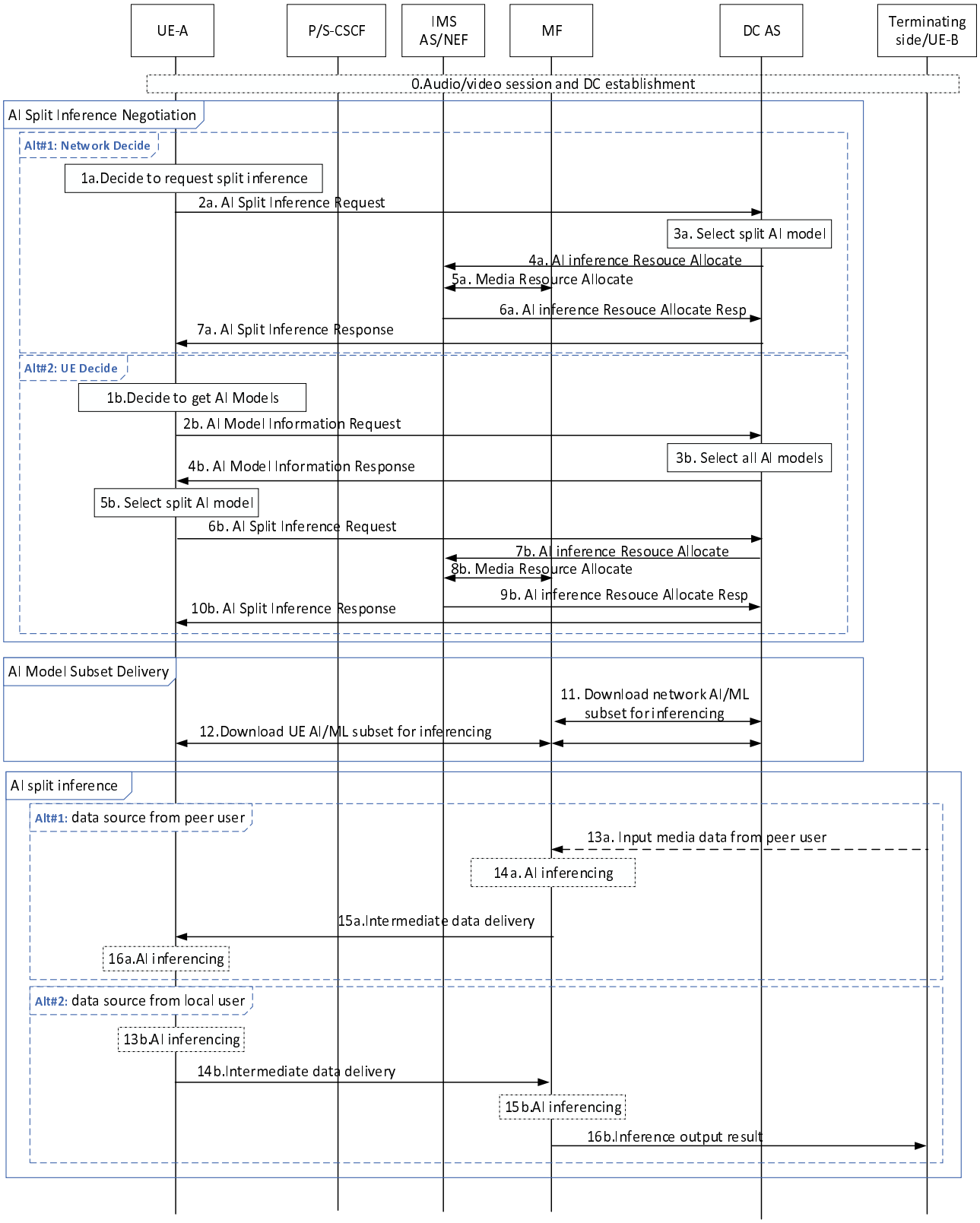


Figure 5.4.3-1 Procedures for split AI/ML operation

0. The audio/video and data channel sessions are established between the UE-A and the UE-B.

AI/ML Split Inference Negotiation (This step may be performed at the beginning or during the session when the UE or network status has changed):

Alternative Case#1: Network decides the split inference:

1a. The UE-A gets the UE's capability information, which may include the AI/ML inference processing capabilities, supported AI/ML framework information, connection capabilities, etc. When the UE-A discovers the UE's local capabilities can't meet the AI/ML service requirement, it decides to trigger the split inference process.

2a. The UE-A sends an AI/ML split inference request to the DC AS with UE's capability information and the service requirement information.

3a. The DC AS gets the MF's capability information, which includes the AI/ML inference processing capabilities, supported AI/ML framework information from the AI/ML Inference Engine, and selects a proper AI/ML model (including the UE AI/ML model subset and the network AI/ML model subset) for split inference from all matched AI/ML models (with different candidate split points) based on the service requirement information, the UE's capability information and the network's capability information.

4a. The DC AS sends an AI/ML Inference Resource Allocation request to the NEF, the request includes the selected network AI/ML model subset information (including the split point and the intermediate data information). The request is transferred to the IMS AS.

5a. The IMS AS instructs a media resource allocation request to the MF.

6a. The MF responds with a successful result to the IMS AS, the IMS AS transfers the response to the NEF, and the NEF transfers it to the DC AS.

7a. The DC AS sends the AI/ML Split Inference Response with the selected UE AI/ML model subset information (including the split point and the intermediate data information) to the UE-A.

Alternative Case#2: UE decides the split inference:

1b. The UE-A gets the UE's capability information, which may include the AI/ML inference processing capabilities, supported AI/ML framework information, connection capabilities, etc. When the UE-A discovers the UE's local capabilities can't meet the AI/ML service requirement, it decides to get the AI/ML models from the network for split inference process.

2b. The UE-A sends an AI/ML Model Information Request to the DC AS with the UE's capability information and the service requirement information.

3b. The DC AS collects all matched AI/ML models with different candidate split points based on the service requirement information, the UE's capability information and the network's capability information.

4b. The DC AS sends the AI/ML Model Information Response with all matched UE AI/ML model subset(s) information (including the split point and the intermediate data information) to the UE-A.

5b. The UE-A selects a proper AI/ML model based on the UE's capability information and the received information in the AI/ML Model Information Response.

6b. The UE-A sends an AI/ML Split Inference Request to the DC AS with the selected network AI/ML model information.

7b. The DC AS sends an AI/ML Inference Resource Allocation request to the NEF, the request includes the selected network AI/ML model subset information (including the split point and the intermediate data information). The request is transferred to the IMS AS.

8b. The IMS AS instructs a media resource allocation request to the MF.

9b. The MF responds with a successful result to the IMS AS, the IMS AS transfers the response to the NEF, and the DCSF transfers it to the DC AS.

10b. The DC AS sends the AI/ML Split Inference Response to the UE-A.

AI/ML Model Subset Delivery:

11. The MF downloads the network AI/ML model subset from the DC AS.

12. The UE-A downloads UE AI/ML model subset from the DC AS over the established data channel.

AI/ML split inference:

Input media data may be pre-processed as inference input data before being passed to the inference engine. Output media data may be post-processed before being consumed.

Alternative case#1: data source from peer user

13a. The MF receives media data from the peer user.

14a. The MF performs network AI/ML inferencing.

15a. The MF deliveries the intermediate data to the UE-A. If the intermediate data delivery function performs optimization or compression on intermediate data, the intermediate data access function may apply corresponding optimization or decompression techniques.

16a. The UE-A performs AI/ML inferencing and output the inference result.

Alternative case#2: data source from local user

13b. The UE-A performs UE AI/ML inferencing based on the media data generated locally.

14b. The UE-A deliveries the intermediate data and passes it to the MF. If the intermediate data delivery function performs optimization or compression on intermediate data, the intermediate data access function may apply corresponding optimization or decompression techniques.

15b. The MF performs network AI/ML inferencing.

16b. The MF sends the inference output data to the UE-B.

NOTE : AI/ML split inference and the negotiation may also happen between the peer UE and the MF. The flow is similar and not repeated here.

5.5 Possible Mapping to IMS using DC Applications

5.5.1 Background

The IMS data channel feature was introduced in Rel-17 to support the enhancement of multimedia telephony with more advanced application logic through web applications. A bootstrap data channel is established as part of the multimedia telephony session, then a list of available web applications is downloaded and offered to the user. Once a user has made a selection, the selected web application is downloaded locally. The remote UE is informed, through a re-INVITE, about the selected web application, so that both endpoints are using the same web application. An application data channel is then established between the endpoints to exchange application specific data. The UE uses the HTTP protocol for the communication over the bootstrap channel.

The architecture extensions to IMS to support data channels is shown in the following figure 5.5.1-1:

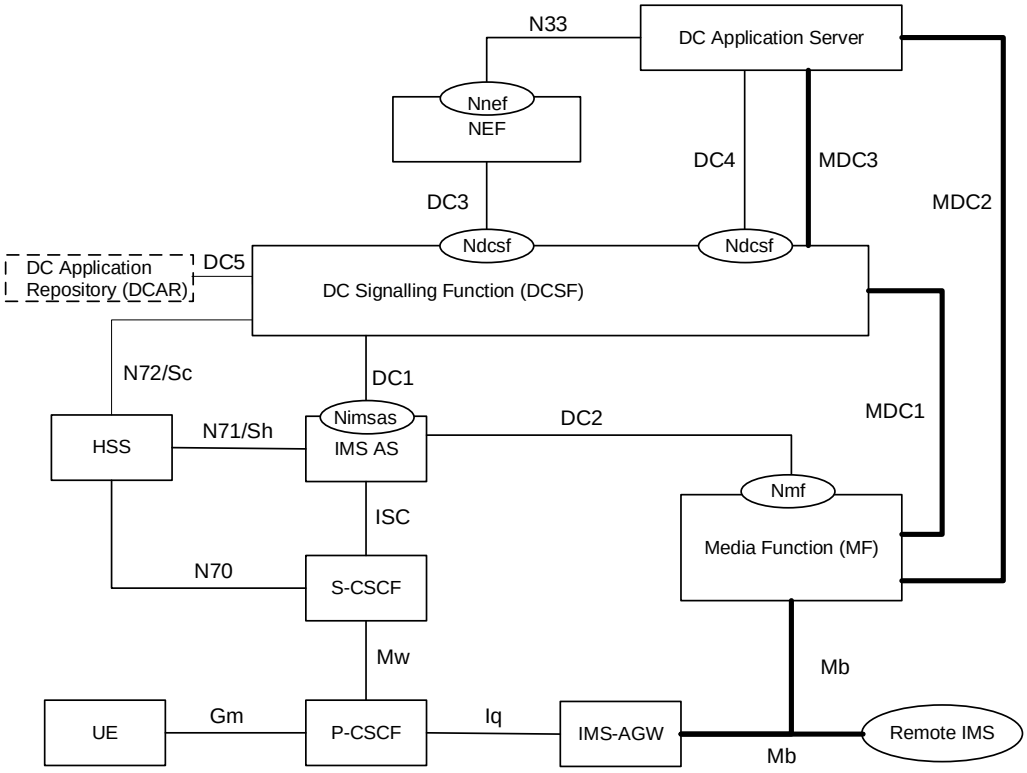


Figure 5.5.1-1: Architecture extensions to IMS to support data channels

The DC Application Server acts as the endpoint for application data channels. It communicates with the DCSF for resource control and traffic forwarding and also supports the interaction with multiple UEs for simultaneous data channel applications.

The DCSF manages the signalling control for data channels. It implements data channel control and manages resources for both bootstrap and application data channels. It also manages the download and configuration of data channel applications from the DCAR (Data Channel Application Repository).

The Media Function (MF) manages media resources and forwards data channel traffic. It terminates the bootstrap data channel from the UE and forwards HTTP traffic to the DCSF. The MF provides the media resources to anchor application data channels and relays the traffic between the UEs. The MF may terminate the application data channel by acting as an HTTP proxy or it may simply relay traffic by acting as UDP proxy.

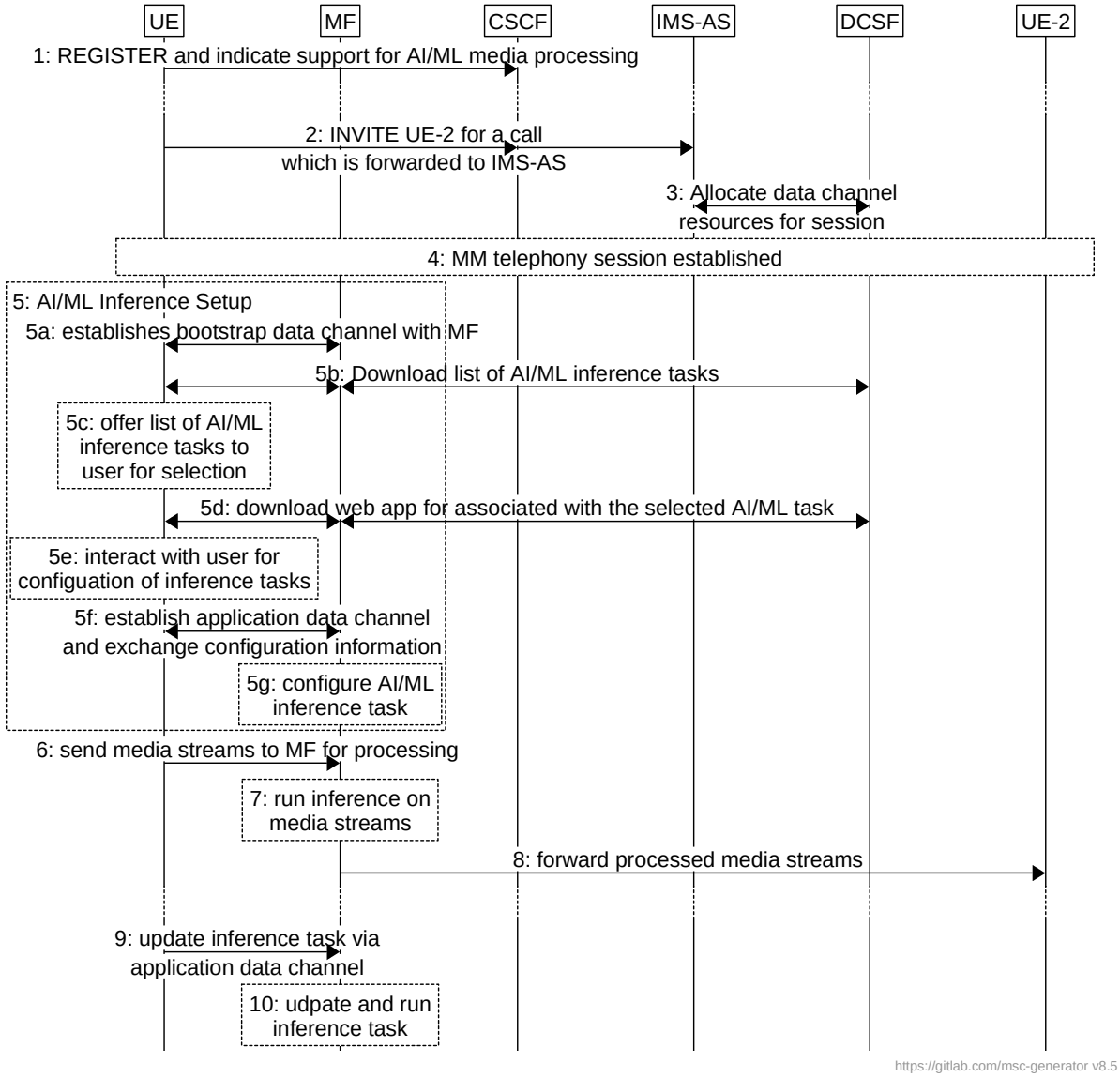
The DCAR stores and manages verified data channel applications. These can then be downloaded by the UE through the DCSF and MF.

The IMS AS is enhanced to support data channel functionalities and manage interactions between the different entities.

5.5.2 Mapping AI/ML Media Processing to IMS

Several AI/ML use cases are analysed and documented in clause 4.2. Among these use cases are the NLP use cases, which describe a wide range of media processing tasks that may be applied to a multimedia call. To support these use cases, it may be possible to integrate the media processing with the media of the MTSI call. This is best done by leveraging the already existing data channel infrastructure, where AI/ML media processing is treated as a special category of web applications.

The supported media processing may be discovered as part of the data channel application discovery process over the bootstrap channel. The AI/ML media processing itself may be run in the UE, MF, or on both (split processing). The following flow chart depicts the process to trigger AI/ML media processing:



The steps are as follows:

1. The UE registers with the SIP registrar and indicates its ability to support AI-based media processing, e.g. through the inclusion of a dedicated parameter of the Contact header field,
2. The UE invites UE2 for a call by sending a SIP INVITE. Alternatively, the UE receives an invite to join a call.
3. The IMS-AS sets up the data channel resources for the connection,
4. The session is established by forwarding the INVITE to the remote UE and receiving the acknowledgement,
5. Setting up the AI/ML inference task using the data channel procedures:
   a. The UE establishes a bootstrap data channel to the DCSF through the MF
   b. The UE downloads a list of the DC applications enhanced with the list of available inference tasks,
   c. The telephony application displays a selection screen with the list of available inference tasks to the user for selection,
   d. Once the user has selected a inference task, the UE downloads a web application associated with the selected AI/ML inference task over the bootstrap channel,
   e. The telephony application renders the web application to the user and collects user's input,

f. The UE then establishes an application data channel to the MF/MRF and and shares any information acquired from the user's input to the MF/MRF,

g. The MF/MRF may use the exchanged application information to configure the selected AI/ML inference task accordingly,

[Editor's note: Needs to study how the MF processes the application information for configuration]

6. The UE sends the audio and video streams of the call through the MF/MRF for processing,
7. The MF/MRF runs the selected inference task with the call media streams as input,
8. The MF/MRF forwards the processed media streams to UE2,
9. The user may update or change the AI/ML inference task. The updates are sent to the MF/MRF via the application data channel,
10. The MF/MRF updates the AI/ML inference task accordingly and continues the inference.

NOTE: The AI/ML inference task may create additional media streams that are then included in the updated SDP offer automatically by the IMS AS.

As described by the flow chart diagram, launching and applying AI/ML inference tasks on the media streams of a call may be realized through the usage of data channels. Bootstrap data channel is used to discover available AI/ML inference tasks. The application data channel may then be established to select, configure, and manage the AI/ML inference tasks.

Split inference may be configured by downloading the DNN model part as part of the web application resources and then using web technologies such as WebCodes and WebNN on the UE side to do the UE inference.

5.5.3 Discovery of AI/ML tasks for IMS calls

Editor's note: Approach in this clause to be validated further.

This clause outlines the procedures, interfaces, and processing for using the IMS data channel framework to discover, select, and configure AI/ML tasks in the context of an ongoing IMS session.

All UEs that support IMS data channels are expected to establish a bootstrap data channel as described in 3GPP TS 26.114 (Rel-17 or later). This bootstrap channel is used for initial data channel signaling and for retrieving a list of available data channel applications from the Data Channel Signaling Function (DCSF). The UE uses HTTP over this bootstrap channel to request the application catalog from the DCAR through the DCSF and the Media Function (MF). The request may also include other filtering parameters to enable the DCSF to filter applications based on operator policies and UE capabilities.

The DCAR is expected to reply with a catalog that includes both standard web applications and AI/ML tasks. AI/ML tasks may be annotated with clear metadata (e.g., AI-Task or Media-Processing) so that the UE can distinctly present them to the user.

Note that this may require coordination with SA2 to enable this annotation of AI/ML tasks (or media processing tasks in general), which have the capability to process and modify the media of a IMS call and may require dedicated processing resources at the MF.

Once the user selects an AI/ML task, the UE downloads the associated configuration web application over the bootstrap channel.

If the AI/ML task and inference are fully run on the UE, the model and all related resources are downloaded as part of the DC web application and no additional configuration is required. How the web application modifies the media streams in this case is FFS and may be left to implementations.

When an AI/ML task is selected, the UE needs to establish a separate application data channel to handle configuration messages. This application data channel is set up according to standard IMS data channel negotiation procedures, ensuring it is separate from the bootstrap channel. The protocol details and message formats used on this application data channel for AI/ML configuration can be proprietary, but the underlying data channel transport is expected to comply with IMS data channel security, integrity, and resource control requirements.

The MF needs to be capable of interpreting or forwarding these AI/ML configuration messages. At a minimum, the configuration messages may include:

1. Identification of the ML Task to execute by the MF
2. Identification of the media streams to be processed (based on SDP mid attributes or equivalent identifiers).
3. Direction of the processing (uplink, downlink, or both).
4. Model-specific parameters (e.g., inference thresholds, language preferences, or any additional control flags).

The UE has to correlate the AI/ML configuration parameters with the corresponding media lines advertised in the SDP. This correlation typically relies on the a=mid: attribute in each media description. If the AI/ML task introduces new media streams (for example, a synthesized audio stream), the IMS-AS needs to insert the updated SDP lines in a re-INVITE procedure.

Below is an example of an SDP offer that includes an audio stream, a video stream, and a data channel for AI/ML configuration. In this example, an attribute (a=processing-task) is shown to illustrate how the AI/ML task might reference specific media streams via the mid identifiers.

```
v=0
o=- 538051857716 2 IN IP4 203.0.113.1
s=AI/ML Session
t=0 0

m=audio 49170 RTP/AVP 97
c=IN IP4 192.114.1.1
a=rtpmap:97 EVS/16000/1
a=mid:audio0

m=video 51372 RTP/AVP 101
c=IN IP4 192.114.1.1
a=rtpmap:101 H264/90000
a=mid:video0

m=application 5000 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 192.114.1.1
a=mid:data0
a=sctp-port: 5000
a=dcmap:0 label="processing-task"
a=processing-task:0 mid=audio0 direction=uplink model="ASR-ModelA"
a=processing-task:1 mid=video0 direction=downlink model="VideoEnhanceB"
```

When the remote endpoint receives this offer, it needs to respond with an SDP answer that either confirms or rejects the proposed AI/ML configuration attributes, in accordance with normal IMS session establishment rules. If the remote

endpoint supports the data channel and the AI/ML tasks, it responds with matching data channel parameters and indicates acceptance of the proposed AI/ML tasks.

Once the application data channel is established, the UE executes the configuration web application locally. This web application may communicate with the MF using proprietary messages that define AI/ML parameters. The MF interpretes these messages and configures the AI engine handling the inference tasks. The UE needs to ensure that any changes in user preferences (e.g., switching models, changing language preferences) are immediately signaled over the data channel.

If the MF fails to allocate resources or if the requested model is unavailable, it needs to provide an appropriate error response. The UE then informs the user of any such failures in a timely manner.

In scenarios where split inference is used (part of the AI/ML model runs on the UE, and part runs on the MF), the UE is expected to download the relevant DNN model fragment as part of the configuration web application. Local processing may employ WebNN or a similar in-device library. The MF needs to process any partial inference data or complementary inference tasks that the UE offloads. This coordination of split tasks is signaled through the same application data channel used for AI/ML configuration, maintaining a consistent view of the AI/ML pipeline among the UE, MF, and any external AI entities.

If, at any point, the user decides to change or deactivate the AI/ML task, the UE sends updated configuration messages over the application data channel. The MF then adjusts the AI/ML pipeline accordingly. If the modifications necessitate new media lines or the removal of media lines, the UE needs to perform a re-INVITE transaction. The IMS AS propagates these changes to the remote endpoint, ensuring that session state remains consistent across the entire IMS domain.

6 Data components for AI/ML-based media services

6.1 General

Based on the identified use cases in clause 4.2, this clause documents the different data components involved in AI/ML-based media services. The delivery of certain data components in either the downlink or uplink direction between the UE and the network is dependent on both the use case requirements and service configuration.

AI/ML data as handled by the AI/ML Data Access/Delivery function consists of AI/ML model data (the data representing trained or untrained models, as well as their associated parameters/weights), and in the case of split AI/ML inferencing, AI/ML intermediate data (the data output from the endpoint performing a first split inference, typically sent as the data input into the endpoint performing a second split inference).

The data formats for AI/ML data components is dependent on the framework used; existing frameworks such as TensorFlow and PyTorch are popular frameworks in both the tech industry and academia. ONNX provides an interoperable AI/ML model format with a uniform model representation to facilitate the exchange of machine learning models between different other AI/ML frameworks (Pytorch, TensorFlow). ONNX may be used to convert a trained model (e.g. Pytorch or TensorFlow) for inference in an ONNX runtime adapted for deployment target.

Clause 6.6 also introduces certain types of metadata which may be relevant to AI/ML-based media services, namely that related to AI/ML model information, split AI/ML operations and service requirement/endpoint capability.

6.2 Model data

6.2.1 Model optimization techniques

Trained models consist of a graph representation of neural networks as well as millions of parameters/weights that are learned during the training phase.

Parameter pruning is one of the main techniques to control the size of a neural network model or an update thereof. Pruning works by removing individual weights or complete structures of a pre-trained model. There is a difference between structured and unstructured pruning. In unstructured pruning, the goal is to reduce the number of non-zero

weights in a layer while approximately preserving the output of that layer. The assumption behind this technique is that only a small subset of the weights is dominant and impacts the performance of the model. The rest of the weights may potentially be ignored/removed. The technique starts by assigning a saliency score to each parameter and then removes the weights with a score below a certain threshold. The resulting network may require retraining to regain the original accuracy. However, this type of technique introduces unstructured sparsity into the neural network, but the resulting tensors of parameters have the same size and shape. The receiver may require special inference hardware or some pre-processing to reduce the inference computational complexity.

In structured pruning, the model graph is altered by completely removing certain structures such as neurons and filters. This may be done by assigning an importance score to each neuron/filter based on the current weight or based on inference data. The neurons/filters with a score below a threshold are removed. Compared to unstructured pruning, this approach does not introduce sparsity but may not yield the same compression results.

Low-rank decomposition is another technique to reduce the size of a model. In low-rank decompression, a tensor, representing the weights of a layer in the DNN, is replaced by a product of two lower-rank tensors in which reduces the number of element-wise multiplications potentially without sensibly altering the performance, providing a proper choice of rank. This technique can both speed up the inference and results in compression gains. Algorithms such as the Singular Value Decomposition (SVD) may be used to obtain the tensors corresponding to any desired rank.

Quantization is another compression technique. It consists of decreasing the precision of the parameters of a model, thus reducing the required memory footprint. The parameters are mapped from a larger space of values into a smaller one, a concept widely used in image and video compression. Better performing quantization techniques may be context aware and operate in a non-linear manner to approximate the distribution of the weight values. Knowledge about the used quantization scale will be required to perform inverse quantization and recover the original weights. If non-linear quantization is used, the technique becomes non-transparent. The resulting parameters may further be losslessly entropy coded, e.g. using Huffman coding.

Knowledge distillation takes a different approach to reducing model size. The goal is to transfer knowledge from a trained network into a smaller model for inference. During the distillation process, the smaller model learns to mimic the output of the larger trained model by minimizing a loss function that takes into account both the hard output values and the soft values (i.e. prior to filter application). Knowledge distillation techniques have in several cases surpassed the accuracy of the original model.

The compression levels achieved by these techniques can be controlled to provide a set or "family" of adaptive trained models which perform the same task but meet different constraints (e.g., memory footprint, latency and/or computational cost). Furthermore, by minimizing the difference between the models during training, the family can be optimized to reduce its memory footprint or the transmission cost of model changes. Examples of such approaches include:

- Pruned models, where each neural network of the family (except the largest one) contains a subset of the neurons of the previous network in the ordered family
- Quantized models, where the family contains neural networks with increasing quantization level of the parameters.
- Early-exit models, where the neural network contains exit points before reaching the final output that generate intermediate predictions/results.

Most of the techniques mentioned above are sender-only and do not require processing on the receiver side. The burden is on the creator of the model to apply these techniques to produce a more compact representation of the model. Some techniques may require processing at the receiver side. The complexity of that processing and the amount of information required to recover the model may vary by technique.

6.2.2 Model update requirements and constraints

6.2.2.1 Evolving requirements and environment conditions after model selection

Use-cases and different workflows delivery comprises the selection and the distribution of adapted trained models or model subsets to the UE for performing AI/ML inference. An offline supervised learning can provide a set of trained models adapted for the UE to environment conditions regarding a UE service requirement. Environment conditions in clause 4.1 or clause 4.3.1 describes different sets of conditions including UE capabilities and network limitations. The UE and the network share these environment parameters to select the trained model that fits best the current conditions to meet the requirements. The selection may depend for example on the current UE capabilities such as the available

memory, the current power consumption, the current battery storage, the current computing power, as well as on the current network conditions such as the network load, the available or the allocated bandwidth to the UE. This may also depend on the service requirements, or on the user preferences on the expected quality of result and on the maximum UE resources such as the energy, memory, computing power for running the AI/ML service.

During the inference stage, environment conditions as listed above may change to such an extent that the selected trained model e.g., DNNs will no longer be appropriate or not optimal to meet the requirements. This will lead to a degraded QoE for the end user. This highlights the need for model updates to meet the new environment conditions.

6.2.2.2 Model accuracy deviation between the training phase and the delivery phase.

The discrepancy between the data seen during training and data used at the time of inference can lead to a decrease in accuracy performance. The actual accuracy of the system may vary depending on the current input data, environment, and context. Updates to the trained models are necessary to continue to meet the accuracy requirements.

6.2.2.3 Applying inference on evolving characteristics of the input media content

The model to be applied can be adapted to the entire media content or sequence thereof, or to a spatial or temporal partition of an input media content, for example to a group of frames, frame slices, frame blocks. The model and/or model parameters such as biases and weights may be updated to adapt to the characteristics of the processed part of the content. The characteristics can relate to the resolution, light e.g., the noise introduced by the camera, content in dark areas, the type of scene. They can also relate to the current demand by the algorithm or the user in terms of expected accuracy or subjective quality of the produced content.

6.2.3 Model serialization

In computing, serialization (or serialisation) is the process of translating a data structure or object state into a format that can be stored (e.g., files in secondary storage devices, data buffers in primary storage devices) or transmitted (e.g. data streams over computer networks) and reconstructed later (possibly in a different computer environment).

The process of saving an AI/ML model to use it later is called serialization. After transmitting or storing the serialized data, it is possible to reconstruct the model later and obtain the exact same structure/object.

6.2.4 Classes of AI/ML models

6.2.4.1 Introduction

Depending on the training method selected, AI/ML models can operate various types of operations as depicted in the figure 6.2.4-1 below:

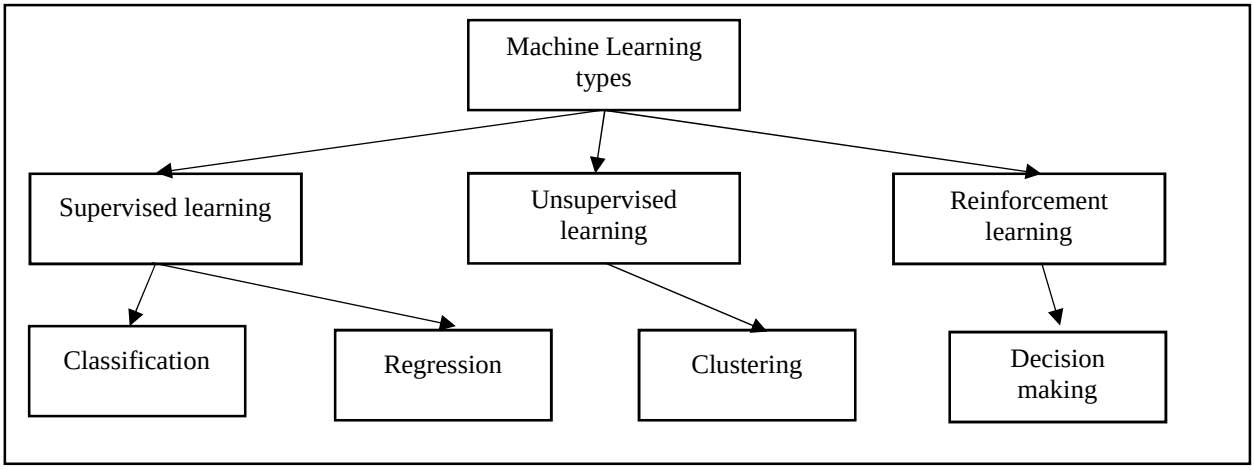


Figure 6.2.4-1: Main classes of AI/ML models

6.2.4.2 Supervised learning

As explained in [14] supervised learning accounts for a lot of research activity in machine learning and many supervised learning techniques have found application in the processing of multimedia content. The defining characteristic of supervised learning is the availability of annotated training data. The name invokes the idea of a 'supervisor' that instructs the learning system on the labels to associate with training examples. Typically, these labels are class labels in classification problems. Supervised learning algorithms induce models from these training data and these models can be used to classify other unlabelled data. The analysis of supervised learning can be seen as the theory of risk minimization. Vector machines and nearest neighbour classifiers are probably the two most popular supervised learning techniques employed in multimedia research.

6.2.4.3 Unsupervised learning

The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format. Unsupervised learning is important in the processing of multimedia content as clustering or partitioning of data in the absence of class labels is often a requirement. The absence of class labels in unsupervised learning makes the question of evaluation and cluster quality assessment more complicated than in supervised learning.

6.2.4.4 Reinforcement learning

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

RL differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead, the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

6.2.5 Existing formats for AI/ML models

6.2.5.1 ONNX format

The Open Neural Network Exchange (ONNX) format [15] is an open specification that was developed to facilitate the exchange of machine learning models between different AI/ML frameworks. ONNX consists of the following components:

- A definition of an extensible computation graph model.
- Definitions of standard data types.
- Definitions of built-in operators.

The ONNX format is built around the Protocol Buffers (Protobuf) open-source cross-platform serialization.

The ONNX Graph is structured as a list of nodes that form an acyclic graph. Each node of the graph represents one of the built-in operators and its attributes. As an example, a node could be a Convolution operation, and its attributes would contain information regarding the padding and stride that must be used. Each edge of the graph represents input or output data tensors. The top-level ONNX construct is a 'Model' and is represented in protocol buffers as the type `onnx.ModelProto`. It provides metadata that is necessary for the reader to determine if they are able to process the stored model. Each model must explicitly name the operator sets that it relies on for its functionality. Operator sets defines a set of operators and their versions. An operator is identified through its unique operator type (`op_type`), which is a case-sensitive operator name.

Built-in operators include a large list of widely used operators such as the following:

- Math operators such as Abs
- DNN operators such as Conv and LSTM
- Activation operators such Sigmoid and ReLU

- Pooling operators such as MaxPool
- Other operators such as error computation and data reformatting operators

The following provides an example of an ONNX model in protobuf format:

```
ir_version: 5
producer_name: "skl2onnx"
producer_version: "1.11"
domain: "ai.onnx"
model_version: 0
graph {
  node {
    input: "X"
    output: "Y"
    name: "Pa_Pad"
    op_type: "Pad"
    attribute {
      name: "mode"
      s: "constant"
      type: STRING
    }
    attribute {
      name: "pads"
      ints: 0
      ints: 1
      ints: 0
      ints: 1
      type: INTS
    }
    attribute {
      name: "value"
      f: 1.5
      type: FLOAT
    }
    domain: ""
  }
  name: "OnnxPad"
  input {
    name: "X"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
          }
          dim {
            dim_value: 2
          }
        }
      }
    }
  }
  output {
    name: "Y"
    type {
      tensor_type {
        elem_type: 1
        shape {
          dim {
          }
          dim {
            dim_value: 4
```

```
          }
        }
      }
    }
  }
}
opset_import {
  domain: ""
  version: 10
}
```

6.2.5.2 NNEF format

The Neural Network Exchange Format (NNEF) [16] is a Khronos developed standard that defines a data format for facilitating the exchange of trained network models. The NNEF format enables the encapsulation of both the structure of the neural network model as well as the associated data. NNEF stores the data in structures that are independent of the training environment that was used for training the network, which will facilitate its consumption on any execution platform. NNEF offers itself as an intermediary between deep learning frameworks, which export into NNEF, and neural network accelerator libraries, which will import and compile the NNEF model for hardware-optimized inference.

The NNEF container consists of the following files:

- a textual file that describes the structure of the neural network
- a binary data file for each variable tensor. These files are structured hierarchically into sub-folders associated with the corresponding operation. Each tensor may have different representations, each matching a different quantized version.
- a quantization file that contains details about the quantization algorithm that is used for quantizing the exported tensors.

The NNEF network structure is described through a computational graph. The computational graph is a directed graph. The nodes of the graph may be data nodes or operation nodes. A directed edge from a data node to an operation node indicates the data is input to the operation. A directed edge from an operation node to a data node indicates the data node is an output.

Data nodes are tensors of different ranks and shapes and may be external, constant, variable, or intermediate/regular tensors. external, constant, and variable tensors all provide an explicit declaration of their shapes. Other tensors shapes will be determined based on the input and operation that is applied to them to produce that tensor. This is commonly known as shape propagation.

The NNEF operation nodes may have attributes that describe the exact computation that needs to be performed. Operations may be composed together to produce more compound operations. Primitive operations are operations that cannot be broken down into simpler operations.

The following is an excerpt from an NNEF graph representation of the VGG-16 network model:

```
version 1.0;

graph VGG_ILSVRC_16_layers(data) -> (prob)
{
   variable_15 = variable<scalar>(label = 'conv4_1_blob2', shape = [1, 512]);
   variable_14 = variable<scalar>(label = 'conv4_1_blob1', shape = [512, 256, 3, 3]);
   variable_13 = variable<scalar>(label = 'conv3_3_blob2', shape = [1, 256]);
   variable_31 = variable<scalar>(label = 'fc8_blob2', shape = [1, 1000]);
   variable_30 = variable<scalar>(label = 'fc8_blob1', shape = [1000, 4096]);
   variable_29 = variable<scalar>(label = 'fc7_blob2', shape = [1, 4096]);
   variable_28 = variable<scalar>(label = 'fc7_blob1', shape = [4096, 4096]);
   variable_27 = variable<scalar>(label = 'fc6_blob2', shape = [1, 4096]);
   variable_26 = variable<scalar>(label = 'fc6_blob1', shape = [4096, 25088]);
   variable_25 = variable<scalar>(label = 'conv5_3_blob2', shape = [1, 512]);
   variable_24 = variable<scalar>(label = 'conv5_3_blob1', shape = [512, 512, 3, 3]);
```

```
    variable_23 = variable<scalar>(label = 'conv5_2_blob2', shape = [1, 512]);
    variable_22 = variable<scalar>(label = 'conv5_2_blob1', shape = [512, 512, 3, 3]);
    variable_21 = variable<scalar>(label = 'conv5_1_blob2', shape = [1, 512]);
    variable_20 = variable<scalar>(label = 'conv5_1_blob1', shape = [512, 512, 3, 3]);
    variable_19 = variable<scalar>(label = 'conv4_3_blob2', shape = [1, 512]);
    variable_18 = variable<scalar>(label = 'conv4_3_blob1', shape = [512, 512, 3, 3]);
    variable_17 = variable<scalar>(label = 'conv4_2_blob2', shape = [1, 512]);
    variable_16 = variable<scalar>(label = 'conv4_2_blob1', shape = [512, 512, 3, 3]);
    variable_12 = variable<scalar>(label = 'conv3_3_blob1', shape = [256, 256, 3, 3]);
    variable_10 = variable<scalar>(label = 'conv3_2_blob1', shape = [256, 256, 3, 3]);
    variable_9 = variable<scalar>(label = 'conv3_1_blob2', shape = [1, 256]);
    variable_8 = variable<scalar>(label = 'conv3_1_blob1', shape = [256, 128, 3, 3]);
    variable_6 = variable<scalar>(label = 'conv2_2_blob1', shape = [128, 128, 3, 3]);
    variable_11 = variable<scalar>(label = 'conv3_2_blob2', shape = [1, 256]);
    variable_5 = variable<scalar>(label = 'conv2_1_blob2', shape = [1, 128]);
    variable_4 = variable<scalar>(label = 'conv2_1_blob1', shape = [128, 64, 3, 3]);
    variable_2 = variable<scalar>(label = 'conv1_2_blob1', shape = [64, 64, 3, 3]);
    variable_1 = variable<scalar>(label = 'conv1_1_blob2', shape = [1, 64]);
    variable_7 = variable<scalar>(label = 'conv2_2_blob2', shape = [1, 128]);
    variable = variable<scalar>(label = 'conv1_1_blob1', shape = [64, 3, 3, 3]);
    variable_3 = variable<scalar>(label = 'conv1_2_blob2', shape = [1, 64]);
    data = external<scalar>(shape = [10, 3, 224, 224]);
    conv = conv(data, variable, variable_1, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1, 1), (1, 1)],
stride = [1, 1]);
    relu = relu(conv);
    conv_1 = conv(relu, variable_2, variable_3, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1, 1), (1,
1)], stride = [1, 1]);
    relu_1 = relu(conv_1);
    max_pool = max_pool(relu_1, border = 'ignore', padding = [(0, 0), (0, 0), (0, 0), (0, 0)], size = [1, 1, 2, 2], stride =
[1, 1, 2, 2]);
    conv_2 = conv(max_pool, variable_4, variable_5, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1,
1), (1, 1)], stride = [1, 1]);
    relu_2 = relu(conv_2);
    conv_3 = conv(relu_2, variable_6, variable_7, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1, 1),
(1, 1)], stride = [1, 1]);
    relu_3 = relu(conv_3);
    max_pool_1 = max_pool(relu_3, border = 'ignore', padding = [(0, 0), (0, 0), (0, 0), (0, 0)], size = [1, 1, 2, 2], stride
= [1, 1, 2, 2]);
    conv_4 = conv(max_pool_1, variable_8, variable_9, border = 'constant', dilation = [1, 1], groups = 1, padding =
[(1, 1), (1, 1)], stride = [1, 1]);
    relu_4 = relu(conv_4);
    conv_5 = conv(relu_4, variable_10, variable_11, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1,
1), (1, 1)], stride = [1, 1]);
    relu_5 = relu(conv_5);
    conv_6 = conv(relu_5, variable_12, variable_13, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1,
1), (1, 1)], stride = [1, 1]);
    relu_6 = relu(conv_6);
    max_pool_2 = max_pool(relu_6, border = 'ignore', padding = [(0, 0), (0, 0), (0, 0), (0, 0)], size = [1, 1, 2, 2], stride
= [1, 1, 2, 2]);
    conv_7 = conv(max_pool_2, variable_14, variable_15, border = 'constant', dilation = [1, 1], groups = 1, padding =
[(1, 1), (1, 1)], stride = [1, 1]);
    relu_7 = relu(conv_7);
    conv_8 = conv(relu_7, variable_16, variable_17, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1,
1), (1, 1)], stride = [1, 1]);
    relu_8 = relu(conv_8);
    conv_9 = conv(relu_8, variable_18, variable_19, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1,
1), (1, 1)], stride = [1, 1]);
    relu_9 = relu(conv_9);
    max_pool_3 = max_pool(relu_9, border = 'ignore', padding = [(0, 0), (0, 0), (0, 0), (0, 0)], size = [1, 1, 2, 2], stride
= [1, 1, 2, 2]);
    conv_10 = conv(max_pool_3, variable_20, variable_21, border = 'constant', dilation = [1, 1], groups = 1, padding
= [(1, 1), (1, 1)], stride = [1, 1]);
```

```
    relu_10 = relu(conv_10);
    conv_11 = conv(relu_10, variable_22, variable_23, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1,
1), (1, 1)], stride = [1, 1]);
    relu_11 = relu(conv_11);
    conv_12 = conv(relu_11, variable_24, variable_25, border = 'constant', dilation = [1, 1], groups = 1, padding = [(1,
1), (1, 1)], stride = [1, 1]);
    relu_12 = relu(conv_12);
    max_pool_4 = max_pool(relu_12, border = 'ignore', padding = [(0, 0), (0, 0), (0, 0), (0, 0)], size = [1, 1, 2, 2],
stride = [1, 1, 2, 2]);
    reshape = reshape(max_pool_4, shape = [10, -1]);
    linear = linear(reshape, variable_26, variable_27);
    relu_13 = relu(linear);
    linear_1 = linear(relu_13, variable_28, variable_29);
    relu_14 = relu(linear_1);
    linear_2 = linear(relu_14, variable_30, variable_31);
    prob = softmax(linear_2, axes = [1]);
}
```

6.2.5.3 Neural Network Coding (NNC) format

The Neural Network Coding (NNC) standard [17] has been developed by ISO/IEC for transmission and storage of machine learning models for multimedia description and analysis. It specifies a compressed representation format for neural network data and processes for its decoding. As shown in Figure 6.2.5-1, NNC follows a toolbox approach: It offers a variety of options to represent and code neural network (NN) data, which can be flexibly selected based on the requirements of a particular use case. In particular, NNC defines data structures and syntax elements to support the following:

- Packaging of NN data of different types in neural network representation (NNR) units for access from a system or application layer.
- Signalling of metadata related to various methods of pre-processing for data reduction
- Compression of NN weights/tensor coefficients using quantization and entropy coding
- Interoperability with other exchange (e.g. NNEF [16], ONNX [15]) or native formats (PyTorch, TensorFlow).

For access from a systems or application layer, NNC packages the NN data in neural network representation (NNR) units. NNR units that can carry different types of NN data: *NNR parameter set* and *NNR layer parameter set units* convey metadata and information related to the entire NN and individual NN layers, respectively. *NNR topology units* contain information on the NN topology, e.g. the connections between layers/tensors. The actual tensor data is conveyed in *NNR quantized information* and *NNR compressed data units*. Finally, *NNR aggregate units* allow to combine several NNR units of different types that are related.

NNC allows to signal metadata related to typical pre-processing and parameter reduction methods in *NNR parameter set units* or *NNR layer parameter set units*. More specifically, NNC supports inclusion of parameters related to sparsification, pruning, low-rank decomposition, unification, batch norm folding, and local scaling.

NNC represents the NN weights/tensors in *NNR compressed* or *NNR quantized information data units*. Tensor/weight coefficients can be signalled as raw data or quantized with different methods, which are uniform, codebook, or dependent quantization. Furthermore, the quantized coefficients can be binarized and entropy coded using a context adaptive arithmetic coder, called DeepCABAC.

NNC can be used as complement to other native (e.g. PyTorch, TensorFlow) or exchange (e.g. NNEF, ONNX) representation formats. This can be done by two means: First, NNC allows to embed topology information of other formats into an NNR bitstream. More specifically, the byte sequences of other formats can be signalled in *NNR topology units*, which are then conveyed together with *NNR compressed data* or *NNR quantized information units* representing the coded or quantized tensors/weights. Second, NNR units representing coded tensors/weights can be embedded in the containers of other formats. Informative recommendations on how to use NNC in combination with PyTorch, TensorFlow, NNEF, and ONNX are given in the Annexes A to E of the standard [17].

NNC also supports compression of incremental updates of neural networks, which can e.g. be applied for updating neural networks or in federated learning scenarios. For such distributed AI/ML scenario, NNC also comprises dedicated high-level syntax that allows to identify updates, their order, and their dependencies.

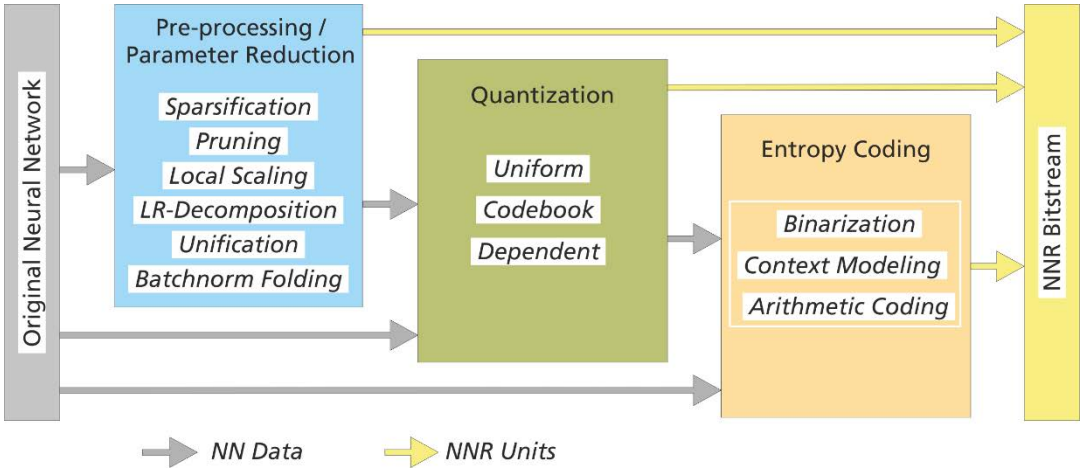


Figure 6.2.5-1: Generation of a neural network representation (NNR) bitstream consisting of NNR units.

As shown in figure 6.2.5-1, tools for pre-processing, parameter reduction, quantization, and entropy coding may be selected based on the complexity and compression requirements of a given use case.

6.2.5.4 PyTorch formats for model distribution

PyTorch provides several formats for distributing machine learning models, such as PyTorch JIT (Just-In-Time) and TorchScript. PyTorch JIT allows for models to be compiled on-the-fly, which provides performance benefits for large models or when deploying to resource-constrained environments. TorchScript allows for models to be exported to a portable format that can be executed on various platforms, such as mobile devices, web browsers, and embedded systems.

6.3 Intermediate data

6.3.1 Introduction

Split AI/ML operation is defined as the distribution of AIML model inferencing between at least two endpoints, for example a UE and a Network endpoint. The data output from the first endpoint (intermediate data) is delivered to the second endpoint to guarantee the expected user experience on running a particular AIML application regarding UE, Network and server capabilities. Requirements for such a split inference service may include avoiding service interruption, and optimizing the network, UE or server resources.

6.3.2 Intermediate data size delivery

Intermediate data characteristics depends on various aspects described in clause 4.1 and clause 4.5 including intermediate data volume or size.

Different factors can impact the size of the intermediate data for delivery, which may require the adaptation of split AI/ML operations between the UE and the network:

- AI/ML inference task use-case and requirement: The service requirements on an AI/ML task drives the intermediate data size. For example, a complex AI/ML task for detecting multiple objects in a dense and moving video requires far more intermediate data than for a simpler AI/ML task on static scene or about a single object.
- AI/ML model for the AI/ML inference task: Different trained AI/ML models for the same AI/ML inference task can be available with different characteristics on not only the AI/ML model architecture and size, but also on the intermediate output size, depending on the split point(s).

- Split point selection: The selection of a split point within an AI/ML model determines the dimension of the intermediate data. The output size at a given split point compared to another may vary from 1 to 5 or more as reflected in 3GPP TR 22.874 [2].

- Adapted trained model for split operation: Adapted models can be designed to provide reduced intermediate data at identified split points [18].

- Optimization: accuracy/quality metrics determine the result of a split inference. Basic precision quantization, from 32 bits to 16/8 bits may reduce the overall size of intermediate data while still meeting the required output result quality/accuracy for the service.

- Inference input video frame rate adjustment: The input frame rate in case of video determines the streaming bitrate of the intermediate data to be delivered. An AI/ML inference task may not produce media content and does not necessarily need to produce an output result at 30 or 60 frames as in the case of video streaming.

- Non-real time delivery: The transmission of intermediate data may not necessarily need to be delivered in a real-time based manner. The result of inferencing split model on an image, a set of images or a video sequence may not require an immediate result. The transmission of intermediate data can be done progressively with a constrained bandwidth,

- Different input image resolutions may produce different intermediate data size for models with variable input size (e.g. image classification models)

6.3.3 Operations for splitting AI/ML models

For split configurations, clause 5.1, an AI/ML model is split in two subsets consisting of a head subset or a part 1 running on a first endpoint (UE resp. Network) and a tail subset or part 2 running on a second endpoint (Network resp. UE). The first endpoint provides intermediate data to the second endpoint over the network.

The computational graph of a model above may be represented by a directed acyclic graph with existing formats or frameworks such as ONNX, NNEF, TensorFlow and PyTorch as described in clause 6.2.5 and 6.4.

A simple split operation on a particular node in such a graph consists of a single branch split when the split node has only one input edge and if the split occurs before the split node (resp. one single output edge if the split occurs after the split node).

A multi-branch split occurs when a split node has more than one input and if the split occurs before the node or more than one output if the split occurs after the node.

Certain specific aspects may need to be considered when applying a split operation to divide a model into two or more subsets:

- A model graph may start taking input media data into account at a node later than the first node. In this case, if the split occurs before the node requiring the input media data, this input media data needs to be provided to the second part of the subset running in the second endpoint, in addition to the intermediate data generated by the first part.

- A model graph may comprise multiple nodes producing partial results. In this case, a split occurring after a node producing partial results will require to collect partial results from both endpoints to compute the final consolidated results.

ONNX is an interoperable format available for frameworks 6.4 and for a large number of models (https://github.com/onnx/models). Splitting a single branch is straightforward using extract_model function (https://onnx.ai/onnx/api/utils.html). A generic multi-branches split using the same function may also be performed after parsing the model to obtain the list of inputs and outputs required by each model subset.

Different endpoints may split an ONNX model down to the split node or from the split node to the end by using the same node identification and parsing rules to apply to the graph. For example, a first endpoint may build the head subset from the full ONNX model and the second endpoint may build the tail subset from the same full ONNX model.

One endpoint may split model from an ONNX model file in two subsets and deliver the required subset to an endpoint

For dynamic split point reselection, an endpoint may prepare different model subsets for the different candidate split point configurations or build the split model subset ‘on demand’ from a full ONNX model when it is required.

6.3.4 Compression related functions

Depending on the AI/ML media service use case (and the required AI/ML task) some compression approaches (e.g., quantization, entropy coding, transformations) can be used to reduce the size of the transferred intermediate data and to adapt the split AI/ML operations between the UE and the network to changing conditions.

Compression functions such as quantization, entropy coding, pruning may be applicable to any intermediate data tensors. Some of these functions require corresponding decompression processes to decode and readapt the intermediate data for the inference of the second part of the model. Different ratios of reduction of intermediate data size can be reached for each split point configuration, for instance by varying quantization parameters. These agnostic compression functions can be used for any model, any split point, any type of model task with different input media data (image, video, audio, text). Agnostic compression evaluations with on-the shelf compression functions provide promising performances in terms of intermediate data size relative to the accuracy of the results.

Another approach to further compress intermediate data is to inject a so-called bottleneck to create smaller intermediate data with an auto-encoder-like structure. Two main cases can be distinguished.

- An original design of the split model can include natural split points, i.e. intermediate layers where the dimensions of intermediate output are reduced by design and their entropy is controlled. The training of this model can be done using a loss function which includes both the end accuracy and the estimation of the size of the bitstream of intermediate data. [19] shows an example of intermediate data bottlenecks using an embedded autoencoder.

- A compression function including bottleneck can be inserted at the encoder side and a corresponding decompression function at the decoder side without modifying the original structure of the split model. The model including the bottleneck may need to be designed and trained for each split model and each split point. Envisioned frameworks may enable a sender to select an optimised function from a set of adaptive compression/decompression functions based on different trained bottlenecks applying different compression factors. The encoder may need to send an indication of the compression profile used for the intermediate data transfer to the decoder side to select and use the corresponding decompression function. For instance, MPEG-FCM as mentioned in clause 4.3 is working on the standardization of such codec, with a current model that includes both the bottleneck concept with trained feature reduction at the encoder and restoration at the decoder. In addition to the feature reduction, the current design maps resulting reduced feature tensors to video frames in order to utilize existing video codecs such as H.264/AVC, H.265/HEVC or H.266/VVC to further compress the features and take advantage of advance temporal compression methods in the case of feature streaming. MPEG-FCM currently delivers very promising performances on intermediate data compression applied to video making use of video decoders such as H.265/HEVC as shown in table 6.3.4-1.

The table 6.3.4-1 below summarizes the different approaches and their characteristics. The reported ratios is the size difference between the original intermediate data and the compressed intermediate data, at near lossless compression accuracy, defined as a final accuracy drop of less than 1% of the original (un-split) model.

Table 6.3.4-1: Approaches and characteristics considered by MPEG FCM.

Approaches	Agnostic	Training required	Number of split points	Compression ratio	Reference
Bottleneck model	No	Yes	One to several	Unknown	research [19]
Basic quantization	Yes	No	Any	2:1 to 4:1	3GPP TR 26.847 [20]
Quantization with Entropy coding with NNC (Neural Network Coding) Codec	Yes	No	Any	5:1 to 10:1(**)	3GPP TR 26.847 [20]
MPEG-FCM (current)	No	Yes (*)	One	6000:1 to 40000:1	[30]
Approaches	**Agnostic**	**Training required**	**Number of split points**	**Compression ratio**	**Reference**
Bottleneck model	No	Yes	One to several	Unknown	research [19]
Basic quantization	Yes	No	Any	2:1 to 4:1	3GPP TR 26.847 [20]
Quantization with Entropy coding with NNC (Neural Network Coding) Codec	Yes	No	Any	5:1 to 10:1(**)	3GPP TR 26.847 [20]
MPEG-FCM (current)	No	Yes (*)	One	6000:1 to 40000:1	[30]

NOTE: The MPEG-FCM results have not been verified.

(*) A retraining is only required for MPEG-FCM feature reduction and restoration functions (see clause 4.3.3).

(**) At this 10:1 ratio, the accuracy is above the 1% loss.

6.4 Existing frameworks for AI/ML

6.4.1 TensorFlow

6.4.1.1 Introduction

TensorFlow [21] is an open-source platform for creating and deploying machine learning models. It provides a wide range of tools (e.g., mode optimization) and libraries (decision forests, Ranking extensions…) for building and training models, and supports several formats for model distribution, including TensorFlow SavedModel, TensorFlow Lite, and TensorFlow.js. These formats allow models to be easily distributed across different platforms and devices, making it easier to deploy machine learning models in various applications.

6.4.1.2 Tensor

In machine learning, a tensor is a multi-dimensional array of numerical data. A tensor may have any number of dimensions, and each dimension represents a specific feature or attribute of the data. For example, a 1-dimensional tensor usually represents a vector of values, such as a list of numbers, while a 2-dimensional tensor can represent a matrix of values, such as an image.

Tensors are are used to represent the input data and the parameters of the machine learning model. For example, in image recognition, the input data is often represented as a tensor of pixel values, while the parameters of the model, such as the weights and biases, are represented as tensors as well.

Operations applied to tensors can be addition, multiplication, and convolution. These operations are used to perform mathematical computations on the tensors, which are then used to train the machine learning model.

In summary, a tensor is a multi-dimensional array of numerical data that is a fundamental data structure used in many machine learning frameworks. It is used to represent the input data and the parameters of the machine learning model and is manipulated using mathematical operations to train the model.

6.4.1.3 Usage of TensorFlow

The following steps are usually defined:

Definition of the computational graph: In TensorFlow, a machine learning model is represented as a computational graph, which is a series of operations (nodes) that are connected by edges. The nodes represent mathematical operations, such as addition, multiplication, or convolution, and the edges represent the flow of data between the nodes. To define the graph, developers use the TensorFlow API to create nodes and connect them in a specific order.

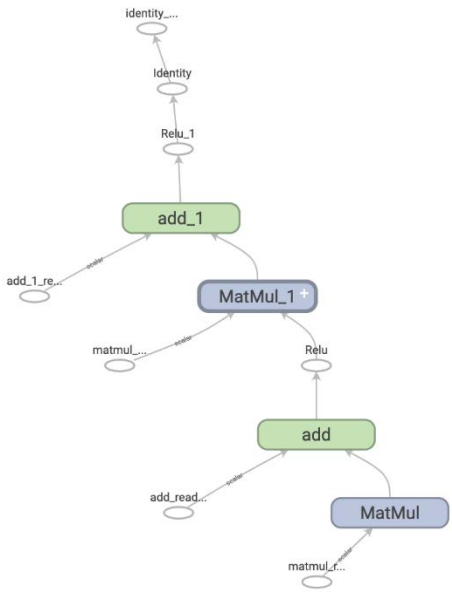


Figure 6.4.1-1: Tensorflow computational graph

Source: https://www.tensorflow.org/guide/intro_to_graphs?hl=fr

TensorFlow graphs as illustrated in figure 6.4.1-1 may be used in environments that don't have a Python interpreter, like mobile applications, embedded devices, and backend servers.

Variables Initialization: Before running the computational graph, the variables used in the graph need to be initialized. These variables represent the parameters of the machine learning model, such as weights and biases, and are updated during training to improve the model's performance.

Session execution: To execute the computational graph, a TensorFlow session is created. The session runs the graph by feeding input data into the graph and calculating the output. During training, the session updates the variables in the graph based on the loss function and optimization algorithm.

Model serialization: Once the model is trained, it can be saved in various formats for later use, such as TensorFlow SavedModel, TensorFlow Lite, or TensorFlow.js. These formats allow the model to be easily deployed on various platforms and devices, including mobile devices, web browsers, and embedded systems.

NOTE: It is expected to analyse the different distribution AI/ML formats that can be used with the TensorFlow framework and the impacts of the selection of TensorFlow framework in terms of interoperability of the corresponding AI/ML formats.

Model deployment: To deploy the model, the saved model can be loaded into a new TensorFlow session and used to make predictions on new data. This can be done on a single machine, a cluster of machines, or in the cloud.

6.4.2 PyTorch

6.4.2.1 Introduction

PyTorch is based on the concept of tensors, which are multi-dimensional arrays of numerical data. Similarly to TensorFlow, Tensors are a fundamental data structure used in PyTorch to represent the input data and the parameters of

the machine learning model. PyTorch provides a range of operations for manipulating tensors, such as addition, multiplication, and convolution.

PyTorch also supports dynamic computation graphs, which allow for more flexibility in building and training machine learning models. This means that the computational graph can be modified on-the-fly during runtime, which makes it easier to build complex models and experiment with different architectures. Additionally, PyTorch provides a high-level API called TorchScript, which allows for models to be exported to a portable format that can be executed on various platforms.

6.4.2.2 Main differences with TensorFlow

6.4.2.2.1 Computational graph

TensorFlow uses a static computational graph, which means that the graph is defined and compiled before the training begins. On the other hand, PyTorch uses a dynamic computational graph, which allows for more flexibility in building and modifying the graph during runtime.

6.4.2.2.2 Ease of use

PyTorch is generally considered to be more user-friendly and simpler than TensorFlow. This is partly due to its dynamic computational graph, which makes it easier to experiment with different models and architectures. PyTorch also has a more Python-like syntax, which is familiar to many developers.

6.4.2.2.3 Visualization

TensorFlow provides comprehensive visualization tools (such as TensorBoard), which allows users to monitor the training progress and visualize the model's performance. PyTorch does not have a built-in visualization tool, but may be integrated with Tensorboard, and there are also several third-party libraries available, such as Visdom.

6.4.2.2.4 Ecosystem

TensorFlow has better support for deploying models on mobile devices and in production environments. However, PyTorch has been gaining popularity in recent years and has a growing ecosystem.

6.4.2.2.5 Research

PyTorch is more popular in the research community, as it allows for faster prototyping and experimentation due to its dynamic computational graph. TensorFlow is more commonly used in industry for production-level applications due to its static graph and better support for deployment.

6.5 Media data

Media data for AI/ML media services typically include image, video, text, audio and speech. In the context of the use cases documented in clause 4.2, such media data may be sent in the uplink direction from the UE, or in the downlink direction to the UE. Media data is typically agnostic to the delivery architecture but may have different requirements and characteristics depending on the use case. Likewise, the delivery architecture used is dependent on the use case, using either 5GMS or RTC defined pipelines and architectures.

3GPP TS 26.511 [23] defines the integration of several media codecs into 5G Media Streaming and provides requirements and recommendations for the support of these media profiles in specific 5G Media Streaming profiles.

3GPP TS 26.113 [24] specifies media capabilities for RTC endpoint terminals, referencing the UE codec requirements for speech/audio and video as specified in 3GPP TS 26.114 [25].

6.6 Metadata

6.6.1 Introduction

Metadata for AI/ML media services may include information describing AI/ML models, inference requirements, endpoint capabilities (UE or network) and information more specific to the configuration, control and management of the basic AI/ML service scenarios (AI/ML model delivery, split AI/ML operation and distributed/federated learning).

NOTE: The delivery of the metadata described in this clause is not specified.

NOTE 2: How consistent and interoperable signaling of metadata (such as of existing framework mentioned in clause 6.4) may be ensured is for further study.

6.6.2 Common AI/ML model information

AI/ML model information metadata is used to describe the characteristics of AI/ML models which may be used for an AI/ML media service. This information may be common to all three AI/ML service scenarios, and may be used in the selection of a suitable AI/ML model by the UE or network, given an AI/ML media service.

Table 6.6.2-1: Common AI/ML model information

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Model information	**Model identifier**	An identifier for an AI/ML model (or variants of it) specified for a certain AI/ML media service. The identifier may be a name, a number, a combination thereof, a hash value. The identifier is defined during the configuration stage.	model_1, model_2
	Number of parameters	Total number of parameters in the neural network.	11 million
	Model size	The size of the AI/ML model file in megabytes.	40MB
	Input size	The maximum size of the input data supported by the AI/ML model in kilobytes.	256 KB
	Output size	The maximum size of the output data supported by the AI/ML model in kilobytes.	256 KB
	Accuracy	The trained accuracy of the AI/ML model as a percentage.	85%
	Target inference latency	The target inference latency specified for a given AI/ML model in milliseconds. Such latency is measured between the input and output layers of the AI/ML model at inference. This value is related to the service inference latency requirement of the service for which the AI/ML model is provided, as well as the typical hardware capabilities of an entity performing the inference of the model.	20ms
	Format/ framework	The format or framework used to express the AI/ML model, including its version number.	Pytorch 2.0 ONNX 1.15.0
	Processing capabilities	Estimated capabilities for processing the model including the computational power such as the computational cost (in FLOPS), the computational complexity (in MAC operations). It also includes the temporary memory to store model parameters.	NPU 10TFLOPS, MEM 10GB

6.6.3 AI/ML model information for split AI/ML operations

AI/ML model information metadata for split AI/ML operations is used to describe the characteristics of AI/ML models for split inference service scenarios. This information may be used in a split point configuration to select a split point (from which a multiple may be predefined by the service provider for a certain AI/ML media service). A trained model can be represented as a directed acyclic graph model represented by a collection of nodes interconnected with edges (e.g. ONNX). A split point may happen before or after a graph node identified by its name or a number.

Table 6.6.3-1: AI/ML model information for split operations

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Split model information	**Split points**	The number of predefined split points at which a certain model can be divided into two for split inferencing.	2
Split point information	**Split point identifier**	An identifier of the split point in a description of a computing graph, may be generated by a neural network description language such as ONNX/NNEF. The identifier may be a name, a number, a combination thereof, a hash value. Identifiers must guarantee unique identification of a specific split point.	Nb:10, 75 Name: Layer_10,
	Split point intermediate data size	The size of the intermediate data resulting from the given split point, in kilobytes. Intermediate data size is typically dependent on the tensor(s) size at the given split point.	1086KB
	Split point number	The number of the split point where the split occurs. The number may belong to set of identified numbers defined at the configuration stage.	10
	Split point name	The name of the split point where the split occurs. The name may belong to set of identified split point names defined at the configuration stage.	conv2d_1234
	Split point flag	An information on whether to consider the split point before the split point identifier or after. The convention on whether it is before or after may be defined at the configuration stage.	before, after

6.6.4 Intermediate data information for split AI/ML operations

Intermediate data information identifies the structure of intermediate data output from a first endpoint that need to be retrieved to feed the inference of the second endpoint after transmission of the intermediate data over the network. A split point configuration may comprise whole or part of intermediate data information.

Table 6.6.4-1: Intermediate data information for split AI/ML operations

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Intermediate data general information	**Tensor structure framework**	The exact underlying tensor structure of the intermediate data tensors including the exact version of it.	PyTorch 2.0, TensorFlow v2.13.0, NumPy v1 .25
	Data direction	This defines the direction of transmitted data, either uplink (from UE endpoint to network endpoint) or downlink (From a network endpoint to the UE endpoint). This information may be useful to configure an intermediate data delivery session	Upstream, Downstream
	Global compression algorithm	Identifies a compression algorithm that can be applied to all the intermediate data tensors. For example, when the connectivity condition between the UE and the network is insufficient to transmit the original intermediate data, a compression algorithm may be applied.	NONE, FCM, SNAPPY, …
	Tensor identifier type	The type of the tensor identifier describing how the tensors are identified. This parameter is defined during the configuration stage.	String, Numerical value.
	Tensor mapping identifier	A unique identifier of a data model representation (e.g. list, table, structure) that maps tensor identifiers with original tensor names. This parameter is defined during the configuration stage.	TensorTable 12245, TensorList13
Intermediate data tensor information	**Tensor list**	List of Tensors that composed the intermediate data	[tensor1, tensor2, tensor3, tensor4]
	Tensor identifier	A unique identifier for the tensor. The identifier may be a name, an index of a tensor list or table, a combination thereof, a hash value.	Tensor1, 10
	Tensor shape	Tensor shape is a tuple of positive integers, where the size of the tuple represents the dimension of the tensor, and each value represents the size in each dimension.	[1,64,64,64].
	Tensor data type	The data type of each intermediate data tensor	Float32, int32
	Tensor compression algorithm	Identifies the compression algorithm(s) that can be applied to a particular tensor. The tensor compression algorithm supersedes the global compression algorithm when both are defined	NONE, FCM, SNAPPY, …

6.6.5 Service requirement information

Service requirement information metadata is used to describe the latency and processing requirements for the AI/ML media service. Such information may be used in the selection of an AI/ML model for the service, and/or the selection of a split point for a certain AI/ML model for split inferencing.

Table 6.6.5-1: Service requirement information

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Service requirement information	**Maximum service inference latency**	The maximum inference latency requirement specified for a given AI/ML media service, in milliseconds. In the case of split inferencing, this requirement includes the delivery latency of the intermediate data between the first and second split inference entities.	100ms
	Minimum service inference accuracy	The minimum accuracy specified for a given AI/ML media service.	80%
	Service type identifier	An identifier for the service type to be supported by the AI/ML model, such as ASR (Automatic Speech Recognition), TTS (Text To Speech), Translation (with the indication of input and output languages).	TTS, ASR, Trans-EN-to-ZH
	Service accuracy	The expected service accuracy	85%

6.6.6 Endpoint capability information

The endpoint capability information includes the capabilities of the endpoint (UE or network) for processing and transmitting the AI/ML model and intermediate data. Such information can be updated due to the change of the endpoint's workload or the network conditions. It can be used for the selection of AI/ML model, split inference, intermediate data compression, progressive model delivery.

Table 6.6.6-1: Endpoint capability information

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Endpoint capability information	**Processing capabilities**	The available resources for processing AI/ML model including the computational power (in FLOPS), the memory to store model parameters and perform the inference.	NPU 10TFLOPS, MEM 10GB
	Supported AI/ML Framework	The AI/ML framework(s) supported by the endpoint.	TensorFlow 2.0
	Supported compression algorithms	The supported compression algorithm(s) for intermediate data compression.	NONE, FCM, SNAPPY, …
	Connection capabilities	This indicates the available bandwidth in bit/s between the UE and the network for transmitting the AI/ML model and/or the intermediate data.	256 kb/s

6.6.7 Distributed/Federated learning information

This clause describes a set of possible control information for managing the training process, synchronization the training rounds, and defining the selection criteria for participating devices, or monitoring the convergence of the training process, in federated learning.

The following metadata categories are listed and further described in the table 6.6.7-1:

- Synchronization information: may be used to ensure that all devices start the training process simultaneously and progress at the same pace. For example, the server may send a synchronization information to all UEs to start a new round of training, as described in step 1 of figure 5.2.4-2.

- Device eligibility information: may be used to define the criteria for selecting the devices that will participate in the training process. For example, the server may send a device eligibility information to all devices that belong to the defined group by the application, as described in step 4 of figure 5.2.4-2.

- Model evaluation information: may be used to evaluate the performance (accuracy, precision…) of the global model for each device and make decisions about the training process. After running the learning phase, a device sends a model evaluation information to the server that measures the accuracy of the model. The server can then decide whether to continue training for another round or stop. Alternatively, this information may be used by the server to request the device to perform an evaluation of a newly downloaded global model, as described in step 7 of figure 5.2.4-2.

- Model update information: may be used to update the model parameters on the devices after each round of training as described in step 5 of figure 5.2.4-2. For example, the server may send a model update information to all devices to update the global model with the new model parameters. Model update information may also be used to update the global model on the server as described in step 15 of figure 5.2.4-2 with the new parameters updated by the local training on the device.

- Failure reporting information: may be used to handle unexpected errors or exceptions that may occur during the training process. For example, the server may send an error information to all devices to handle a device failure or network disruption as described in step 6 of figure 5.2.4-2. Alternatively, the AI Model training engine in the UE sends a failure information to the Federated learning engine in the server if a failure occurs as described in step 15 of figure 5.2.4-2.

Table 6.6.7-1: Federated learning information

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Synchronization information	**Round number**	indicates the training round in a model training	45
	Start time	indicates the start time of the training	"2025-02-16 12:00:00"
	Duration	indicates the desirable duration of the training. This value just shows an indication of the desirable time for completing the training round.	"2025-02-18 12:00:00"
Device eligibility information	**Group identifier**	used to assign a new id for the devices that meet the eligibility criteria of this information. If the device is eligible, it uses this value as one of its group ids and from now on, it reacts to information with the same group id.	"Group_A", "Group_B"
	Application group identifier	is assigned by the application on the device and if that value is equal to the value of this field, then the device is eligible	"Group_A", "Group_B"
	Hardware	parameter defining the hardware eligibility criteria for the device.	"OS ": "Linux", "processor_speed": "1.2 GHz", "memory": "8 GB"
	Location	parameter defining the location eligibility criteria for the device.	"Europe", "World"
	Language	parameter defining the language eligibility criteria for the device.	"English", "German"
	Data library identifier	defines the data library an eligible device is required to have.	"Library_123"
Model evaluation information	**Round number**	the round after which the evaluation is performed	32
	Metric number	shows the number of metrics included in this information body	2
	metric	the round after which the evaluation is performed	"Accuracy=85%" "Precision=92%"
Model evaluation information	**Model identifier**	Identifier of the model used to process the federated learning	"Model_AA"
	Model update parameters	Parameters includes the new model vector of values	[0.1, -0.2, 0.3, 0.4]
	New model identifier	The identifier of the new model when the server sends the updated model to one or more devices	"Model_AA-v2"
Failure reporting information	**Failure notice**	Description of the reason of failure	"Connection lost", "not enough memory"
	Timestamp	Time at which the failure is detected	"2025-02-16 14:52:00"

6.6.8 Compression metadata

6.6.8.1 Compression settings for a split point configuration

6.6.8.1.1 Introduction

The compression settings below identify:

- The candidate compression algorithms profiles to apply to intermediate data tensors.
- A split point compression characteristics (e.g. size reduction and performance metrics) for an association of compression algorithm profiles to intermediate data tensors.

A split point configuration may comprise whole or part of compression profile and associated characteristics.

6.6.8.1.2 Compression algorithm profiles

A list of compression algorithm profiles includes a description, a unique identifier and associated parameter supported for intermediate data compression. A subset of compression algorithms profile may be negotiated and exchanged between endpoints regarding different endpoints capabilities.

Table 6.6.8-1: Compression information

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Compression algorithm profile	**Compression algorithm profile list**	List of compression algorithm profiles.	
	Compression algorithm profile description	Identifies the compression algorithm(s) that can be applied to the intermediate data tensors.	NONE, Quantization, FC_VCM, SNAPPY Neural Network Coding
	Compression algorithm profile level identifier	Identifies the profile level of the compression algorithm.	FCM high 5.1, FCM main 5.3, FCM 6.4 NNC xxx 5.7.9, NNC yyy 5.8, NNC yyy 6.4
	Compression algorithm profile parameter set	List of compression parameters of the selected compression algorithm that fulfil the compression profile.	Param 1 Param 2 e.g. Qp Quantization Parameter = -15

6.6.8.1.3 Intermediate data tensors and associated compression profile and characteristics

This identifies information for associating individual or group of compression profiles to intermediate data tensors for a split point configuration. This includes split point information, expected split point compression characteristics associated to a compression profile.

Table 6.6.8-2: Intermediate data tensors and associated compression profile and characteristics

Metadata category	Metadata type	Definition	Metadata type description (Examples)
Split point information	**Split point identifier**	Key identifier of the split point clause §6.6.3 to associate value data below	Nb:10, 75 Name: Layer_10,
Intermediate data general information	**Tensor identifier type**	The type of the tensor identifier describing how the tensors are identified. This parameter is defined during the configuration stage.	String, Numerical value.
	Tensor mapping identifier	A unique identifier of a data model representation (e.g. list, table, structure) that maps tensor identifiers with original tensor names. This parameter is defined during the configuration stage.	TensorTable12245, TensorList13
split point compression characteristics	**Compressed intermediate data size**	The compressed intermediate data size. If no compression, the size is the baseline data size.	12 Mbytes
	Compressed intermediate data size ratio	The ratio gives an indication on how much the data has been reduced after apply compression. Example of representation of ratios: $compression\ ratio\ a) = \frac{\text{uncompressed intermediate size}}{\text{compressed intermediate size}}$ $compression\ ratio\ b) = \frac{uncompressed\ intermediate\ size - compressed\ intermediate\ size}{uncompressed\ intermediate\ size}$ $where\ compression\ ratio\ percentage = compression\ ratio\ b) * 100$	ratio a) 5:1 or 5/1 means the size has been reduced from a factor 5 ratio b) 0,8 means a reduction of 80% of the baseline size
	Compression performance metric value	The measured performance metric value depending on the performance metrics used, e.g. Map score, F1 score, accuracy.	F1 Score, Map score
	Compression performance metric ratio	The ratio is calculated from the metric value obtained using compression divided by the value obtained without compression. for example, it may indicate how much accuracy has been affected	Ratio 0.9 or 90% means a reduction performance accuracy of 10%
Intermediate data tensors associated to	**Tensor list**	List of tensors or groups of tensors that composed intermediate data	e.g. list of ONNX tensor names Tensor1, Tensor2

compression profile		**Tensor group compression name**	This identifies a group of tensors when each group is associated with a common compression profile.	Group 1, Group 2
		Tensor group compression type	This identifies the type for all tensors belonging to tensor group.	Float32
		Tensor compression granularity	This indicates if the compression profiles are applied to each tensor one by one (keyword is “tensor”), or to all the tensors of this group (keyword is “global”).	“tensor”, “global”
		Tensor identifier	A unique identifier for the tensor. The identifier may be a name, an index of a tensor list or table, a combination thereof, a hash value.	Tensor1, 10
		Tensor shape	Tensor shape output. The output tensor shape may be different from input and uncompressed tensor shape or may be transposed	[1,64,64,64].
		Tensor data type	The type of the tensor	Int32, Float32,
		Tensor compression algorithm profile identifier	Identifies the selected compression algorithm profile	FCM high 5.1, FCM main 5.3, FCM 6.4 NNC xxx 5.7.9, NNC yyy 5.8, NNC yyy 6.4

6.7 Existing optimization and compression tools for AI/ML models

6.7.1 AIMET library

The AI/ML Model Efficiency Toolkit (AIMET) is a library that provides advanced model quantization and compression techniques for trained neural network models. The library focuses on unilateral (sender-only) techniques that do not require any decoding on the receiver side. Figure 6.7.1-1 depicts the concept of the AIMET library.

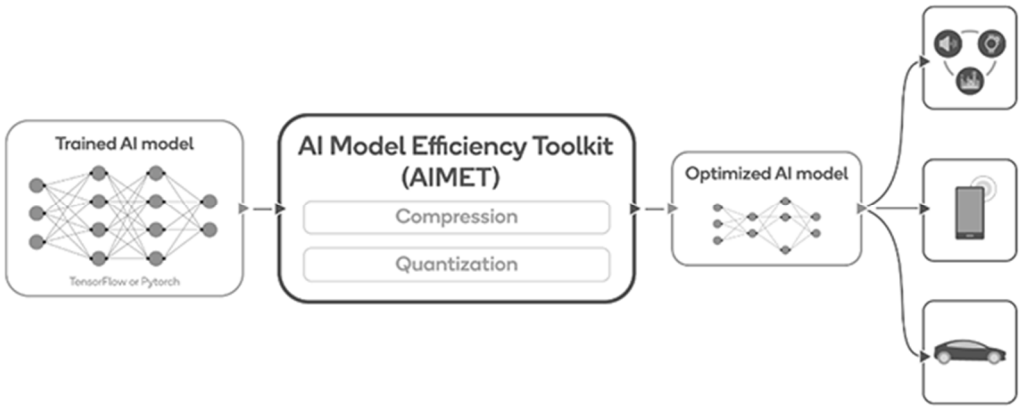


Figure 6.7.1-1: Concept of the AIMET library

The library is designed to work with trained PyTorch and Tensorflow/Keras models and can automate the optimization without significant loss in accuracy. The library supports advanced quantization and compression techniques that contribute to faster inference and lower memory footprint.

The following python code shows how the library may be used to compress a trained DNN:

```
from aimet_torch.compress import ModelCompressor
ssvd_compressed_model, ssvd_comp_stats =
ModelCompressor.compress_model(model=model,
       eval_callback=eval_callback,
       eval_iterations=1,
       input_shape=(1, 3, 224, 224),
       compress_scheme=CompressionScheme.spatial_svd,
       cost_metric=CostMetric.mac,
       parameters=params)
print(ssvd_comp_stats)
```

The source code may be found in [26].

6.7.2 MPEG NNC

Among the available tools for compression of neural networks the NNCodec [27] (neural network encoder/decoder) is publicly available and conforms to the MPEG NNC standard [17].

NNC use cases and validation results provided by MPEG.

Table 6.7.2-1 shows examples of AI/ML model distribution using MPEG's NNC standard. These results have been reported by MPEG for the verification of NNC in different use cases [28]. Compression rates are given in percent of the original models at working points that have approximately the same performance as the original models (transparent performance). More detailed information on the different applications can be found in the documents referenced in [28].

Table 6.7.2-1: Application and verification of NNC in different use cases as reported by MPEG.

Application	Model / layer types	Datasets	Metrics	Codec	Compression rate at transparent performance
Image super-resolution	SWINv2 (Vision transformers)	DIV2K	PSNR, SSIM, LPIPS	NNCodec	9-15%
Image super-resolution	EDSR (2D convolutions)	DIV2K	PSNR, SSIM, LPIPS	NNCodec	15%
Image restoration	NAFNet (2D convolutions)	GoPro	PSNR	NNCodec	18%
Learned quality metric (LPIPS)	AlexNet backbone (2D convolutions, fully connected)	DIV2K	LPIPS score	NNCodec	9%
Image Compression	Autoencoder, 2D convolutions	CIFAR100	PSNR, SSIM	NCTM	17%
INVR (NERFs)	DyNERF, MixVoxels	CBABasketball, Mirror	PSNR	NCTM	10-20%
Point cloud compression	GRASP-Net (3D convolutions)	MPEG test sequences	D1/D2 PSNR	NNCodec	20%
Visual object classification	VGG16, ResNet50, MobileNet v2 (2D convolutions, pooling, batch-normalisation, fully connected)	ImageNet	top-1, top-5	NCTM	3-12%
Visual object classification	SWIN (vision transformers)	MS COCO	top-1	NNCodec	10-12%
Object detection	SWIN (vision transformers)	ImageNet1K	mAP	NNCodec	16%
Object detection	Yolo v3 (2D convolutions, pooling, batch-normalisation, fully connected)	MS COCO	F1	NCTM	10%
Acoustic scene classification	convolutions, fully connected	DCASE 2017 Task1	classification accuracy	NCTM	4%

Recommender system	Custom (feature embedding, fully connected)	MovieLens	top-100	NNCodec	2-4%
Adaptive bitrate selection using reinforcement learning	Pensieve (convolutions, fully connected)	Pensive-Pytorch	average reward	NCTM	20%
NLP	BERT (transformer encoders)	SQuAD	F1	NCTM	15%

In summary, MPEG reports that some models may be compressed to 2% to 20% at transparent performance. According to [28], even greater bit rate reductions are possible when tolerating small performance reductions as a trade-off.

Table 6.7.2-2 shows MPEG's coding results for the distribution of incremental AI/ML model updates that are reported in MPEG's whitepaper [29]. MPEG's results are related to different models for image recognition. In the *federated learning* use cases, the model is trained from scratch. In the *federated transfer learning* use cases the model is re-trained for a new dataset.

In the MPEG framework, training is performed in multiple rounds using a server and 16 clients. Each training round consists of the following steps:

- Each client performs a local training step based on its local data and its local model. It computes a local model update, which is the difference of the local model before and after the local training step. Then, it sends the local model update to the server.
- The server averages all received local model updates and sends the averaged update to the clients.
- Each client adds the received averaged update to its local model as it was before the local training step.

This way, all clients start the next training round with the same local model.

The table 6.7.2-2shows MPEG's results in an *uncompressed communication* scenario and a *compressed communication* scenario.

The *uncompressed communication* scenario is given for reference. The table shows the top-1 accuracy that is achieved after training with the number of training rounds that is also given in the table. Moreover, the table shows the accumulated number of GB that are transmitted for sending the local and averaged model updates without any kind of compression.

In the *compressed communication* scenario, local and averaged model updates are lossy compressed with NNC. The number of training rounds has been selected to approximately match the accuracy achieved in the *uncompressed communication* scenario. The table shows that MPEG reports a reduction of the transmitted data volume to 3% to 0.1% (of the uncompressed data volume) when using compression with NNC, while the accuracy is not reduced by more than 1%.

Table 6.7.2-2: Application of NNC in different federated learning use cases as reported by MPEG.

Model	Datasets	Uncompressed communication			Compressed communication		
		Training Rounds	Top-1 accuracy in %	Transmitted data in GB	Training Rounds	Top-1 accuracy in %	Transmitted data in % of uncompressed
Federated Learning							
ResNet56	CIFAR-100	99	57.79	7.86	103	57.49	2.87
ResNet50	ImageNet-200	75	53.00	229.61	89	52.58	2.08
MobileNetV2	ImageNet-200	97	45.44	30.79	94	45.33	3.07
Transfer-Federated Learning							
ResNet18	ImageNet-1k to Pascal VOC	49	72.61	70.16	46	72.13	0.64
ViT-B/16	ImageNet-1k to Pascal VOC	20	78.54	219.68	46	77.98	0.11

More detailed information on the different scenarios can be found in the documents referenced in [29].

6.8 User plane metadata

6.8.1 User-plane metadata for split inferencing

Metadata for split operations described in clause 6.6 may be negotiated between two endpoints at the negotiation stage before the inference loop starts. However, metadata may also need to be carried out during the inference stage with intermediate data. The reasons why the identified user-plane metadata information need to be transmitted with intermediate data payload are listed below:

- A multi-branch split operation produces several tensors. Therefore, at the inference stage, the intermediate data payload contains multiple tensors. Metadata information must be transmitted to identify each tensor data in the intermediate data payload at the receiver endpoint:
  - Tensor identifiers
  - Tensor type
- Some models may be designed to infer media data with variable size. Splitting these models may produce intermediate tensor data of varying shapes. In this case, the size of the intermediate tensors can only be calculated at inference stage. Tensor data size characteristics need to be transmitted with the tensor data:
  - Tensor shape
  - Tensor type
- When a compression algorithm is reselected, for example to modify the compression ratio of the intermediate data or to increase the prediction quality, information on the (re)selected compression algorithm and related tensor information may need to be transmitted with tensor data:
  - Tensor identifier
  - Tensor shape
  - Tensor type
  - Tensor compression algorithm profile identifier

The following table 6.8-1 summarizes the user-plane metadata that may need to be carried with intermediate data payload as described above.

Table 6.8-1: User-plane metadata.

Metadata category	Metadata type	Definition	Examples
User-plane metadata	**Tensor list**	**List of Tensors that composed the intermediate data**	
	Tensor identifier	**A unique identifier for the tensor. The identifier may be a name, an index of a tensor list or table, a combination thereof, a hash value.**	Tensor1 10
	Tensor shape	**Tensor shape is a tuple of positive integers, where the size of the tuple represents the dimension of the tensor, and each value represents the size in each dimension.**	[1,64,64,64].
	Tensor data type	**The data type of each intermediate data tensor**	Float32, int32
	Tensor compression algorithm profile identifier	**Identifies the selected compression algorithm profile**	FCM high 5.1, FCM main 5.3, FCM 6.4 NNC xxx 5.7.9, NNC yyy 5.8, NNC yyy 6.4
Intermediate data payload	**Tensor data**		[]

The following table 6.8-2 shows an example of user-plane data, including metadata applied to an object detection inference of the *ssd_resnet* model for the different scenarios:

- Multi-branch split at node 30 where each tensor must be identified among the intermediate data tensors (Relu_output_0, Conv_output_0),
- Tensor shape changing at runtime, with a variable image dimension feeding the second part of the split inference (e.g. [1,256,75,75])
- Reselection of a compression profile settings applied for the same or different split point (e.g. Conv_output_0 with a new quantized parameter qp=-14).

Table 6.8-2: User-plane metadata example.

data category	data type	Data example
Intermediate data tensor	`Intermediate data tensor element list`	
	`Tensor identifier`	`Relu_output_0`
	`Tensor shape`	`[1,256,75,75]`
	`Tensor data type`	`float32`
	`Tensor compression profile Identifier`	`nnc.xx (with qp=-14)`
	`Tensor data`	`[]`
	`Tensor identifier`	`Conv_output_0`
	`Tensor shape`	`[1,512,38,38]`
	`Tensor data type`	`float32`
	`Tensor compression profile Identifier`	`nnc.xx with qp=-14`
	`Tensor data`	`[]`

7 Conclusion

AI/ML in media services involve the use of AI/ML models to perform media processing. The AI/ML models used to process media, typically involving video, audio as input. The output can be an enhanced version of the media, such as improved picture quality or translated audio, a detailed description of the media itself, like object recognition labels, or entirely new media, such as converting text or sign language into speech or video. In order to support such AI/ML based media processing, three common scenarios have been studied in the context of different use cases and architecture:

- UE device AI inferencing
- AI inferencing in the network
- Split AI inferencing between the UE and the network.

The following media related use cases were considered:

- Object detection, see clause 4.2.2
- Video quality enhancement in streaming, see clause 4.2.3
- Crowd source media capture, see clause 4.2.4
- NLP on speech, see clause 4.2.5

In this study, the broad findings for AI/ML model transfer in TR 22.874 [2] have been further analyzed with specific focus on media-based AI/ML use cases and scenarios. In particular:

- Model delivery in the 5G system and IMS for media related use cases.
- Model split operation in the 5G system and IMS for media related use cases.
- Federated learning in the 5G system and IMS for media related use cases.

This document describes how AI/ML models and data may be distributed over the 5G system and or IMS and documents the split AI/ML operations between different AI/ML endpoints (noticeably the UE and the network). This includes some study of compression of AI/ML model data and intermediate data due to the broad range of applications and AI/ML models feasibly evaluations for a given set of scenarios are documented in TR 26.847 [20] as part of this study.

Functional architectures are presented for:

- Basic AI/ML model distribution
- Split AI/ML operation
- Distributed/federated learning

Different AI user plane data components have been identified and documented (AI model data, intermediate data, inference input and output data), and a set of logical AI functions have been defined.

The identified logical AI functions are further mapped to the 5G system, addressing the underlying 5GMS/RTC and IMS DC architectures including respective procedures. This mapping outlines how AI media use cases integrate with various architectures and procedures, detailing the provisioning, capability discovery/negotiation, and session support for delivering AI data components. It also explains the use of AI media functions at different endpoints based on the negotiated service configuration. Additionally, three collaboration scenarios are studied, each offering varying levels of MNO network support for AI/ML functions. Clause 6 details data formats for AI components that can possibly be used to implement the different architecture.

Regarding inference output media data and its related the traffic characteristics and media interoperability aspects while potentially relevant aspects to study, these have not been covered in this study since the focus was primarily on the AI/ML inference process itself.

Based on the details in the report, the following next steps are identified:

Normative work in release-20:

For collaboration scenario 3 IMS services:

- Recommend stage 3 normative work on the support of AI/ML model distribution and operation in IMS.
- Extend TS 26.114 and TS 26.264 specifications to support AI/ML data delivery in IMS services, as identified in clause 5.4.
- Extend TS 26.114 and TS 26.264 specifications to support AI/ML media processing in IMS services, as identified in clause 5.5.
- Specify support for AI/ML data signalling and negotiation, including support for split inferencing, as identified in clause 6.6.
- Select interoperable formats for AI/ML model data as identified in clause 6.2 and intermediate data as identified in clause 6.3.
- Define the support of the configuration, delivery, compression, and processing of AI/ML data in a new specification, as needed and identified in clause 6, along with relevant optional metadata as identified in clause 6.6.
- Collaborate with SA2 on related matter where necessary.

New Study in Release 20 or beyond:

For collaboration scenarios 1 (OTT) and 2 (Hosting):

- Further study, identify and document the traffic characteristics of the AI/ML data components (as defined in clause 5.3.1 and detailed in clause 6) for the relevant use cases, as introduced in TR 26.847.
- Further study and identify any potential needs for new QoS identifiers, metrics and/or QoS procedures to support the delivery of the AI/ML data components based on the architectures in TS 26.501, TS 26.506, and TS 26.114 for 5GMS, RTC, and IMS respectively.

For collaboration scenario 3 non-IMS services:

- Further study and investigate stage 2 aspects for the architectures in TS 26.501 (5GMS) and TS 26.506 (RTC), identifying potential key issues related to:
- The support of AI/ML model distribution and operation, based on details in clause 5.3.6.
- The support of split AI/ML inferencing between the UE and the network, based on details in clause 5.3.5.
- The support of distributed/federated learning, in particular SA2 defined features, as identified in clause 5.3.7.
- Collaborate with SA2 on related matter where necessary for Release 20 or beyond.

Annex A:
Collaboration scenarios

A.1 Introduction

Collaboration scenarios enable multiple levels of network support for AI/ML for Media. The description of these scenarios is meant to develop the appropriate architecture mappings based on them.

A.2 Scenarios description

A.2.1 Collaboration Scenario 1: AI/ML OTT

In this collaboration scenario, the AI/ML service is offered completely over-the-top. The service provider deploys its own application servers in the cloud and offers a mobile application to its end-users. The MNO may provide assistance to these sessions by allocating the appropriate traffic handling for the identified application streams, including QoS allocation to meet the application's requirements.

A.2.2 Collaboration Scenario 2: AI/ML hosting

In this collaboration scenario, the MNO offers CDN operations to perform large scale distribution of AI/ML data, such as trained models, model updates, etc.

The MNO is responsible for ensuring the right AI/ML data is delivered to the end device at the right time. The service is required to identify the capabilities of the receiving device and match it to the distributed AI/ML data, e.g. a UE that only supports 8-bit quantized models in TensorFlow Lite will receive the matching variant of the ML model.

A.2.3 Collaboration Scenario 3: MNO-operated AI/ML services

In this collaboration scenario, the MNO is offering the AI/ML-based service and is responsible for distributing the AI/ML model, updates and the corresponding data. The AI/ML component may be part of one of the operator services, such as the multimedia telephony service (MTSI). It may also be a background service, e.g. one that optimizes the network usage by sharing traffic data and performing federated learning. Another possibility is that the AI/ML component is part of a completely new MNO-offered service, e.g. to support IoT, autonomous driving, or cloud media processing and storage.

A.3 Usage of collaboration scenarios

Identifying the collaboration scenarios is crucial to organize any future normative work on AI/ML. The work may be structured based on these collaboration scenarios. Prioritizing these collaborations scenarios based on market relevance and complexity may be used to guide the potential consequent normative work.

Note: Collaboration scenario 3 needs to be further broken down to understand the relevance and exact requirements of each use case.

A.4 Architecture variants for collaboration scenarios

A.4.1 General

This clause addresses the derivative architecture for each of the collaboration scenarios. The three collaboration scenarios are summarized below and further details is specified in A.2.

The three collaboration scenarios are specified based on the location of the required General Media and Logical AI/ML Functions in the trusted domain, defined as follows:

- Collaboration scenario 1: AI/ML OTT
- Collaboration scenario 2: AI/ML Hosting
- Collaboration scenario 3: MNO-operated AI/ML Services

The list of key functions in the trusted domain differs for each collaboration scenario as described in table A.4.1-1

Table A.4.1-1: Mapping of functions to each collaboration scenario

General Media Architecture Functions	Logical AI/ML Functions	Collaboration scenario 1	Collaboration scenario 2	Collaboration scenario 3
Media AF	~	Required	Required	Required
Media AS	AI/ML Data Access/Delivery	Optional	Required	Required
	AI/ML Inference Engine	N/A	N/A	Optional
	Federated Learning Engine	N/A	N/A	Optional
Media Client	~	Required	Required	Required
Media Session Handler	~	Required	Required	Required
Media Access Function	AI/ML Data Access/Delivery	Required	Required	Required
	AI/ML Inference Engine	Required	Required	Required
	Federated Learning Engine	Optional	Optional	Optional
Media Application Provider	AI/ML Enabled Application Provider	Required	Required	Required
Media-aware Application	AI-aware Application	Required	Required	Required
NOTE: Depending on the specific use case for each collaboration scenario, RTC or 5GMS specific functions may also be relevant.				

A.4.2 Collaboration scenario 1 architecture

Figure A.4.2-1 shows the architecture variant for collaboration scenario 1 when the AI/ML service, and delivery session is completely running over the top. For this case, many of general delivery function entities are not in scope. However, the Media AF is present in the trusted domain to support QoS allocation, bitrate recommendations, and QoE report collection in order to meet the application's requirements.

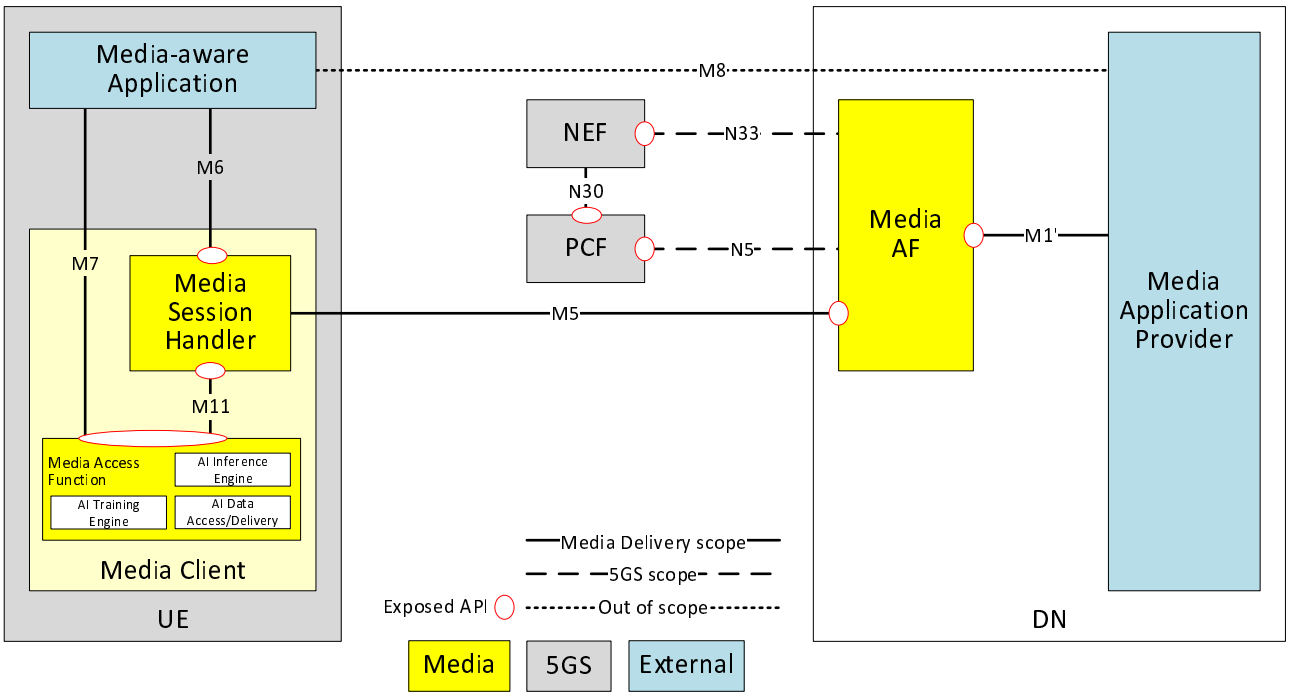


Figure A.4.2-1: Derivative AI/ML data delivery architecture for collaboration scenario 1

A.4.3 Collaboration scenario 2 architecture

Figure A.4.3-1 shows the architecture variant for collaboration scenario 2 when the MNO offers CDN operations and the corresponding functions to perform large scale distribution of AI/ML model data, such as trained models, model updates etc. The MNO is also responsible for ensuring that the right AI/ML model data is delivery to the UE at the right time, and according to the capabilities of the UE. Depending on the use case scenario, existing 5GMS AF or RTC AF functions may be used.

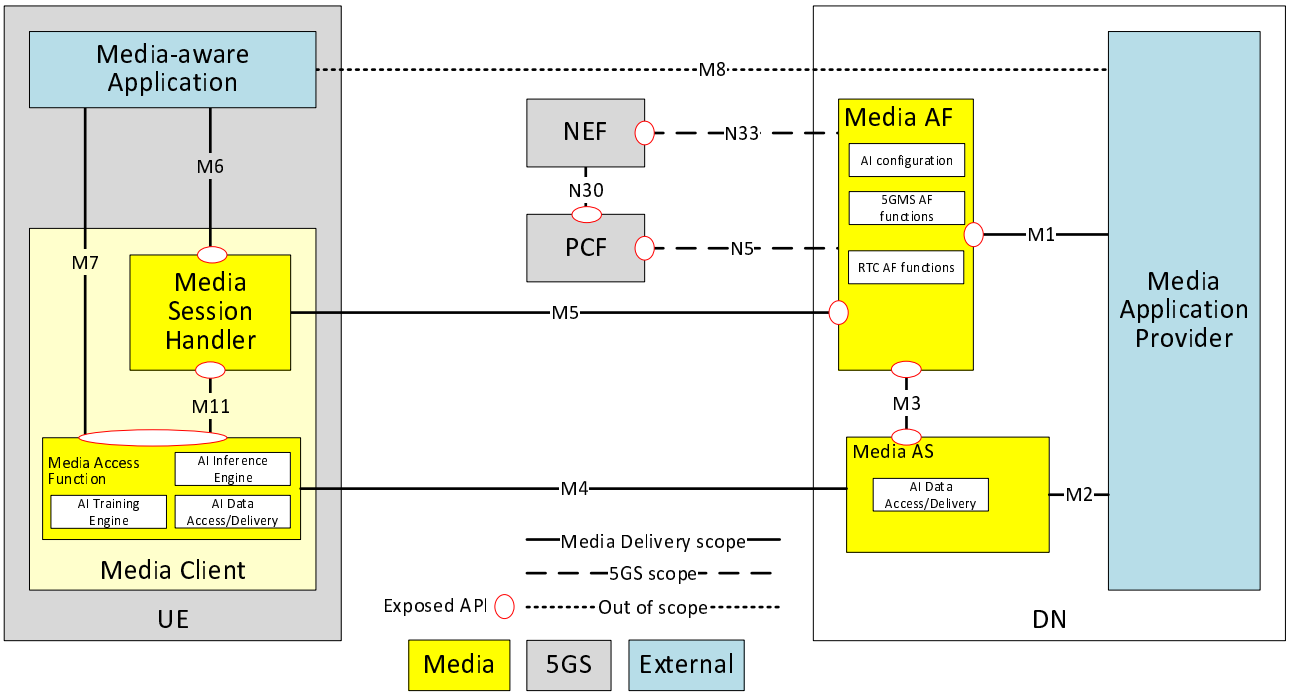


Figure A.4.3-1: Derivative AI/ML data delivery architecture for collaboration scenario 2

A.4.4 Collaboration scenario 3 architecture

Figure A.4.4-1 shows the architecture variant for collaboration scenario 3 when the MNO is offering a fully operated AI/ML media service. Depending on the use case scenario, the AI/ML component may be part of one of the existing operator services (e.g. MTSI), or it may be a background service (e.g. AI/ML based data collection or federated learning), or it may be an offering of a new AI/ML based MNO-offered service (e.g. use cases which support network based AI/ML inferencing or related processing and storage).

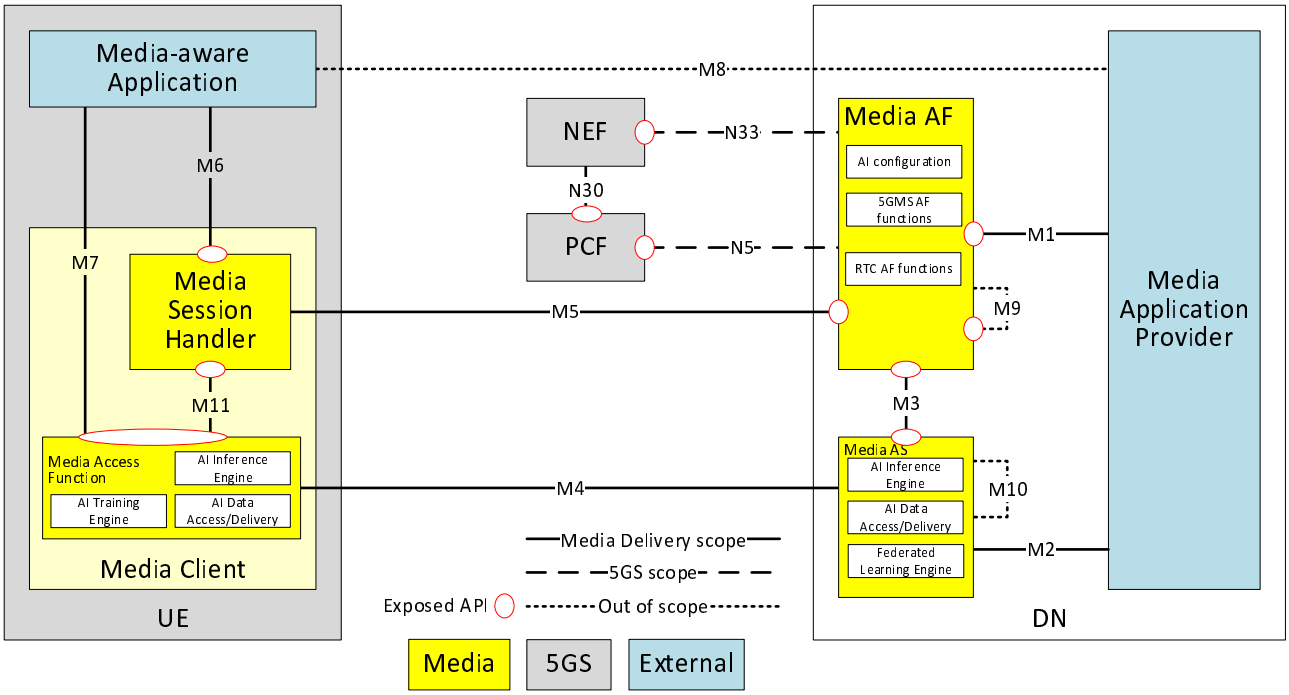


Figure A.4.4-1: Derivative AI/ML data delivery architecture for collaboration scenario 3

A.5 AI/ML collaboration scenarios

A.5.1 Relevance of use cases to collaboration scenarios

Depending on the service configuration, each of the scenarios under the use cases in clause 4.2 may fall under one or more of the collaboration scenarios defined in this annex.

The relevance of each use case scenario to the collaboration scenarios are described in detail in this clause.

NOTE: The list below is focused on cases with relevance to the scope of this study and is not exhaustive. Certain use case scenarios may also be realised through other collaboration scenarios, albeit with less relevance to the scope of this study.

Object recognition in image and video:

UE inference only: collaboration scenarios: 1, 2

CS 1: the AI/ML model is delivered to the UE over the top using service provider deployed application servers.

CS 2: the MNO offers CDN operations to deliver the AI/ML model to the UE according to its capabilities and requirements.

Network inference only: collaboration scenario 3

CS 3: the MNO offers an AI/ML service which supports the complete offloading of AI/ML inferencing in the network.

Split inference

CS 3: the MNO offers an AI/ML service which supports the delivery of the appropriate (partial) AI/ML model to the UE, together with the configuration and support of split inferencing between the UE and the network.

Video quality enhancement in streaming:

End-to-end neural network-based video coding: both sender and receiver apply part of a DNN model

CS 3: the MNO offers an AI/ML service which supports the delivery of the appropriate AI/ML model to the UE, together with the configuration and support of AI/ML inferencing in the network (typically the sender).

Neural network based post-processing for video coding: receiver AI/ML post-processing only

CS 1: the AI/ML model is delivered to the UE over the top using service provider deployed application servers.

CS 2: the MNO offers CDN operations to deliver the AI/ML model to the UE according to its capabilities

Crowd-sourcing media capture:

Device inference

CS 2: the MNO offers CDN operations to deliver the AI/ML model to the UE according to its capabilities and overall service configuration.

Network inference

CS 3: the MNO offers a complete crowd-sourcing media capture service where media captured on multiple devices are aggregated for AI/ML inferencing in the network.

Natural Language Processing (NLP) on speech:

Network based distributed/federated learning

CS 1: the required data for the service (partially trained AI/ML model, training data etc.) is delivered to and from the UE over the top using service provider deployed application servers.

CS 3: the MNO offers a new service providing network based distributed/federated learning for this use case.

Annex B:
Change history

Change history							
Date	**Meeting**	**TDoc**	**CR**	**Rev**	**Cat**	**Subject/Comment**	**New version**
2022-01	SA4#118 e	S4-220498				Agreements after SA4#118e (S4-220391: TR skeleton)	0.1.0
2022-11	SA4#121	S4-221376				Inclusion of use cases	0.2.0
2023-02	SA4#122	S4-230378				Introduction of split models and configurations (S4-230401)	0.3.0
2023-02	SA4#122	S4-230405				Update of this Change history table	0.3.1
2023-06	SA4#124	S4-231043				Workflow and procedures (S4-230830)	0.4.0
2023-11	SA4#126	S4-231923				Model Data (S4-231885), formats (S4-231772), frameworks (S4-231884), Federated learning (S4-231886), architecture (S4-231959)	0.5.0
2024-01	SA4#127	S4-240421				Media delivery architecture (S4-240209), model data (S4-240247), compression tools (S4-240271), Metadata (S4-240436), Adaptive model workflow (S4-240449)	0.6.0
2024-04	SA4#127-bis-e	S4-240799				Procedure for AIML distribution and operation (S4-240649), existing formats (S4-240646), general architecture (S4-240782), basic workflow for split inferencing (S4-240783), frameworks update (S4-2400ZZ), existing frameworks update (S4-240784)	0.7.0
2024-05	S4#128	S4-241323				Architecture updates (S4-241027), AI-related work (S4-241327), Federated learning procedures (S4-241329), Split inferencing procedures update (S4-241330), split operations (S4-241301), Crowdsourcing use case (S4-241294), intermediate data compression approaches (S4-241331)	0.8.0
2024-08	SA4#129-e	S4-241597				Annex on collaboration scenarios (S4aV240029)	0.8.1
2024-08	SA4#129-e	S4-241662				Editorial: missing refs, architectures for collaboration scenarios (S4-241557), RTC scenarios (S4-241647), compression metadata (S4-241667), mapping to IMS (S4-241715, S4-241689), MPEG FCM update (S4-241725), update on metadata for split operations (S4-241729), update on split AIML procedures (S4-241742), collaboration scenarios and use cases (S4-241740)	0.9.0
2024-11	SA4#130	S4-241950				TR structure (S4-241950), Definitions (S4-211955), Metadata (S4-241981), Data components (S4-241954), formats and compression (S4-242178), Media Data (S4-242212), user plane data (S4-242181), conclusions (S4-242218)	0.10.0
2025-02	SA4#131	S4-250335				Editorial clean-up. (S4-231884 was partly implemented) Abbreviations, language consistency, references numbering, typos, styles fixed. (S4-250201,S4-250283), Federate learning metadata (S4-250206), tasks discovery over IMS (S4-250352), conclusions (S4-250403)	0.11.0
2025-03	SP#107					Version 1.0.0 created by MCC for presentation to TSG SA	1.0.0
2025-04	SA4#131-bis-e	S4-250624				Related work (S4aV250024)	1.0.1
2025-04	SA4#131-bis-e	S4-250676				Various edits (S4-250504), conclusions (S4-250649, S4-250711)	1.1.0
2025-05	SA4#132	S4-250855				Editorial improvements	1.1.1
2025-05	SA4#132	S4-251089				Conclusions (S4-251128, S4-251050), definitions (S4-251130)	1.2.0
2025-05	SA#108	SP-250639				Version 2.0.0 created by MCC to send to TSG SA for approval	2.0.0
2025-06	SA#108					Version 19.0.0 created by MCC upon approval in SA#108	19.0.0

History

Document history		
V19.0.0	January 2026	Publication