**Source: Samsung Electronics Co., Ltd. (Rapporteur)**

**Title: [FS\_AI4Media] Permanent Document v0.6**

**Version: 0.6**

**Agenda Item: 15.2**

**Document for: Agreement**

# 1 Introduction

During SA4#117-e the New Study Item on “Artificial Intelligence (AI) and Machine Learning (ML) for Media” in S4-220226 was agreed and afterwards approved in by SA#95e in SP-220328.

The objective of this study item are primarily to identify the media service architectures and relevant service flows, model operation configurations, data components including available data formats, and the data traffic characteristics in AI/ML for media related services. Key performance indicators and performance metrics are also identified.

The concrete objectives are as follows:

* List and describe the use cases for media-based AI/ML scenarios, based on those defined in TR 22.874.
* Describe the media service architecture and relevant service flows for the scenarios, identifying for each use case the impacts on the architecture, including any potential gaps with existing 5G media service architectures. Also describe the model operation configurations for each use case, including split AI/ML operations, identifying where certain AI/ML operations occur.
* Identify and document the available data formats and suitable protocols for the exchange of different data components of various AI/ML models, such as model data, metadata, media data, and intermediate data necessary for such model operation configurations. Also investigate the data traffic characteristics of these data components for delivery over 5G system, including whether there are any needs and potentials for data rate reduction.
* Identify and study key performance indicators for such scenarios, based on the initial considerations in TS 22.261, with additional emphasis on the use cases, model operation configurations and data components as identified in earlier objectives, focusing on objective performance metrics considering the KPIs identified.
* Identify potential areas for normative work as the next phase and communicate/align with SA2 as well as other potential 3GPP WGs on relevant aspects related to the study.

# 2 Related works

## 2.1 AI/ML work in 3GPP WGs

This clause documents the 3GPP activity related to AI/ML in other Working Groups.

- SA1 has completed an initial study item on traffic characteristics and performance requirements for AI/ML model transfer in 5GS (FS\_AMMT), documented in TR 22.874. This technical report describes a variety of different use cases for AI/ML in 5G, with many that are related to media services. The media related use cases described in TR 22.874 are used as a basis for those listed and described in clause 4.2 of this TR. Resulting from this study item, SA1 has completed related normative works by way of multiple CRs on TS 22.261 (AMMT), reflecting new service requirements and KPIs for AI/ML model transfer in 5GS. Leading from this initial work, SA1 has also subsequently established a Rel-19 study on AI/ML model transfer phase 2 (FS\_AIML\_MT\_Ph2), the objectives of which are to study new use cases and potential service and performance requirements to support efficient AI/ML operations using direct device connection. This study avoids overlaps with stage-23 work ongoing in Rel-18.

- SA2 is in progress of a study item on system support for AI/ML-based services (AIMLsys). The scope of this study is based on requirements from SA1, including 7 key issues related to the training and inference processes of AI/ML applications, namely monitoring of network resources to support application AI/ML operations, 5GC information exposure to UE and authorized 3rd party, enhancing external parameter provisioning, QoS and policy enhancements, among others.

- SA3 has recently approved a study item on security and privacy of AI/ML-based services and applications in 5G (FS\_AIML). The objectives are to identify what security and privacy management is needed for data transmission to application layer AIML, including authentication and authorization of data collection and sharing between UE, AF and the network, and securing of AIML-based services and operations.

- SA5 has a study item on AI/ML management (FS\_AIML\_MGMT), related to automation and intelligence in 5G, including management and orchestration (e.g. MDA), 5GC (e.g., NWDAF), and NG-RAN. The objectives are to provide validation/testing of models and AIML enable functions, deployment of these models and functions, and configuration and performance evaluation of AIML enabled functions. The study will also investigate what coordination is needed between AIML management capabilities and 5GC AIML capabilities.

- SA6 is in progress of a study on application data analytics enablement service (FS\_ADAES), the goal is to study how to provide application layer data analytics as a possible new capability at the enablement layer for supporting the application specific layer to receive useful statistics/predictions for the application service, while complementing the analytics provided by the 5GS.

- RAN1 is in progress of a study on the 3GPP framework for AI/ML for NR air interface. The goal of this study is to explore the benefits of augmenting the air-interface with features enabling improved support of AI/ML based algorithms for enhanced performance and/or reduced complexity/overhead. Enhanced performance here depends on the use cases under consideration and could be, e.g., improved throughput, robustness, accuracy or reliability, etc.

- RAN3 has a study item on specify data collection enhancements and signalling support within existing NG-RAN interfaces and architecture (including non-split architecture and split architecture) for AI/ML-based Network Energy Saving, Load Balancing and Mobility Optimization (AIML\_RAN). Normative work is expected to start in Q3 2022.

## 2.2 AI/ML work in MPEG WGs

### 2.2.1 MPEG Video Coding for Machine (FC\_VCM)

In the MPEG Requirements Working Group which explores new market needs, an ad-hoc group has been created to study the optimization of the Compression of Features in the context of Video Coding for Machine tasks (FC-VCM).

Intermediate data can consist of large tensors of floating-point values, which would require very large bitstream over 5G to enable split inference between the network and the UE. Therefore, compression may be required in this scenario. The VCM encoder and the VCM decoder would then be part of the intermediate delivery function and intermediate access function, respectively.

Figure 2.2.1-1 illustrates the considered pipeline where, like in the current study, a first part of the Neural-Network-based algorithm is split into two parts. The intermediate features are first encoded on the sender side and embedded in a bitstream, which is decoded at the receiver before inferring the second part of the Neural Network.



Figure 2.2.1-1: FC-VCM pipeline

This standard, which targets use-cases matching the proposed Intermediate data transfer, is expected to be finalized by the end of 2025.

The current baseline considers the use of traditional video compression methods, e.g., the latest H.266/VVC standard, to encode the features that are processed and packed into input frames to the codec. The activity has just started, and new methods are going to be proposed. As the AI models considered in this study rely on Neural Networks, it can be envisioned to optimize the compression of the intermediate features using trained auto-encoders as well, to minimize the size of the bitstreams to be transmitted over 5G, while conserving an acceptable accuracy of the inferred models.

# 3 Definition of terms, symbols and abbreviations

## 3.1 Terms

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

**AI/ML model:** A trained AI/ML model.

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

**Inference engine**: Functionality that provides runtime environment for a machine learning

model and exposes corresponding machine learning model inference capability [5].

**AI/ML model subset:** An elementary element of an AI/ML model that can be inferred independently.

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

**AI/ML model split points:** The points in a DNN AI/ML model where it is split into multiple AI/ML model subsets.

**AI/ML inference endpoint:** UE or Network inference engine that infers a result from executing an AI/ML model, or a part of it.

**Split AI/ML model:** An AI/ML model composed of AI/ML subsets that are distributed to, and inferred on different inference endpoints.

**Intermediate data:**Output from the inference process of an AI/ML model that is not considered the final inference result.

**Model update:** Partial or fullupdate of a trained model which may include its internal structure and/or related parameters (e.g. weight, biases).

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

TR 22.874 [1] 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.

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 [1].

## 4.1 Object Recognition in Image and Video

Based on clause 5.1 and 5.2 of TR 22.874 [1], 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:

1. 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.
2. 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 Video Quality Enhancement in Streaming

### 4.2.1 Sender-receiver approaches

#### 4.2.1.1 End-to-End neural network-based video coding

Based on clause 5.3 of TR 22.874 [1], 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 4.2.1.1-1:

说明: A screenshot of a cell phone

Description automatically generated

**Figure 4.2.1.1-1**

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.

#### 4.2.1.2 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 4.2.1.1, this approach does not use an intermediate data stream.

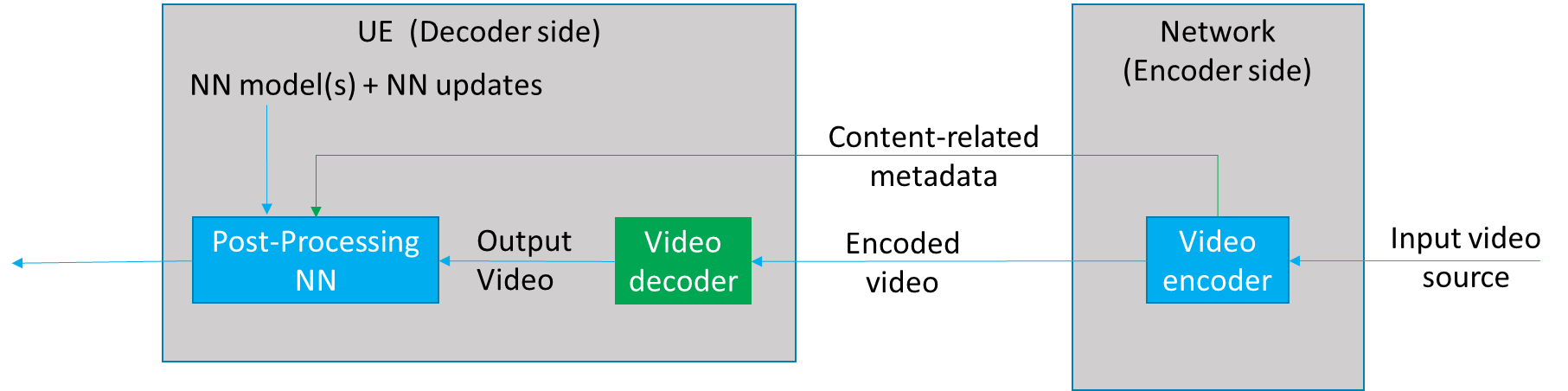


Figure 4.2.1.2-1 Neural network based post-processing for video coding use-case

Figure 4.2.1.2-1 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.

## 4.3 Crowd-Sourcing Media Capture

This use case and its corresponding scenarios are based on clause 6.2 of TR 22.874 [1]. 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.

### 4.3.1 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).

### 4.3.2 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.

## 4.4 NLP on Speech

Based on clause 6.3 of TR 22.874 [1], 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.

## 4.5 Split model adjustment during ongoing AI/ML service

Based on clause 5.5 of TR 22.874 [1], this use case covers all the cases where when the AI/ML models are computing intensive, the work tasks can be fully or partially offloaded to the network and the AI/ML split points can be dynamically adjusted by considering various factors such as UE capabilities (e.g. processing capability/computation resources), service performance, intermediate data volume, and network conditions such as bandwidth etc.

The AI/ML models are set to have different candidate split points and each candidate split point has different workload and communication requirements, as well as intermediate data characteristics. A policy decision point for the media task will adjust the split points of the AI/ML model for an ongoing service based on the factors of current UE’s capabilities, communication performance, intermediate data volume, network conditions etc. to make sure that the media work task can be executed well, guaranteeing the UE experience and avoiding service interruption.

For the 5G media system, both UE capabilities and network conditions are required to be monitored and used as some of the considering factors when updating the AI/ML model split points for an ongoing service; the UE and network can then inference based on the newly updated AI/ML split models in real time.

## 4.6 Deployment options

AI4Media services can be categorized into one of four different deployment scenarios, depending on how AI/ML is used in the service, these four scenarios and their characteristics being:

1. AI/ML used for media processing and/or handling:

- Where both the source and output to the service are media data.

- The AI/ML inference engine is typically inside that of a media-processing pipeline.

- In this scenario, a media service may trigger an AI4Media service.

2. AI/ML based service with media data as an input:

- Where the source to the service is media data and the output is non-media data.

- The AI/ML inference engine is typically outside that of a media-processing pipeline, and acts as a media consumer.

- In this scenario, an AI4Media service may trigger a media service.

3. AI/ML used for media generation:

- Where the source to the service is non-media data and the output is media data.

- The AI/ML inference engine is typically inside that of a media-processing pipeline, and acts as a media generator.

- In this scenario, a media service may trigger an AI4Media service.

4. AI/ML media service where a media pipeline is dedicated for the AI/ML framework:

- Where split AI/ML involves intermediate data having media characteristics.

- Where an AI/ML model is delivered in a streaming manner.

- In this scenario, an AI4Media service may trigger a media service.

Considering the use cases in this permanent document, the use cases may be mapped to the scenarios introduced as shown in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Scenario 1** | **Scenario 2** | **Scenario 3** | **Scenario 4** |
| **Use cases** | Video Quality Enhancement in Streaming  Crowd-Sourcing Media Capture | Object Recognition in Image and Video |  | NLP on Speech  Use cases where split AI/ML or AI/ML model streaming is involved |

From these scenarios, there is a need to consider both:

* 1. How an existing media service may support AI/ML, in particular how the media service may be triggered by an AI4Media application or service, or vice-versa. This is important if the media service is one that is supported by existing frameworks in SA4 (such as 5GMS), wherein the AI4Media service may need to be tightly coupled, or integrated into the existing media service framework (depending on the media pipeline).
  2. AI/ML media services where a new AI/ML framework (including its related AI/ML data formats and delivery mechanisms), may need to be defined in order to support dedicated AI/ML media pipelines.

# 5 Media service architecture for AI/ML

## 5.1 AI/ML model composition

Figure 5.1-1 depicts an AI/ML model composed of different AI/ML subsets based on various split points. 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 subset may be used in different compositions depending on the configurations of the model composition (e.g. M’0 and M ’00 according to figure 5.1-1).

In figure 5.1-1, (a) and (b) are examples of AI/ML inference endpoints running an AI/ML model M composed of two subsets M0, M1. A endpoint (network/UE) may run the AI/ML model subset M0 while downloading the other subset 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.

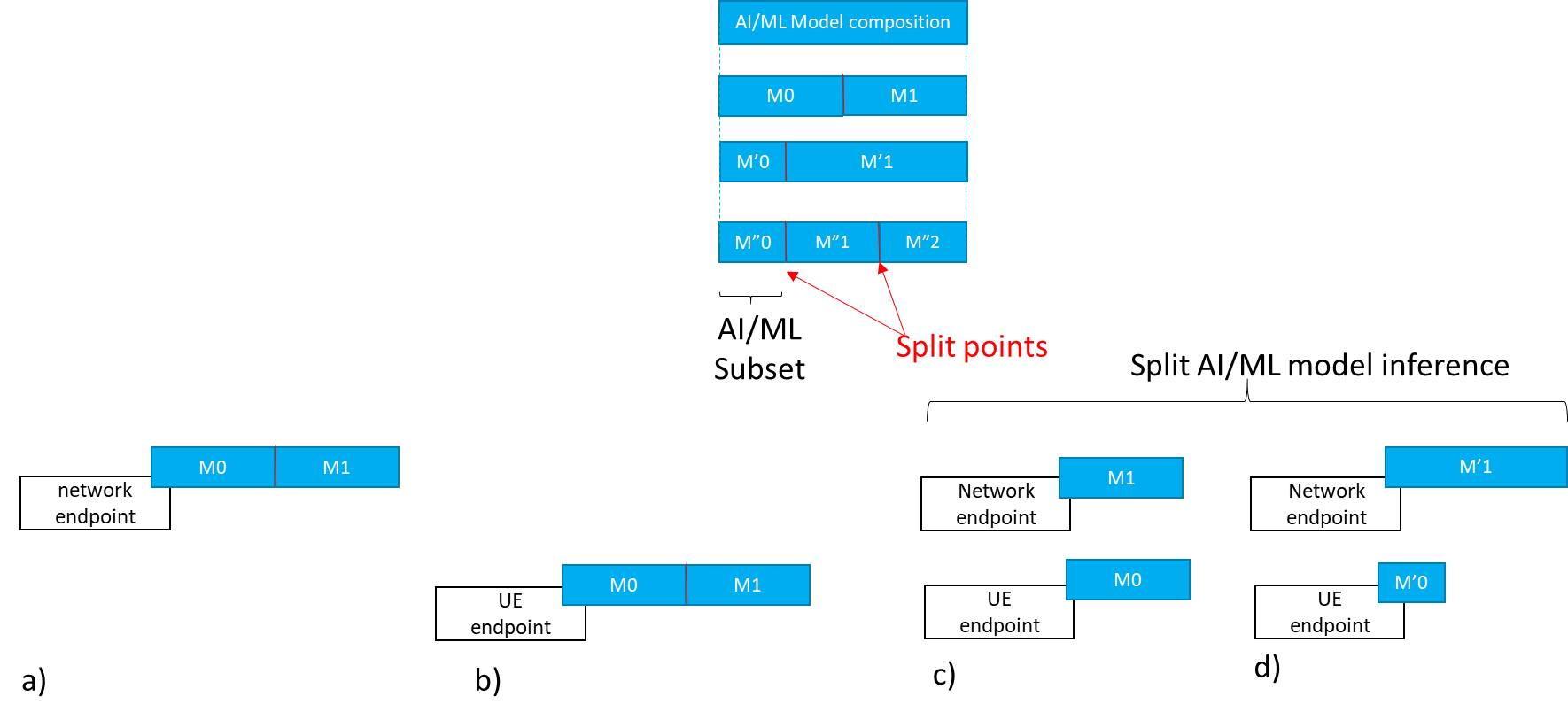


Figure 5.1-1 AI/ML model composition example

### 5.1.1 Split AI/ML model inference topologies

#### 5.1.1.1 UE as media data source

Figure 5.1.1.1-1 depicts an example of split AI/ML model inference topology where the UE is the media data source and runs the first model subset M0 as described in scenario (a) of clause 4.1 (object recognition). Figure 5.1.1.1-2 depicts another example of a split AI/ML model inference topology where the UE is also the media data source but the network server runs the first subset M0 as described in scenario (b) of clause 4.1. Assuming that the necessary AI/ML model subsets are already available at each endpoint, figure 5.1.1.1-1 and figure 5.1.1.1-2 show the data exchanged between the different split inference endpoints, including input media data, intermediate data, and inference results.

The results can be a textual indication of the recognized object, an output score, a bounding box, enhanced media data, etc.

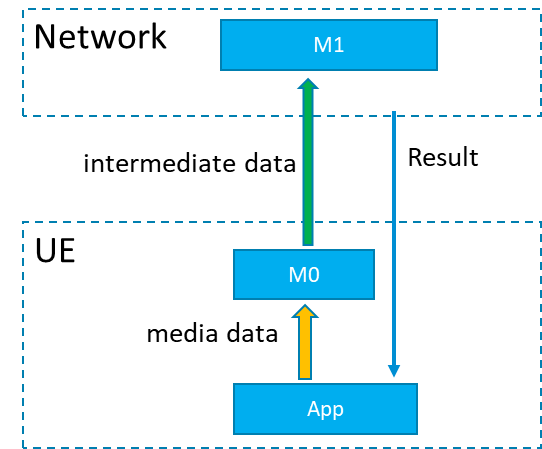


Figure 5.1.1.1-1: Split AI/ML model inference where the UE is the media data source with first inference endpoint on the UE

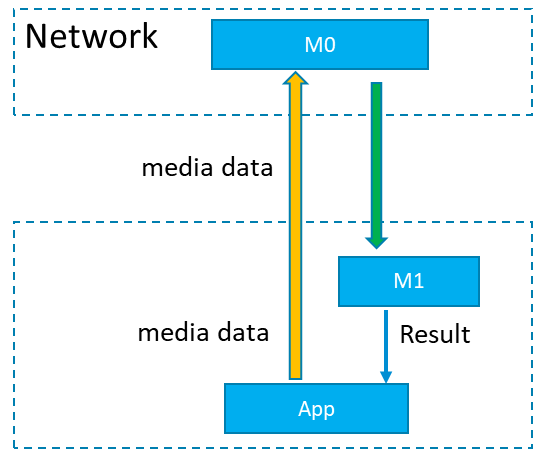


Figure 5.1.1.1-2: Split AI/ML model inference where the UE is the media data source with first inference endpoint on the network

#### 5.1.1.2 Provider/network as media data source

Figure 5.1.1.2-1 depicts examples of split model topologies where the network or the AI/ML provider ingests the media data, such as in the use-case of clause 4.2 (video quality enhancement).

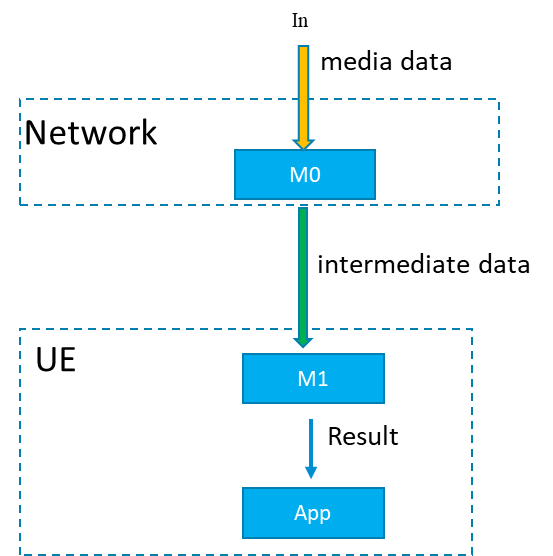


Figure 5.1.1.2-1: Split AI/ML Model inference where the network/ AI/ML service provider ingests the media data

## 5.2 Basic architectures and workflows

Considering the related use cases as documented in TR 22.874 and also as documented in the latest version of the Permanent Document, we can start from some basic scenarios for consideration of a basic architecture for AI/ML media services.

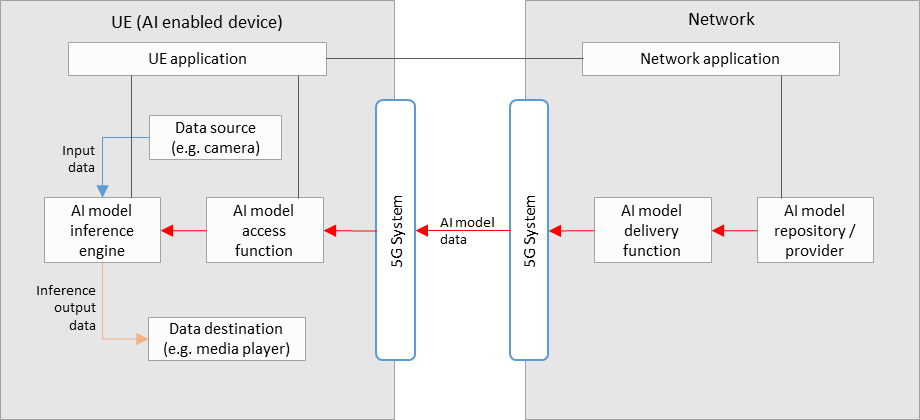
The basic starting scenarios are:

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:
   1. Basic scenario with an inference in the network or in the UE.
   2. 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…).

### 5.2.1 Complete/Basic AI/ML model distribution

#### 5.2.1.1 Basic architectures



**Figure 5.2.1.1-1: Basic architecture for AI/ML model delivery with inference in the UE**

Figure 5.2.1.1-1 shows a simple basic architecture for AI/ML model delivery, as described in scenario 1) of clause 5.2, with an inference of a pre-trained AI/ML model in the UE, as described in scenario 2a) of clause 5.2.

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.

#### 5.2.1.2 Basic workflows

Figure 5.2.1.2-1 shows a basic workflow for AI/ML model delivery with inference in the UE. Steps for the procedures shown are described below.



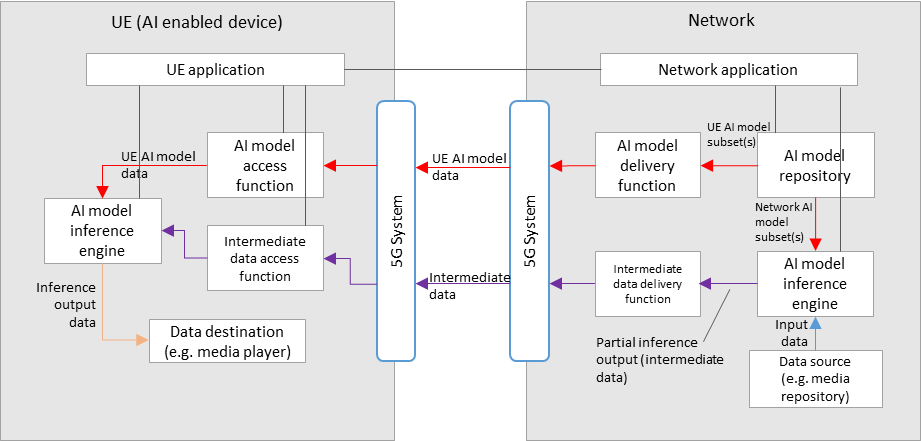
**Figure 5.2.1.2-1: Basic workflow for AI/ML model delivery with inference in the UE**

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 Split AI/ML operation

#### 5.2.2.1 Basic architectures



**Figure 5.2.2.1-1: 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 5.2.2.1-1 shows a simple basic architecture for split inferences between the network and the UE, as described in scenario 2b) of clause 5.2, 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.

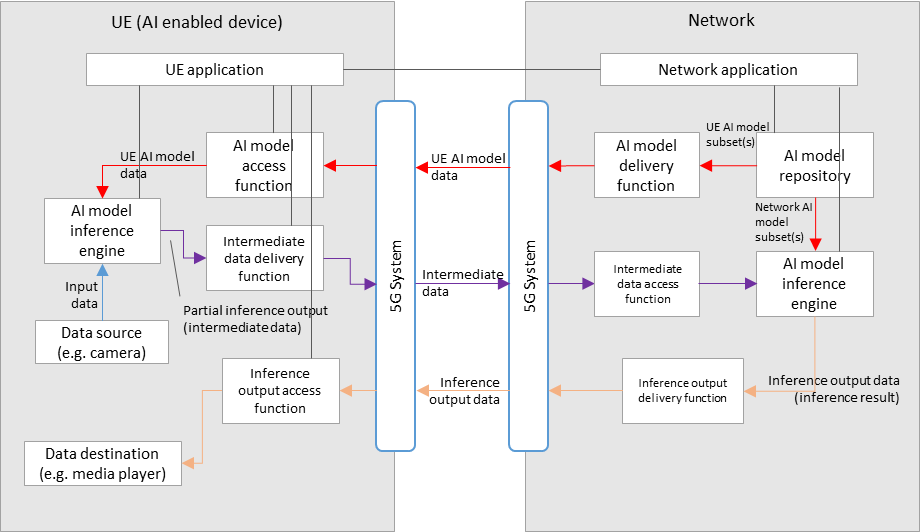
For the split inference (network-UE) scenario, additional components are required:

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).



**Figure 5.2.2.1-2: Basic architecture for split inference between the UE and network, with media data source in the UE**

Figure 5.2.2.1-2 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 should 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 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, clarifications FFS.

#### 5.2.2.2 Basic workflows

Figure 5.2.2.2-1 shows a basic workflow for split inference between the network and UE, with media data source in the network. Steps for the procedures shown are described below.



**Figure 5.2.2.2-1: Basic workflow for split inference between the network and UE, with media data source in 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 (including split capabilities), AI/ML service information (e.g. service requirements, split AI/ML model descriptions), and delivery methods. Such information may be used for the selection of a suitable split AI/ML model configuration, and its associated UE and network AI model subsets, for the service.

1. The *UE Application* and *Network Application* communicate to trigger split AI model delivery, using the information from the initialization and establishment step.
2. A split AI model is selected between the *UE Application* and *Network Application*.
3. The *Network Application* identifies the selected UE and network AI model subsets in the *AI model Repository/Provider*.
4. The *AI Model Inference Engine* in the network receives the network AI model subset.
5. The *AI Model Access Function* establishes a UE AI model subset delivery session with the *AI Model Delivery Function*.
6. The *AI Model Access Function* receives the UE AI model subset.
7. In the UE, the *AI Model Access Function* passes the UE AI model subset to the *AI model Inference Engine*.
8. In the network, the *Data Source* passes media data to the *AI model Inference Engine.*
9. The network *AI model Inference Engine* performs network AI inferencing.
10. The *Intermediate Data Access Function* establishes an intermediate data delivery session with the *Intermediate Data Delivery Function*.
11. In the UE, the *Intermediate Data Access Function* receives intermediate data and passes it to the *AI Model Inference Engine*.
12. The *AI Model Inference Engine* in the UEperforms AI inferencing.
13. The *AI Model Inference Engine* passes the inference output result to the *UE Data Destination* for consumption.

Figure 5.2.2.2-1 shows a basic workflow for split inference between the UE and network, with media data source in the UE.

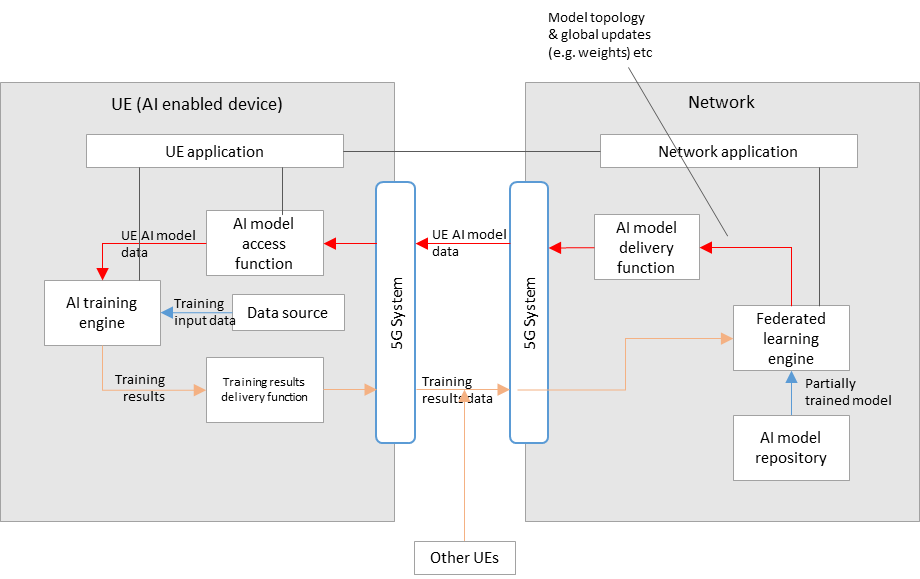


**Figure 5.2.2.2-2: Basic workflow for split inference between the UE and network, with media data source in the UE**

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 (including split capabilities), AI/ML service information (e.g. service requirements, split AI/ML model descriptions), and delivery methods. Such information may be used for the selection of a suitable split AI/ML model configuration, and its associated UE and network AI model subsets, for the service.

1. The *UE Application* and *Network Application* communicate to trigger split AI model delivery, using the information from the initialization and establishment step.
2. A split AI model is selected between the *UE Application* and *Network Application*.
3. The *Network Application* identifies the selected UE and network AI model subsets in the *AI model Repository/Provider*.
4. The *AI Model Inference Engine* in the network receives the network AI model subset.
5. The *AI Model Access Function* establishes a UE AI model subset delivery session with the *AI Model Delivery Function*.
6. The *AI Model Access Function* receives the UE AI model subset.
7. In the UE, the *AI Model Access Function* passes the UE AI model subset to the *AI model Inference Engine*.
8. In the UE, the *Data Source* passes media data to the *AI model Inference Engine.*
9. The UE *AI model Inference Engine* performs UE AI inferencing.
10. The *Intermediate Data Access Function* establishes an intermediate data delivery session with the *Intermediate Data Delivery Function*.
11. In the network, the *Intermediate Data Access Function* receives intermediate data and passes it to the *AI Model Inference Engine*.
12. In the network, the *AI Model Inference Engine* performs network AI inferencing.
13. The UE Data Destination receives the inference output result from the network.

### 5.2.3 Distributed/federated learning



**Figure 5.2.3-1: Basic architecture for distributed/federated learning between the network and multiple UEs**

Figure 5.2.3-1 shows a simple basic architecture for distributed/federated learning between the network and UE(s), as described in scenario 3) of clause 5.2.

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.

#### 5.2.3.2 Basic workflows

Figure 5.2.3.2-1 shows a basic workflow for distributed/federated learning with training in the UE, the results of which are aggregated in the network. Steps for the procedures shown are described below.



**Figure 5.2.3.2-1: 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 *AI Model Access Function* establishes an AI model delivery session with the *AI Model Delivery Function*.
5. The *AI Model Access Function* receives the partially trained AI model.
6. The *AI Model Access Function* passes the partially trained AI/ML model to the *AI model Training Engine* in the UE.
7. The *Data Source* passes the training input data to the *AI model Training Engine.*
8. The *AI Model Training Engine* performs AI training.
9. A training result delivery session is established between the *Training Result Delivery Function* and the *Federated Learning Engine*.
10. The *Federated Learning Engine* receives training results data from the UE.
11. The *Federated Learning Engine* performs training aggregation of training results from multiple UEs, and updates the partially trained AI model.
12. The updated partially trained AI model is delivered to the UE as from step 5.

## 5.3 Architecture for AI data delivery

### 5.3.1 AI data components

AI related user plane data include:

* 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.

### 5.3.2 AI4media data logical functions

User plane logical functions supporting the scenarios identified in the PD 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 Architecture for AI data delivery over 5G



Figure 5.3.3-1 AI data delivery general architecture

A possible architecture for AI data delivery over 5GS is shown in figure 5.3.3-1. Depending on the service scenario and/or use case, certain dedicated AI/ML logical subfunctions could be mapped to, or instantiated by 5GMS functions.

The 5G AI data delivery system shown in figure 5.3.3-1 includes the following main functional blocks:

* **5G AI Client** running on the UEcontains two subfunctions:
  + **AI data 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:
    - *AI capability manager* subfunctions that monitors, shares and/or reports UE capabilities with/to the *AI capability manager* function of the5G AI 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.
  + **AI Data Handler:** A function on the UE that communicates with the 5G AI Application Server (AS) and the AI data 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.
* **5G AI-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.
* **5G AI 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
  + *An AI inference engine*, which has the capability to perform the inferencing of (split) AI models.
* **5G AI AF(Application Function):** An Application Function that provides various control and configuration functions to the AI Data Session Handler on the UE and/or to the AI 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:
  + *AI capability manager* subfunctions monitors, shares and/or reports Network capabilities with/to the *AI capability manager* function of the *AI data 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.

### 5.3.4 Example procedure for Split AI/ML operation

Figure 5.3.4-1 shows an example procedure for split AI/ML operation, including three main parts:

* AI split inference management, and
* AI data delivery session
* Split inference processing



1. Service provisioning and announcement of AI data service on the network side, in particular between the 5GAI AF (application function) and the 5GAI 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 AI model specific information, such as the structure, the size, complexity and latency requirements of the AI model.

AI split inference management:

1. 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.
2. 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 shared during this step.
3. 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.
4. Acknowledge split and provide the AI data split inferencing access info. In this step, the network (5GAI 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.
5. The split configuration outcome is notified to the 5GAI-aware application.

AI data delivery session

1. Request the start of AI data delivery. On confirmation, the application triggers the 5GAI client to request the start of AI data delivery using the AI data access information provided in step 7.
2. The 5GAI client request the AI data to be deliver from the 5GAI AS.
3. Pipelines for the delivery of AI model data from the 5GAI AS to the 5GAI 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.
4. Start inference process in the UE. In this step, the 5GAI 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.
5. Start inference process in the server. In this step, the 5GAI 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.
6. Pipelines for the delivery of intermediate data from the 5GAI AS to the 5GAI 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

1. The split inference runs between the UE and the network.

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

## 6.1 Model data

### 6.1.1 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. Table 6.1.1-1 depicts the characteristics of some of the state-of-the-art DNNs as provided by [6].

|  |  |  |  |
| --- | --- | --- | --- |
| **Model** | **#Parameters (M)** | **Footprint (MB)** | **#FLOPs (B)** |
| 1.0 MobileNet-224 | 3.3 | 13.2 | 0.28 |
| EfficientNet-B0 | 5.3 | 21.2 | 0.39 |
| DenseNet-169 | 14 | 56 | 3.5 |
| Inception-v3 | 24 | 96 | 5.7 |
| ResNet-50 | 26 | 104 | 4.1 |
| VGG-16 | 138 | 552 | 16 |
| SSD300-MobileNet | 6.8 | 27.2 | 1.2 |
| EfficientDet-D0 | 3.9 | 15.6 | 2.5 |
| FasterRCNN-MobileNet | 6.1 | 24.4 | 25.2 |
| SSD300-Deeplab | 33.1 | 132.4 | 34.9 |
| FasterRCNN-VGG | 138.5 | 554 | 64.3 |
| YOLOv3 | 40.5 | 122 | 71 |

**Table 6.1.1-1: State-of-the-art DNN characteristics [6]**

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. We differentiate 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 very efficient 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.

### 6.1.2 Model update requirements and constraints

**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.

**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.

**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.

### 6.1.3 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.

### 6.1.4 Classes of AI/ML models

#### 6.1.4.1 Introduction

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

Decision making

Clustering

Regression

Classification

Supervised learning

Unsupervised learning

Reinforcement learning

Machine Learning types

#### 6.1.4.2 Supervised learning

As explained in [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>] 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.

#### 6.1.4.3 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.

#### 6.1.4.4 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).

### 6.1.5 AI model evaluation

In the process of AI/ML, no matter on the training set or on the new sample, there is always some difference between the output result of the model and the real value. Model evaluation is a process of using different evaluation metrics to understand the performance of artificial intelligence/machine learning models and its advantages and disadvantages. It is an indispensable part of the model development phases which can help to discover the appropriate model to express the data and evaluate the performance of the selected model.

Different AI/ML work tasks have different evaluation metrics, and the same machine learning task will also have different evaluation metrics, each metric has different emphasis, e.g., classification, regression, ranking, clustering, recommendation, etc.

Make classification as an example, there will have at least four types of outcomes as follows:

**True Positives (TP)**: predict an observation belongs to a class and it actually does belong to that class;

**True Negatives (TN):** predict an observation does not belong to a class and it actually does not belong to that class;

**False Positives (FP)**: predict an observation belongs to a class but it does not belong to that class;

**False Negatives (FN):** predict an observation does not belong to a class but it does belong to that class.Three main metrics are used to evaluate or measure the performance of a classification model: **accuracy, precision, and recall**.

**Accuracy** measures how often the classifier makes the correct predictions, it is defined as the ratio between the number of correct predictions and the number of total predictions.

**Precision** measures the proportion of predicted positive results that are actually positive, it is defined as the fraction of examples (true positives) among all of the examples which were predicted to belong in a certain class (positive).

**Recall** measures how much the classifier can predict in an actual positive sample, it is defined as the fraction of examples which were predicted to belong to a class with respect to all of the examples that truly belong in the class.

## 6.2 Intermediate data

### 6.2.1 Intermediate data transfer optimization techniques

Intermediate data consist of large tensors computed by the first part of a split neural network. The following table provides some examples of intermediate data sizes for a few neural networks taking an input image of size 224x224x3, i.e., a tensor containing 150,528 values):

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Layer | Dimension | Number of values |
| squeezeNet | maxpool4 | 27x27x256 | 186 624 |
|  | maxpool8 | 13x12x512 | 79 872 |
| mobileNet v3 | layer 7 | 28x28x40 | 31 360 |
|  | layer 14 | 14x14×112 | 21 952 |
| VGG | layer 6 | 112x112x128 | 1 605 632 |
|  | layer 12 | 28x28x512 | 401 408 |

Some compression approaches (e.g., quantization, entropy coding, transformations) can be used to reduce the size of the transferred intermediate data and to adapt the split AI/ML operations between the UE and the network to changing conditions.

As a generic approach, the UE or the network endpoint selects a function among a set of compression and/or decompression functions built to adjust the characteristics of output intermediate data to the current network conditions or to meet the expected latency. For example, different functions can meet different bandwidth requirements.

## 6.3 Media data

## 6.4 Metadata

## 6.5 Existing formats for AI/ML models

### 6.5.6 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  } |

### 6.5.6 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]);  } |

### 6.5.7 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 6.5.7-1, 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).

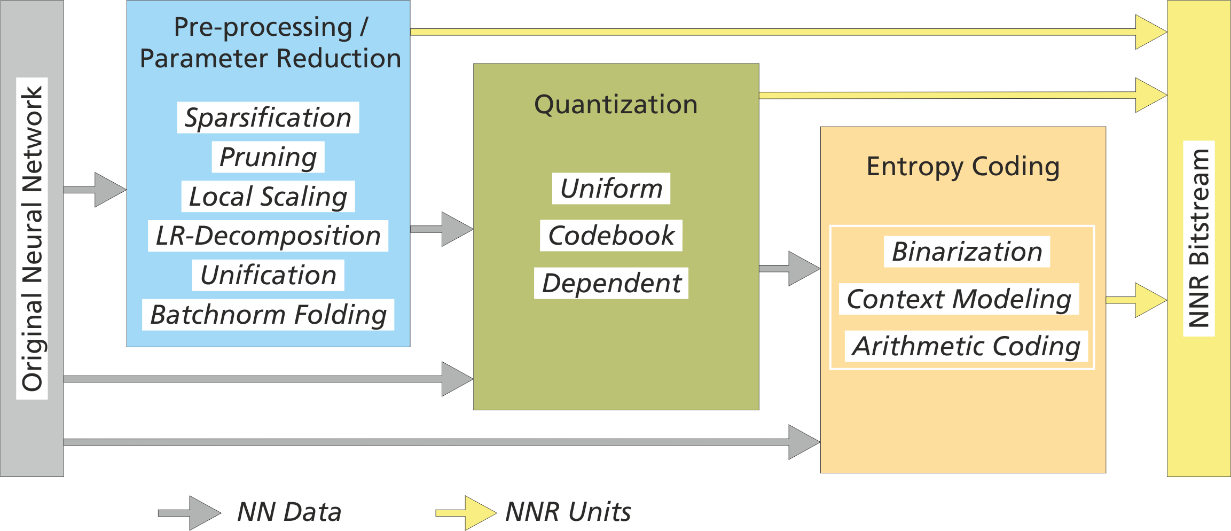
For access from a systems or application layer, NNC packages the NN data in neural network representation (NNR) units. NNR units that can carry different types of NN data: *NNR parameter set* and *NNR layer parameter set units* convey metadata and information related to the entire NN and individual NN 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.



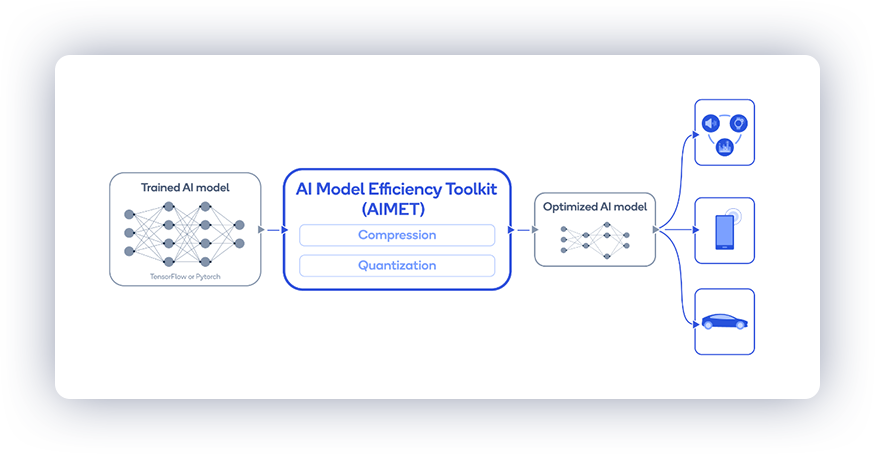
**Figure 6.5.7-1: Generation of a neural network representation (NNR) bitstream consisting of NNR units. Tools for pre-processing, parameter reduction, quantization, and entropy coding can be selected based on the complexity and compression requirements of a given use case.**

## 6.6 Existing optimization and compression tools for AI/ML models

### 6.6.1 AIMET library

Qualcomm has recently released the AI Model Efficiency Toolkit (AIMET). 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.

The following figure depicts the 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 [7].

# 7 AI/ML evaluation framework

Agreement @ SA4 #122 (S4-230144):

Defining an evaluation framework for AI/ML, including a set of anchor models and corresponding data sets, based on the use cases and scenarios identified in clause 4. The evaluation to include:

* Evaluation of different split points for the model and documentation of the intermediate data.
* Comparison of different checkpoints of the model to evaluate model updates.
* Comparison of compressed and non-compressed trained model and their accuracies.

# 8 Traffic characteristics

## 8.1 Complete/Basic AI/ML model distribution

## 8.2 Split AI/ML operation

### 8.2.1 Examples of split point references

#### 8.2.1.1 Feature Maps used in MPEG VCM (Video Coding for Machines) Track 1

The pipeline that is considered for track 1 on video coding for machine-only vision tasks is as follows:

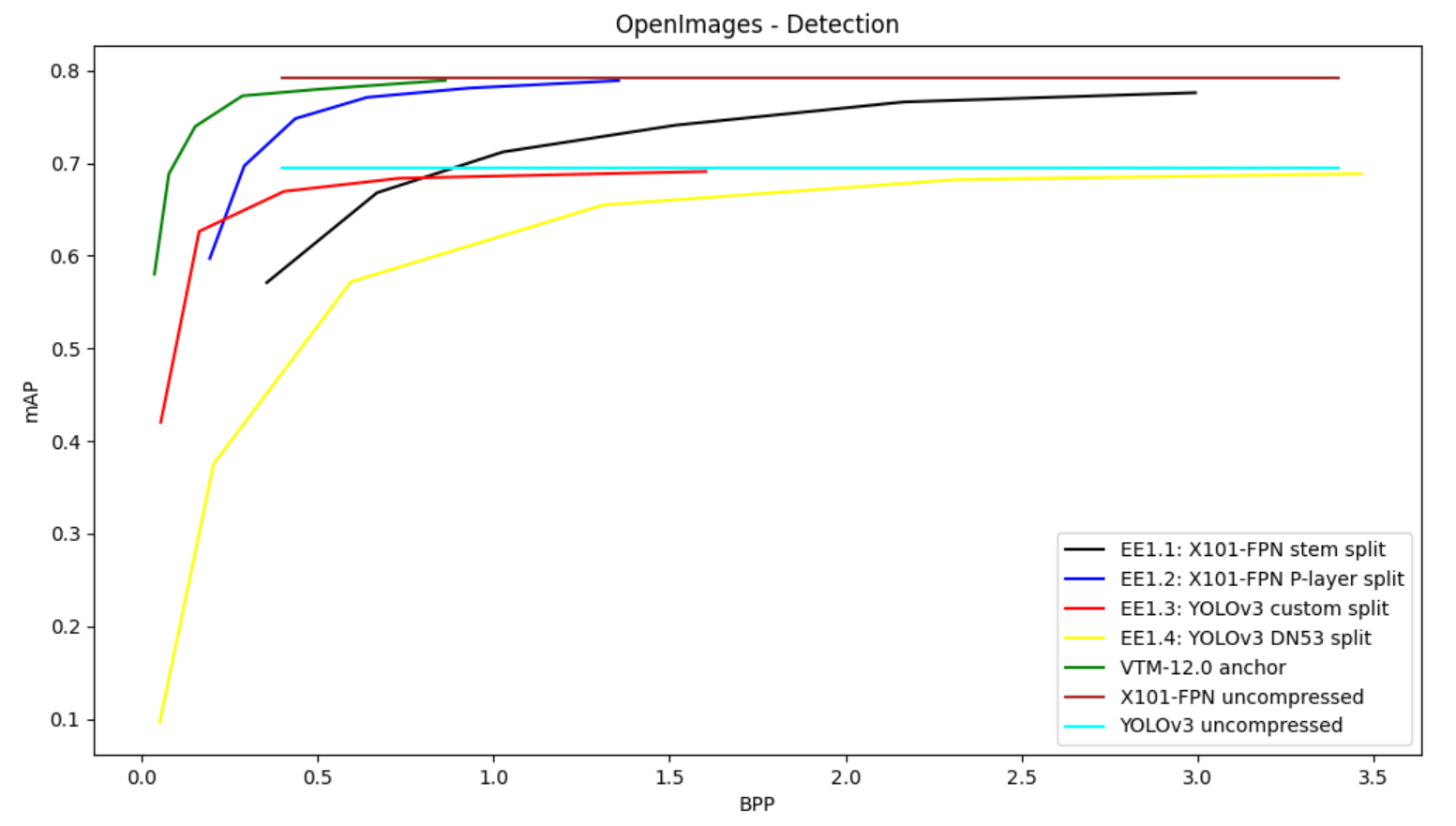


The video is first analyzed to extract the feature maps, which will be compressed by VCM. For the anchors, which are used for comparison purposes, the VVC video codec is then used to compress the feature maps. Some pre-processing maybe needed, for example to pack all channels of the feature map into a single atlas map. VCM proposals will be measure against this basic approach.

The group selected 4 popular networks, covering the typical computer vision tasks. The selected networks and the corresponding split points are summarized by the following table:

|  |  |  |
| --- | --- | --- |
| **Designation** | **Network** | **Split Point** |
| EE1.1 | X101-FPN | Stem layer split point |
| EE1.2 | P-layer split point |
| EE1.3 | YOLOv3 | custom split point (layers 75, 90, 105) |
| EE1.4 | DarkNet-53 split point |

The anchor results are described by the following figure:



## 8.3 Distributed/federated learning

# 9 KPIs

# 10 References

[1] 3GPP TR 22.874, Study on traffic characteristics and performance requirements for AI/ML model transfer in 5GS

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

[3] The Khronos NNEF Working Group, “Neural Network Exchange Format”, <https://www.khronos.org/registry/NNEF/specs/1.0/nnef-1.0.5.html>

[4] “Text of ISO/IEC FDIS 15938-17 Compression of Neural Networks for Multimedia Content Description and Analysis”, MPEG document N00080, ISO/IEC JTC 1/SC 29/WG 04, April 2021.

[5] Y.3179: Architectural framework for machine learning model serving in future networks including IMT-2020

[6] Agiollo A., et al., “Load Classification: A Case Study for Applying Neural Networks in Hyper-Constrained Embedded Devices” Journal of Applied Sciences, December 2021

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