|  |  |
| --- | --- |
| 3GPP TR 26.927 V0.7.0 (2024-04) | |
| Technical Report | |
| 3rd Generation Partnership Project;  Technical Specification Group Services and System Aspects;  Study on Artificial Intelligence and Machine Learning in 5G media services;  (Release 18) | |
|  | |
|  |  |

The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP.  
The present document has not been subject to any approval process by the 3GPPOrganizational Partners and shall not be implemented.  
This Report is provided for future development work within 3GPPonly. The Organizational Partners accept no liability for any use of this Specification.  
Specifications and Reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organizational Partners' Publications Offices.

|  |
| --- |
|  |
| ***3GPP***  Postal address  3GPP support office address  650 Route des Lucioles - Sophia Antipolis  Valbonne - FRANCE  Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16  Internet  http://www.3gpp.org |
| ***Copyright Notification***  No part may be reproduced except as authorized by written permission. The copyright and the foregoing restriction extend to reproduction in all media.  © 2023, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).  All rights reserved.  UMTS™ is a Trade Mark of ETSI registered for the benefit of its members  3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners LTE™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners  GSM® and the GSM logo are registered and owned by the GSM Association |

Contents

Foreword 6

Introduction 7

1 Scope 8

2 References 8

3 Definitions of terms, symbols and abbreviations 8

3.1 Terms 8

3.2 Symbols 9

3.3 Abbreviations 9

4 Introduction to AI/ML for media 9

4.1 General 9

4.2 Media-based AI/ML use cases and scenarios 9

4.2.1 Introduction 9

4.2.2 Object recognition in image and video 10

4.2.3 Video quality enhancement in streaming 11

4.2.3.1 Sender-receiver approaches 11

4.2.3.1.1 End-to-End neural network-based video coding 11

4.2.3.1.2 Neural network based post-processing for video coding 11

4.2.4 Crowd-sourcing media capture 12

4.2.4.1 Introduction 12

4.2.4.2 Device inference 12

4.2.4.3 Network inference 13

4.2.5 Natural Language Processing (NLP) on speech 13

4.3 Related work 13

5 Media service architecture for AI/ML 13

5.1 AI/ML Split configurations 13

5.1.1 AI/ML model composition 13

5.1.2 Topologies of split AI/ML inference 15

5.1.2.1 Introduction 15

5.1.2.2 UE as the media source 15

5.1.2.3 Network as the media source 16

5.2 Architectures and service flows 16

5.2.1 Introduction 16

5.2.2 Complete/basic AI/ML model distribution 17

5.2.2.1 Basic architectures 17

5.2.2.2 Basic workflows 18

5.2.2.2.1 Generic model delivery 18

5.2.2.2.2 Adapative model delivery 18

5.2.3 Split AI/ML operation 20

5.2.3.1 Basic architectures 20

5.2.3.2 Basic workflows 22

5.2.4 Distributed/federated learning 25

5.2.4.1 Basic architecture 25

5.2.4.2 Basic workflows 25

5.3 Architecture for AI data delivery 27

5.3.1 AI data components 27

5.3.2 Media-related AI data logical functions 28

5.3.3 Mapping AI functions to the generalized 5G media delivery architecture 28

5.3.4 Architecture and components for AI data delivery over 5G 28

5.3.4.1 Introduction 28

5.3.4.2 Network functions and UE entities 29

5.3.5 Procedure for Split AI/ML operation 30

5.3.6 Procedure for AI/ML model distribution and operation 32

6 Data components for AI/ML-based media services 33

6.1 General 33

6.2 Model data 33

6.2.1 Model optimization techniques 33

6.2.2 Model update requirements and constraints 34

6.2.2.1 Evolving requirements and environment conditions after model selection 34

6.2.2.2 Model accuracy deviation between the training phase and the delivery phase. 35

6.2.2.3 Applying inference on evolving characteristics of the input media content 35

6.2.3 Model serialization 35

6.2.4 Classes of AI/ML models 35

6.2.4.1 Introduction 35

6.2.4.2 Supervised learning 36

6.2.4.3 Unsupervised learning 36

6.2.4.4 Reinforcement learning 36

6.2.5 Existing formats for AI/ML models 36

6.2.5.1 ONNX format 36

6.2.5.2 NNEF format 38

6.2.5.3 Neural Network Coding (NNC) format 40

6.2.5.4 PyTorch formats for model distribution 41

6.3 Intermediate data 41

6.3.1 Introduction 41

6.3.2 Intermediate data size delivery 41

6.4 Existing frameworks for AI/ML 42

6.4.1 TensorFlow 42

6.4.1.1 Introduction 42

6.4.1.2 Tensor 42

6.4.1.3 Usage of TensorFlow 42

6.4.2 PyTorch 43

6.4.2.1 Introduction 43

6.4.2.2 Main differences with TensorFlow 43

6.4.2.2.1 Computational graph 43

6.4.2.2.2 Ease of use 43

6.4.2.2.3 Visualization 43

6.4.2.2.4 Ecosystem 43

6.4.2.2.5 Research 43

6.5 Media data 43

6.6 Metadata 43

6.6.1 Introduction 44

6.6.2 Common AI model information 44

6.6.3 AI model information for split AI/ML operations 44

6.6.3 Intermediate data information for split AI/ML operations 45

6.6.4 Service requirement information 46

6.6.5 Endpoint capability information 46

6.6.6 Distributed/Federated learning information 47

6.6.6.1 Control information 47

6.6.6.1.2 General 47

6.6.6.2 Synchronization information 47

6.6.6.2.1 Definition 47

6.6.6.2.2 Behavior 47

6.6.6.2.3 Parameters 47

6.6.6.3 Device eligibility information 48

6.6.6.3.1 Definition 48

6.6.6.3.2 Behavior 48

6.6.6.3.3 Parameters 48

6.6.6.4 Model evaluation information 48

6.6.6.4.1 Definition 48

6.6.6.4.2 Behavior 48

6.6.6.4.3 Parameters 48

6.6.6.5 Model update information 49

6.6.6.5.1 Definition 49

6.6.6.5.2 Behavior 49

6.6.6.5.3 Parameters 49

6.6.6.6 Failure reporting information 49

6.6.6.6.1 Definition 49

6.6.6.6.2 Behavior 49

6.6.6.6.3 Parameters 49

7 Traffic characteristics 52

7.1 General 52

7.2 Complete/Basic AI/ML model distribution 52

7.3 Split AI/ML operation 52

7.4 Distributed/federated learning 52

8 KPIs 52

8.1 General 52

8.2 List of KPIs 52

9 Potential Normative Work 52

10 Conclusion 52

Annex <A>: <Informative annex title for a Technical Report> 53

Annex <X>: Change history 54

# 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 drafting the TS/TR, pay particular attention to the use of modal auxiliary verbs! TRs shall not contain any normative provisions.

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

The present document …

# 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".

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

[bb] 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>

[cc] Supervised Compression for Resource-Constrained Edge Computing Systems https://arxiv.org/pdf/2108.11898.pdf

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

[ac] "Application and Verification of NNC in Different Use Cases", MPEG document MDS22894 WG04 N00366, MPEG Video Coding ISO/IEC JTC 1/SC 29/WG 04, July 2023.

…

[x] <doctype> <#>[ ([up to and including]{yyyy[-mm]|V<a[.b[.c]]>}[onwards])]: "<Title>".

# 3 Definitions of terms, symbols and abbreviations

This clause and its three subclauses are mandatory. The contents shall be shown as "void" if the TS/TR does not define any terms, symbols, or 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].

Definition format (Normal)

**<defined term>:** <definition>.

**example:** text used to clarify abstract rules by applying them literally.

## 3.2 Symbols

For the purposes of the present document, the following symbols apply:

Symbol format (EW)

AI Artificial Intelligence

DNN Deep Neural Network

HDR High Dynamic Range

ML Machine Learning

NLP Natural Language Processing

NN Neural Network

SDR Standard Dynamic Range

UE User Equipment

UL Up-Link

## 3.3 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].

Abbreviation format (EW)

<ABBREVIATION> <Expansion>

# 4 Introduction to AI/ML for media

## 4.1 General

[Editor’s note: Introduction to the concepts of artificial intelligence and machine learning].

## 4.2 Media-based AI/ML use cases and scenarios

### 4.2.1 Introduction

TR 22.874 [aa] 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 device. The device 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 device. Alternatively, the network endpoint may firstly execute the operation/model up to a specific part/layer and then sends intermediate data to the device, 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 mobile device. 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 can 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 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 can perform the training for the next iteration.

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

### 4.2.2 Object recognition in image and video

Based on clause 5.1 and 5.2 of TR 22.874 [aa], 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 mobile device to the edge or a cloud data center.

Split inference of trained 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:

- Device/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 feeds the input data to the UE inference model (e.g., to preserve privacy). The UE then uploads intermediate output data from the UE inference model to the network inference, which in turn executes the remaining part of the model (e.g., process the intensive computations) and finally returns the results or a processed image/video to the UE.

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 executing the remaining layers of the model (e.g., task specific operations) and returning 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 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.

### 4.2.3 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 [aa], 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:

说明: A screenshot of a cell phone

Description automatically generated

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 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.

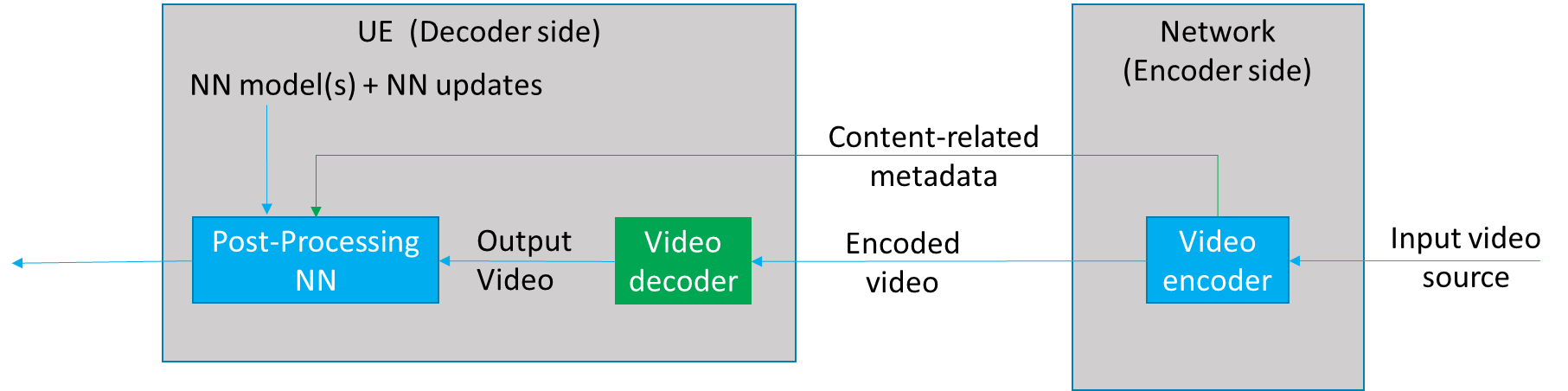


Figure 4.2.3-2: Neural network based post-processing for video coding use-case

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 [aa]. 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.

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

#### 4.2.4.2 Device 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 a large number of UEs in a short period of time. The UEs are heterogeneous, running with different types of operating systems (e.g., Android or iOS), 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 device 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-device 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 a large number of 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 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.

### 4.2.5 Natural Language Processing (NLP) on speech

Based on clause 6.3 of TR 22.874 [aa], 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.

## 4.3 Related work

[Editor’s note: list the AI/ML-related activities in 3GPP and elsewhere, e.g. MPEG…].

# 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 splittable, meaning that it may be theoretically represented by several sub-models separated by split points as illustrated for a fully connected artificial neural network (ANN) example in figure 5.1.1-1.

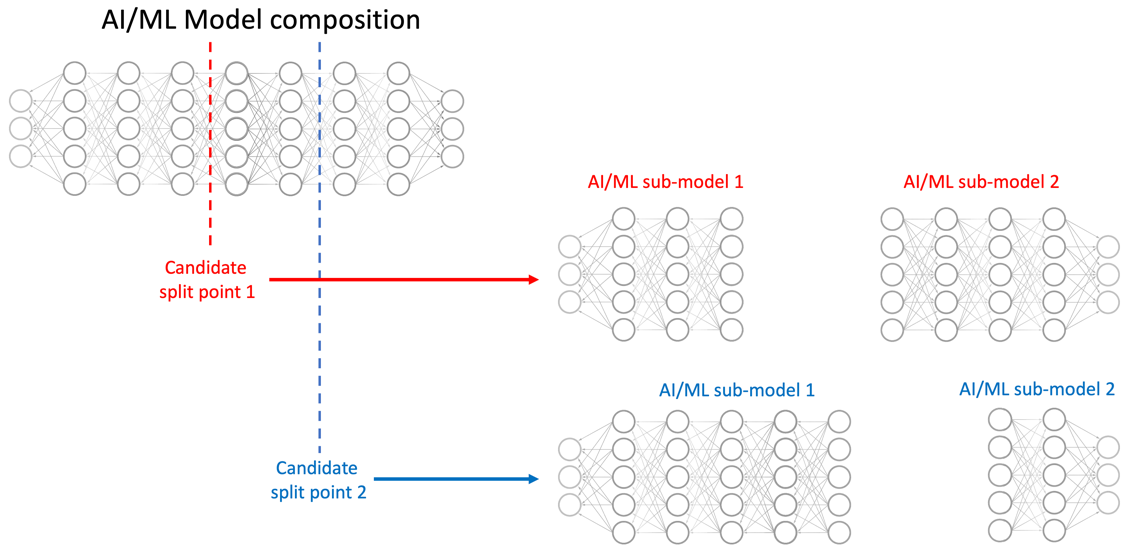


Figure 5.1.1-1: AI/ML model composition examples with a fully connected ANN

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.

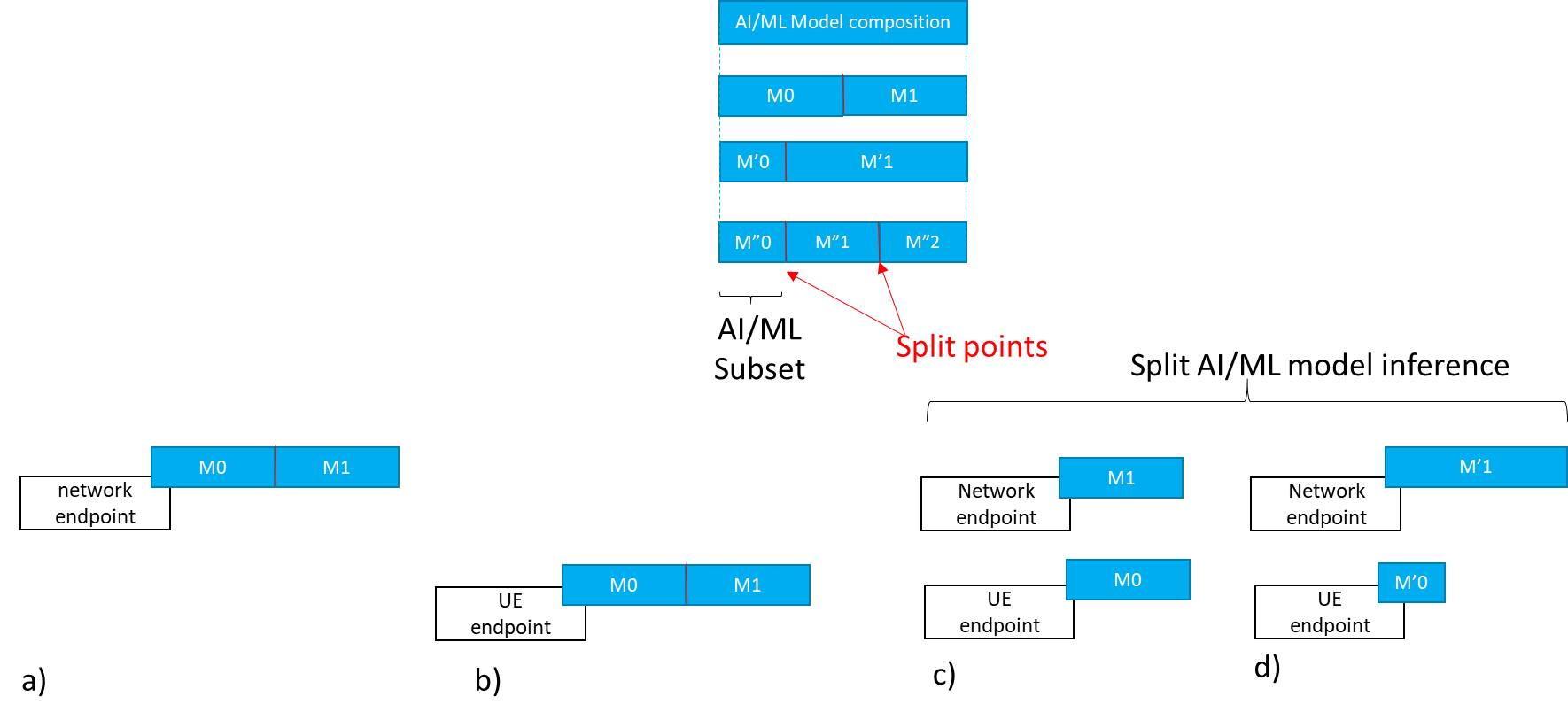


Figure 5.1.1-2: General AI/ML model composition examples

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 one split-point (i.e., only two sub-models) are addressed in this report.

NOTE: Service configurations addressing more than 2 AI/ML sub-models are for further study.

### 5.1.2 Topologies of split AI/ML 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 to be processed is.

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. 2 cases distinguished:

- The first AI/ML sub-model 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 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 first AI/ML sub-model runs a partial inference. The intermediate data is then sent to the UE and used locally by the second AI/ML sub-model 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.

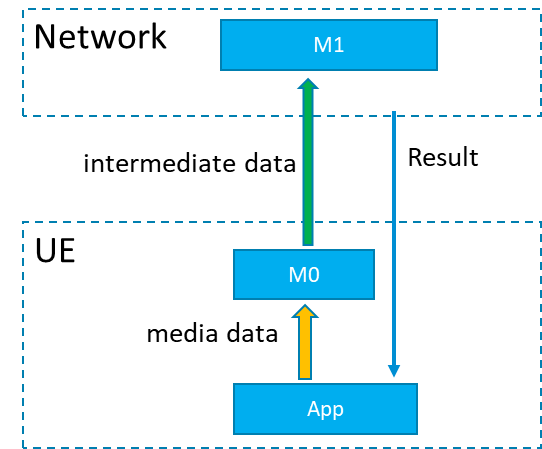


Figure 5.1.2-1: Split AI/ML model inference where the UE is the media data source with first inference endpoint on the UE

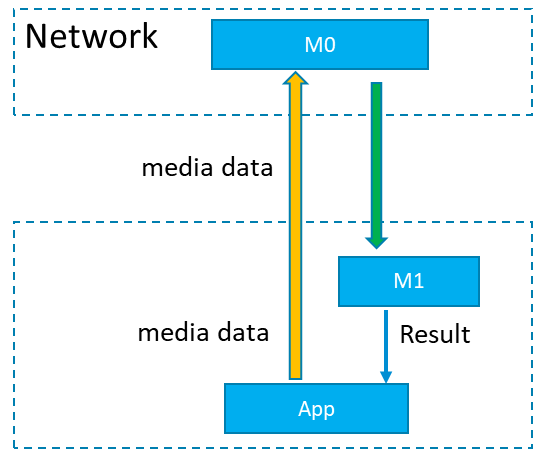


Figure 5.1.2-2: Split AI/ML model inference where the UE is the media data source with first inference endpoint on the network

#### 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 first AI/ML sub-model runs a partial inference. The intermediate data is sent to the UE that already has the second AI/ML sub-model available. This second AI/ML sub-model 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.

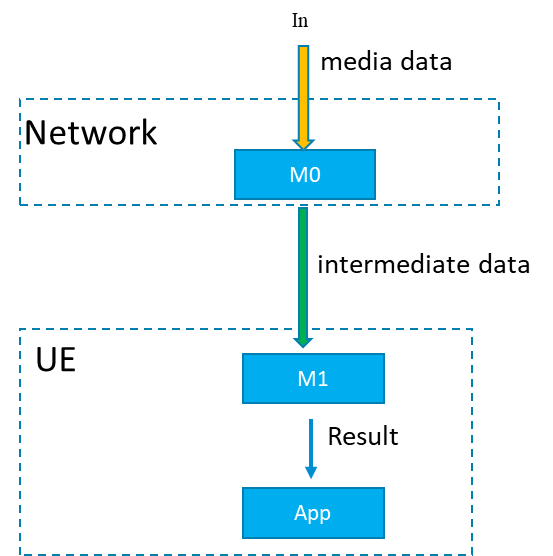


Figure 5.1.2-3: Split AI/ML Model inference where the network is the media source

## 5.2 Architectures and service flows

### 5.2.1 Introduction

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

The basic scenarios are:

1) Delivery of a pre-trained AI/ML model from network to UE, typically at the start of an AI 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 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 device inference (resp. network inference). Depending on the characteristics of the intermediate data, such as if the intermediate data is media content data, it may be practical to consider 5GMS architectures, procedures and/or protocols for the streaming delivery of such intermediate media data.

3) Distributed/federated learning using multiple UE devices with local training sets, and a central server in the network. Typically a central model is distributed to UEs for local training. UEs use local data available to the device 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 UE devices 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 (eg. traffic flow characteristics, latency, bitrate…).

### 5.2.2 Complete/basic AI/ML model distribution

#### 5.2.2.1 Basic architectures

Une image contenant texte, diagramme, logiciel, Police

Description générée automatiquement

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.

In the network:

- An AI model in the repository is selected for the AI media service by the network application, and sent to the delivery function for delivery to the UE. Selection of an AI 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 model delivery function sends the AI 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 model data.

In the UE:

- A UE application provides an AI media service using the AI model inference engine and AI model access function.

- The AI model access function receives the AI model data via the 5G system, and sends it to the AI model inference engine. Receiver side optimization or decompression techniques for AI model data may be included.

- The AI model inference engine performs inference by using the input data from the data source (e.g. a camera, or other media source) as the input into the AI model received from the AI model access function. The inference output data is sent to the data destination (e.g. a media player).

Depending on the exact service scenario, AI model updates may be necessary during the service, and different AI model data delivery pipelines may be considered for such purposes. An AI model update may consist of a change in the AI model structure without disrupting the AI media service. If the AI model has requirements on UE memory, processing/computing capabilities or if running the AI 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 Model. For example, when the memory usage of the UE processing the AI Model exceeds a certain threshold, or if UE performance deteriorates, the UE can actively send a request to the network for an AI Model update. Alternatively, the network may also trigger the AI 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 5.2.2-2 shows a basic workflow for AI/ML model delivery with inference in the UE. Steps for the procedures shown are described below.

Une image contenant texte, reçu, ligne, diagramme

Description générée automatiquement

Figure 5.2.2-2: 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 model delivery, using the information from the initialization and establishment step.

2. An AI model is selected between the UE Application and Network Application.

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

4. The AI Model Access Function establishes an AI model delivery session with the AI Model Delivery Function.

5. The AI Model Access Function receives the AI model.

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

7. The Data Source passes media data to the AI model Inference Engine.

8. The AI Model Inference Engine performs AI inferencing.

9. The AI Model Inference Engine passes the inference output result to the UE Data Destination for consumption.

##### 5.2.2.2.2 Adapative 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 5.2.1.2-2 and text below shows a basic workflow for adaptive model delivery update. Steps for the procedures shown are described below.



**Figure 5.2.1.2-2: 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 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 5.2.1.2-1), 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 model in the AI model Repository/Provider.

Adaptive AI model delivery session loop

* 1. The AI Model Access Function establishes an AI model delivery session with the AI Model Delivery Function.
  2. The AI Model Access Function receives the AI model of the precision requested by the UE.
  3. The AI Model Access Function passes the AI/ML model to the AI model Inference Engine in the UE.
  4. The Data Source passes data to the AI model Inference Engine, AI Model Inference Engine performs AI inferencing,
  5. The *AI Model Inference Engine* performs AI inferencing.
  6. AI Model Inference Engine passes the inference output result to the UE Data Destination for consumption.

Mode delivery update.

* 1. The UE application triggers a model precision update for updating the AI model to a higher precision.

AI 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.

* 1. 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

Une image contenant texte, diagramme, logiciel, capture d’écran

Description générée automatiquement

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 model is executed on the network side and the second part on the UE.

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

In the network:

- An AI model inference engine that receives both the network AI model subset(s), and input data, for network inference. The input data may come from the UE through 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 final inference output data is sent to the data destination (e.g. a media player).

Une image contenant texte, capture d’écran, diagramme, Police

Description générée automatiquement

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.

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

In the UE:

- An AI model inference engine that receives both the network AI model subset(s), and input data (from a UE data source), for UE inference.

- An intermediate data delivery function receives the partial inference output (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 inference output data from the network via the 5GS, and sends it to the relevant data destination according to the AI 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 final inference output data is sent 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 model subset” and “network AI model subset”)

- Resource allocation and management for network inference, including ingestion of network AI 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 FFS.

- Certain components may also overlap with functions defined in 5GMS or RTC architectures, clarifications are FFS.

#### 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.

Une image contenant texte, reçu, diagramme, nombre

Description générée automatiquement

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 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 inference processing capabilities available for the AI service, supported AI framework information, connection capabilities, etc.

**Alternative Case#1: Network decides the split inference**

2a. The UE Application sends an AI 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 inference processing capabilities, supported AI framework for the AI 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 model (including the UE AI model subset and the network AI model subset) for split inference from all matched AI models (with different candidate split points configurations) based on the service requirement information, the UE’s capability information and the network’s capability information.

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

6a. The AI Model Inference Engine responds with a successful result to the Network Application.

7a. The Network Application sends the AI Split Inference Response with the selected UE AI 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 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 inference processing capabilities, supported AI framework for the AI Inference Engine, available bandwidth (uplink/downlink) and preferred delivery method.

3b. The Network Application collects all matched AI models with different candidate split points configurations based on 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 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 Model Information Response.

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

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

8b. The AI Model Inference Engine responds with a successful result to the Network Application.

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

**AI Model Subset Delivery:**

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

11. The AI Model Inference Engine in the network receives the network AI model subset.

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

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

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

**AI split inference:**

**Alternative case#1: data source in the network**

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

16a. The network AI model Inference Engine performs network AI 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 Model Inference Engine.

19a. The AI Model Inference Engine in the UE performs AI inferencing.

20a. The AI 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 model Inference Engine.

16b. The UE AI model Inference Engine performs UE AI 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 Model Inference Engine.

19b. In the network, the AI Model Inference Engine performs network AI inferencing.

20b. The network AI 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

Une image contenant texte, capture d’écran, diagramme, Plan

Description générée automatiquement

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

Figure 5.2.4-1 shows a simple 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 model repository, that is passed to the AI 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 model data is received by an AI model access function via the 5GS, which then passes the data to the AI training engine.

- An AI training engine in the UE trains the AI 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.

Une image contenant texte, reçu, diagramme, nombre

Description générée automatiquement

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 model is selected between the UE Application and Network Application.

3. The Network Application identifies the selected partially trained AI model in the AI 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 Model Access Function of an eligible device receives the partially trained AI 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 model Training Engine.

9. The AI 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 model Training Engine.

14. The AI 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 model.

17. The updated partially trained AI 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 data delivery

### 5.3.1 AI data components

AI-related user plane data includes:

- AI model data, including data describing the topology/structure of the AI 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, defined as the output data from the 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 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.

- Inference output data, which is the data corresponding to the output result of the final AI inference process for the service. Depending on the nature of the AI data inferencing for the given AI 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, corresponds to all inputs feeding the AI 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 for media use-cases, input is media data (image, video, audio, etc.)

### 5.3.2 Media-related AI data logical functions

The identified User plane logical functions supporting the scenarios include:

- AI data delivery function

- AI data access function

- AI model inference engine

For split AI/ML, control plane functions in both the UE and network are needed for configuration, capability exchange and reporting:

- AI capability manager

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

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

Table 5.3-1:Logical AI functions

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

### 5.3.4 Architecture and components for AI data delivery over 5G

#### 5.3.4.1 Introduction

Une image contenant texte, capture d’écran, diagramme, Plan

Description générée automatiquement

Figure 5.3.4-1: AI data delivery general architecture

A generalized 5G Media Delivery architecture supporting AI 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, additional definitions for AI 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 Application Function (AF) to establish and control the configuration of an AI 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 UE model subset part for a split inference topology between the UE and the network.

- **Media Access Function**: A function on the UE that communicates with the Media AS and the Media Session Handler to establish an AI data delivery session. The function contains:

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

- An AI 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 model data for inference process. This includes instantiating an AI data access client to access and retrieve AI models or AI 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 data with deserialization or decompression technique.

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

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

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

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

- **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. 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.

### 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 split inference management, and

- AI data delivery session

- Split inference processing

Une image contenant texte, diagramme, reçu, Parallèle

Description générée automatiquement

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

1. Service provisioning and announcement of AI 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 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 models. Such additional information may contain a list of features available from each AI model, including its variants, including specific information such as the inferencing accuracy, the size, the amount of nodes, structure, complexity and latency requirements of the AI model.

AI split inference management:

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

4. Requesting AI split inference. Either the UE or the network requests the other side for an AI split inference service. If information describing the AI 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 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 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 may be shared between the network and UE depending on the configuration and a set of split points can be negotiated.

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

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

Split AI data session

8. Request the start of intermediate data delivery. On confirmation, the application triggers the Media Client to request the start of AI data delivery using the AI 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 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, or any other form of delivery mechanism required by the AI data service.

11. Start inference process in the UE. In this step, the Media client triggers the inference process (the AI 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 AS (the AI 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, or any other form of delivery mechanism required by the AI 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 using the pipelines defined in the AI data delivery session.

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 media service status (for example AI inference status, latency, resource status, capability status, dynamic media properties etc.) to the 5GAI AF.

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

18. The Media Session Handler may receive network status, or network AI 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 model selection, split point configuration or the AI 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 models from the Media Application Provider, the procedure consists of an ingest session (where AI models are uploaded to the Media AS), and a provisioning session, during which the Media Client can access the AI models and the Media Application Provider can control and monitor the AI models and its delivery.

Une image contenant texte, diagramme, nombre, Police

Description générée automatiquement

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 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 models. In step 9, the Media Client performs AI inferencing using media as an input into the AI 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.

# 6 Data components for AI/ML-based media services

## 6.1 General

[Editor’s note: Identify and document the data types and possible data formats for the different data components listed.].

## 6.2 Model data

### 6.2.1 Model optimization techniques

Trained models consist of a graph representations 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 aforementioned techniques are sender-only techniques that 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 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:

Decision making

Clustering

Regression

Classification

Supervised learning

Unsupervised learning

Reinforcement learning

Machine Learning types

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

#### 6.2.4.2 Supervised learning

As explained in [bb] 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.

Reinforcement learning 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 [2] is an open specification that was developed to facilitate the exchange of machine learning models between different AI 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 format that was developed initially by Google.

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) [3] 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 [4] 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.

- Signaling 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 [2], ONNX [3]) 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 signaled 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 signaled 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 [4].

SC29 WG04 is also already working on a second edition of ISO/IEC 15938-17, of which a Draft International Standard (DIS) has been completed. The second edition adds the functionality to compress incremental updates of neural networks, which can e.g. be applied to sending updates of neural networks or to federated learning scenarios.

Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement

Figure 6.2.5-1: Generation of a neural network representation (NNR) bitstream consisting of NNR units. Tools for pre-processing, parameter reduction, quantization, and entropy coding can 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 inference task use-case and requirement: The service requirements on an AI task drives the intermediate data size. For example, a complex AI task for detecting multiple objects in a dense and moving video requires far more intermediate data than for a simpler AI task on static scene or about a single object.
* AI model for the AI inference task: Different trained AI models for the same AI inference task can be available with different characteristics on not only the AI 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 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 [aa].
* Adapted trained model for split operation: Adapted models can be designed to provide reduced intermediate data at identified split points [cc].
* 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 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)

## 6.4 Existing frameworks for AI/ML

[Ed’s note: add a reference to those solutions]

### 6.4.1 TensorFlow

#### 6.4.1.1 Introduction

TensorFlow 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.

### 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 can 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

[Editor’s note: referring to the media data streaming formats and profiles in 26.512.]

## 6.6 Metadata

[Editor’s note: Metadata may include metadata to describe AI/ML model types, metadata for split operation configurations, AI/ML operation endpoint capability metadata etc.]

### 6.6.1 Introduction

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

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

### 6.6.2 Common AI model information

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Metadata category** | **Metadata type** | **Definition** | **Metadata type description (Examples)** |
| **Model information** | **Model identifier** | An identifier for an AI model (or variants of it) specified for a certain AI 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 model file in megabytes. | 40MB |
| **Input size** | The maximum size of the input data supported by the AI model in kilobytes. | 256 KB |
| **Output size** | The maximum size of the output data supported by the AI model in kilobytes. | 256 KB |
| **Accuracy** | The trained accuracy of the AI model as a percentage. | 85% |
| **Target inference latency** | The target inference latency specified for a given AI model in milliseconds. Such latency is measured between the input and output layers of the AI model at inference. This value is related to the service inference latency requirement of the service for which the AI 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 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 model information for split AI/ML operations

AI model information metadata for split AI/ML operations is used to describe the characteristics of AI models for split inference service scenarios. This information may be used in the selection of a split point (from which a multiple may be predefined by the service provider for a certain AI 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 happened before or after a graph node identified by its name or a number.

|  |  |  |  |
| --- | --- | --- | --- |
| **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. 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 give split point, in kilobytes. Intermediate data size is typically dependent on the tensor 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 |

### 6.6.3 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Metadata category** | **Metadata type** | **Definition** | **Metadata type description (Examples)** |
| **Intermediate data information** | Tensor structure information | The exact underlying tensor structure of the intermediate data tensors including the exact version of it. | PyTorch 2.0,  Tensor flow v2.13.0, NumPy v1 .25 |
| Tensor shape | The tensor shape(s) when the output is intermediate data. 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 element data type | The data type of each output intermediate data tensor | :int64, Float32 |
| 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 |
| Compression algorithm | Identifies the compression algorithm(s) that can be applied to the intermediate data. 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, FC\_VCM, SNAPPY, … |

### 6.6.4 Service requirement information

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

|  |  |  |  |
| --- | --- | --- | --- |
| **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 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 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% |

### 6.6.5 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 work load or the network conditions. It can be used for the selection of AI model, split inference, intermediate data compression, progressive model delivery.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 Framework** | The AI framework(s) supported by the endpoint. | TensorFlow 2.0 |
| **Supported compression algorithms** | The supported compression algorithm(s) for intermediate data compression. | NONE, FC\_VCM, SNAPPY, … |
| **Connection capabilities** | This indicates the available bandwidth in bit/s between the UE and the network for transmitting the AI model and/or the intermediate data. | 256 kb/s |

### 6.6.6 Distributed/Federated learning information

#### 6.6.6.1 Control information

##### 6.6.6.1.2 General

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.

Editor's note: Placeholder for merging current text on distributed/federated learning metadata.

|  |  |  |
| --- | --- | --- |
| **Metadata category** | **Metadata type** | **Definition** |
| **Control information** |  |  |
| **Synchronization information** |  |  |
| **Device eligibility information** |  |  |
| **Model evaluation information** |  |  |
| **Model update information** |  |  |
| **Failure reporting information** |  |  |

#### 6.6.6.2 Synchronization information

##### 6.6.6.2.1 Definition

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.

##### 6.6.6.2.2 Behavior

The network application sends synchronization information to all UE applications to start a new round of training at the same time as described in step 1 of figure 5.2.4-2. The information contains the round number and may also contain a timestamp indicating when the training round should begin.

##### 6.6.6.2.3 Parameters

The possible parameters are:

- The Round\_number indicates the training round in a model training.

- The Start\_time indicates the start time of the training.

- The Duration indicates the desirable duration of the training. This value just shows an indication of the desirable time for completing the training round.

#### 6.6.6.3 Device eligibility information

##### 6.6.6.3.1 Definition

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.

##### 6.6.6.3.2 Behavior

The Federated learning engine sends a device eligibility information to the AI model training engine to select the devices that meet certain criteria defined by the application as described in step 4 of figure 5.2.4-2. Depending on the number of criteria met, the application assigns a group id to the device. For example, the criteria could contain information about the device's operating system, processor speed, available memory, available image library (number of images…), geographical location of the device, language setting, and other attributes.

##### 6.6.6.3.3 Parameters

The possible parameters are:

- The Group\_id is 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.

- The Application\_group\_id, 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.

- The Hardware, Location, and Language parameters define the hardware, location, and language eligibility criteria respectively for the device.

- The Data\_library\_id defines the data library an eligible device shall have.

Note: if more than one eligibility field exists, the device needs to meet all criteria to become eligible.

#### 6.6.6.4 Model evaluation information

##### 6.6.6.4.1 Definition

Model evaluation information may be used to evaluate the performance 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.

##### 6.6.6.4.2 Behavior

For Federated learning engine sends the model evaluation information to the AI model training engine in the UE containing the metrics to be used for evaluation such as accuracy or precision as described in step 7 of figure 5.2.4-2.

##### 6.6.6.4.3 Parameters

The possible parameters are:

- The Round\_number shows the round after which the evaluation is performed.

- The Metric\_number shows the number of metrics included in this information body.

- The Metric is one or more of the Name-Value pairs showing the name of the metric and the corresponding value obtained in the evaluation.

#### 6.6.6.5 Model update information

##### 6.6.6.5.1 Definition

Model update information may be used to update the model parameters on the devices after each round of training. 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 with the new parameters updated by the local training on the device.

##### 6.6.6.5.2 Behavior

The server may send a model update information to all devices to update the AI/ML model with the new model parameters as described in step 5 of figure 5.2.4-2. The information contains the model id of the AI/ML model to be updated, the updated model parameters that the UE will use to train the model in the next round, and the new model id when the parameters are updated.

After running the training locally, each AI Model training Engine in the UEs may send a model update information to the server with the updated parameters as described in step 15 of figure 5.2.4-2. Together with the received model evaluation information, the server can decide if the global model needs to be updated or not. The model update information then only contains the model id of the AI/ML model used for local training and the updated parameters.

##### 6.6.6.5.3 Parameters

The possible parameters are:

- The Parameters includes the new model vector of values.

- The New\_model\_id is the id of the new model when the server sends the model to one or more devices.

#### 6.6.6.6 Failure reporting information

##### 6.6.6.6.1 Definition

Error 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.

##### 6.6.6.6.2 Behavior

The server sends a request to all devices to report a device failure or network disruption as described in step 6 of figure 5.2.4-2. For example, if a device fails to send its model parameters back to the server, the device should notify the server so that the device has been removed from the training process.

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.

##### 6.6.6.6.3 Parameters

The information describes the reason for the failure.

6.7 Existing optimization and compression tools for AI/ML models

6.7.1 AIMET library

The AI 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 6.7.1-1depicts the concept of the AIMET library.

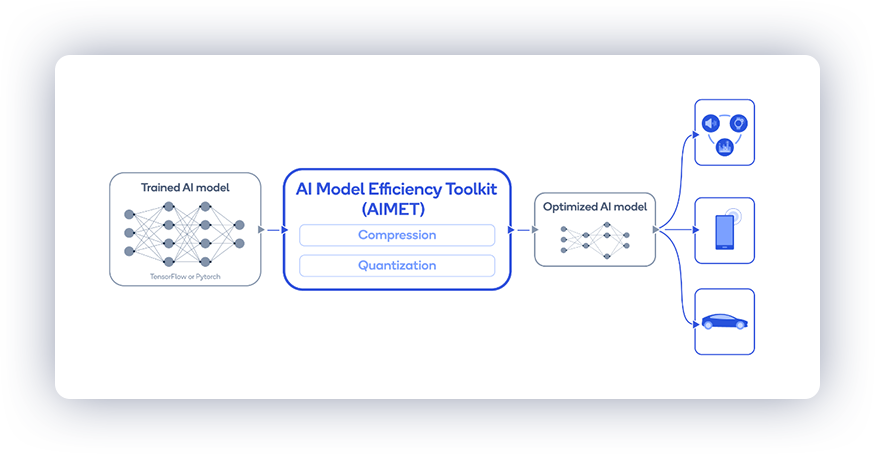


Figure 6.7.1-1: Concept of the AIMET library

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 [ab].

6.7.2 MPEG NNC

Available tools for compression with the NNC standard are MPEG’s reference software NCTM (neural network compression test model)[ref] and NNCodec[ref] (neural network encoder/decoder).

NNC use cases and validation results provided by MPEG WG4

Table 6.7.2.1-1 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 [ac]. 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 [ac].

Table 6.3.2.1-1: 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 can be compressed to 2% to 20% at transparent performance. According to [ac], even greater bit rate reductions are possible when tolerating small performance reductions as a trade-off.

# 7 Traffic characteristics

## 7.1 General

[Editor’s note: Based on the architectures, identify for the relevant data components for each of the scenarios, the corresponding traffic characteristics (burst size, delay/bandwidth/reliability requirements etc.)]

## 7.2 Complete/Basic AI/ML model distribution

## 7.3 Split AI/ML operation

## 7.4 Distributed/federated learning

# 8 KPIs

## 8.1 General

## 8.2 List of KPIs

[Editor’s note: E.g. Latency, data rate, reliability, accuracy…]

# 9 Potential Normative Work

# 10 Conclusion

Annex <A>:  
<Informative annex title for a Technical Report>

Informative annexes in Technical Reports do not use "(informative") in the title, since all annexes in TRs are informative. Use style "Heading 9" in TRs.

Annex <X>:  
Change history

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Change history** | | | | | | | |
| **Date** | **Meeting** | **TDoc** | **CR** | **Rev** | **Cat** | **Subject/Comment** | **New version** |
| 2022-01 | SA4#118e | 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 (S4231884), 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 |  |  |  |  | Procedure for AIML distribution and operation (S4-240649), existing formats (S4-240646), general architecture (S4-240XX), basic workflow for split inferrencing (S4-240YY), frameworks update (S4-2400ZZ), existing frameworks update (S4-240ZZ) | 0.7.0 |