**Source: Samsung Electronics Co., Ltd., [Qualcomm, InterDigital, Fraunhofer, Tencent]**

**Title: [FS\_AI4Media] Unified Evaluation Framework**

**Agenda Item: 9.7**

**Document for: Agreement**

# 1 Introduction

Merge of S4-230509, S4-230553, S4-230583\_r01, S4-230584\_r01, S4-230585\_r01, S4-230587\_r01.

Contents to be included into the Permanent Document v0.7, to also be considered as separate document at a later stage.

# 2 Unified Evaluation Framework.

1. 7 AI/ML evaluation framework

7.1 Introduction

The evaluation framework is designed to accommodate different scenarios for the different use cases for the usage and deployment of AI/ML over 5G networks. A scenario describes the evaluation for a specific use case. Use cases have been identified as part of the SA1 study and a selected subset is documented in TR26.927.

7.2 Scenario template

A scenario should provide the following information (aligned with TR 26.955, Annex A):

* **Scenario name** <give the scenario a catchy name>
* **Motivation for the scenario and its use case relevance:**

Why is the scenario relevant for AI/ML multimedia services? Under which of the following use cases does the scenario fall?

* Object Recognition in Image and Video
* Video Quality Enhancement in Streaming
* Crowd-Sourcing Media Capture
* NLP on Speech
* **Description of the scenario:**

This provides a description of the scenario addressing potentially the relation to the three AI/ML evaluation framework objectives, including AI/ML model split points, AI/ML model checkpoints and updates, and AI/ML model data compression. The description should be more specific than the use case description as provided in TR26.927. Predominantly the description should allow to develop a baseline solution.

* **Supporting companies and 3GPP members:**

a. This documents the 3GPP members that support this scenario in terms of providing the information, test material, test requirements and the characterization for the tests. For each of the identified necessities, a tick box is created in the template.

b. Preferably several 3GPP members are included in the support.

c. Cross-verification is preferably done by the supporters of the scenario

* **Anchor AI/ML DNN model(s) for the scenario:**

Give the name and details of the trained AI/ML DNN model(s) that will serve for building anchors for this scenario, as well as the data set used for its training. Such trained AI/ML models are not only limited to readily available base AI/ML models, but can also include models developed using transfer learning. There may be more than one candidate anchor AI/ML model for the scenario. As an example, details may include:

a. Base model used (including links to such base model)

b. Framework language used (e.g. TFLite, Pytorch)

c. Architecture/model type (e.g. CNN, RNN)

d. Number of layers

e. Number of parameters

f. Model size

g. Details of data set used for training

* **Testbed architecture and anchors**

Describe and detail the testbed architecture and anchors to be used for the scenario. The architecture and anchors should be based on the ones as defined in clause 7.4, with modifications matched to the scenario.

* **Test configuration factors, constraints and settings:**

Describe the test configuration factors, constraints and settings for the scenario. Depending on the nature of the scenario, examples are shown below.

AI/ML model split configuration factors, constraints and settings:

For scenarios considering the feasibility of AI/ML split points, many factors may contribute to the split point decision for the scenario, including those related to device/network status and conditions, as well those related to the AI/ML model used, such as its architecture and complexity. Possible split point decision factors may include:

|  |  |  |
| --- | --- | --- |
| Categories | Parameters | Details |
| **Devices Involved** | *CPU/GPU* | Device processor capabilities |
| *Battery* | Device battery status |
| *Heat* | Device heating / user health considerations |
| **Network** | *Cellular* | Network selection, bandwidth, latency |
| *Mobility* | Network handover and mobility |
| **Intermediate Data** | *Size* | Data transmission decision, data weights |
| *Type* | Video, Audio/Speech, Text, Binary etc. |
| **Model Type** | *Architecture* | CNN, RNN, GAN, LSTM, etc. |
| **User focus** | *APP KPI* | Latency Requirement , Service criticality |
| *Data Privacy* | Data transmission allowed or not |
| *Cost of hosting* | Deployment cost at cloud/server |

The scenario may also describe split point constraints, such as limiting split points to those that do not change the model topology and its parameters, splitting only at the layers of the AI/ML model, etc.

Compression or optimization constraints and settings:

For scenarios considering the compression or optimization of the AI/ML model, and/or the intermediate data (where applicable to split inference scenarios), describe the compression or optimization constraints and settings.

* **Feasibility/performance evaluation metrics and requirements:**

Depending on the scenario,feasibility and performance metrics may be either related to model performance, or to the test bitstream (the nature of which depends on the use case scenario).

List and describe the relevant feasibility/performance evaluation metrics for the scenario. A list of possible metrics is detailed in clause 7.5.

* **Test dataset(s) and scripts for the scenario:**

Describe and provide data sets that will be used for the evaluation of this scenario. This should include a description of the license, access procedure, and the dataset annotation format. Same test datasets may be used for similar scenarios falling under the same use case, as listed in 2.

Also provide scripts that will be used for performing the evaluation and calculating the metrics.

Further details are provided in clause 7.6.

* **Detailed test conditions:**

Provide the detailed test conditions, in particular the descriptions of the input and outputs of the task.

* **Interoperability considerations for the scenario:**

Interoperability considerations for the scenario may include those related to the delivery considerations for the AI model and other corresponding data (such as intermediate data), including delivery methods, protocols and packetization methods.

1. AI/ML model delivery formats, methods and pipelines: encapsulation formats for AI model data (if necessary), related to the delivery methods and pipelines which may be considered (e.g. download, streaming). This may be related to model update requirements and constraints.
2. AI/ML model optimization methods: methods of model optimization which are not considered under the evaluation methods described under the AI/ML model data compression evaluation defined.
3. Intermediate data compression, delivery formats, methods and pipelines.
4. Related to a and c above: streaming protocols such as TCP / UDP
5. Related to a and c above: packetization methods such as RTP
* **External performance data**

References to external performance data that can be added, for example other SDOs, public documents and so on.

* **Expected time plan for the scenario completion**
* **Additional information**

7.3 Prioritizing scenarios

Due to the complexity of this evaluation work, scenarios should be prioritized based on their feasibility within a reasonable time frame. A higher priority should be given to scenarios for which the use case is actual, i.e. already being deployed and used.

Priority should also be given to scenarios that are based on mobile phones and devices, compared to others based on e.g. automotive, surveillance, or UAVs (drones).

Finally, precedence should be given to evaluating the aspects and solutions that are considered in the SA1 study as documented in TR22.874. These are:

* AI/ML operation splitting between AI/ML endpoints
* AI/ML model/data distribution and sharing over 5G system
* Distributed/Federated Learning over 5G system

7.4 Testbed architectures and anchors

Unless proven otherwise, a common set of architectures is assumed for the evaluation framework, irrespective of the scenario.

The anchor architectures are as follows:

* Running inference completely on the device
* Receiving a compressed video (e.g. from the device), and running inference completely at the network and potentially sharing the inference results with the device.

These anchor architectures are depicted by the following figure:

 

In the figure, the left hand side represents the anchor for running the inference at the device side. The right hand side shows the architecture for the anchor where the inference is run on the network side. The anchor model for running on the device should be derived from the anchor model running on the network.

The derivation process may include:

* Quantization to match the device’s inference engine, e.g. converting the weights and inputs to fixed point or unsigned integers.
* Re-training of the converted model to optimize for the inference platform. This is allowed but should be accompanied by results without re-training.
* Conversion to an exchange format such as ONNX

The supported model libraries are PyTorch and Keras/Tensorflow2.

7.4.1 Split inference testbed architecture

The figure below shows an example testbed architecture for split inference related scenarios:



*Derivation of test metrics for scenarios related to AI/ML model split points*

7.4.2 Intermediate data testbed architecture

****

An architecture for evaluation of intermediate data is represented in the figure above. The model inference is split between a local and the remote compute node according to scenarios to define. The local to remote direction simulates an uplink communication while the remote to local direction simulates a downlink communication. Depending on the scenario, the sender of the intermediate data may be the local inference node or the remote inference node.

The testbed architecture includes the following main functional blocks:

* ***Anchor untrained model architecture(optional)***: A model to be trained with reference training data.
* ***Anchor training data(optional)***: Input data set and training parameters used to build a new reference trained model.
* **Anchor model (pre-trained)**: A model with a documented architecture with pre-trained weights. The model optimization (e.g., quantization) or compression is part of the reference trained model.
* **Reference framework/library:** For example, TensorFlow, Pytorch, etc.
* **Split points model configuration:** Configuration of selected split points for the set of models to evaluate.
	+ **Local:** Anchor model fully run on the local node.
	+ **Remote:** Anchormodel fully run on the remote node.
	+ **Split configuration:** Anchor model runs on the local and the remote nodes.
		- Selected Split points to evaluate.
* **Reference test data input:** For example, a reference picture or video sequence.
* **Inference nodes:**
	+ **Local inference node:** The local inference node emulates an end-device such us a UE.
	+ **Remote inference node:** The local inference node emulates a network node such as edge/cloud/5G CN Application server.
* **Data Delivery/Access.** This may include selection of different means for delivery and access of intermediate data:
	+ **Data encoding/decoding**: This includes for example serialization/deserialization, optimization, compression/decompression.
	+ **Uplink/Downlink communications:** The scenarios involve both uplink and downlink communications. The evaluation can consider different protocols to be used in the uplink and downlink, as well as real-world emulation constraints (downlink bandwidth vs. uplink bandwidth).

7.4.3 Model data testbed architecture

The figure below shows an example testbed architecture for AI/ML model data related scenarios:



*Derivation of test metrics for scenarios related to AI/Ml model data compression*

7.5 Metrics

Given that most scenarios that we’re dealing with in the scope of this study involve computer vision tasks, for model performance metrics, the evaluation framework should reuse existing metrics that are well-established in the research community. There exists different metrics depending on the type of task performed by the model.

For object classification tasks, the metrics are:

1. Accuracy: Accuracy is the simplest metric for evaluating classification performance. It measures the percentage of correctly classified objects out of the total number of objects in the dataset. While accuracy is easy to understand and compute, it can be misleading if the dataset is imbalanced, or the cost of misclassifying different categories is not equal.
2. Precision: Precision measures the proportion of true positives among all the objects that the model classified as positive. It is useful when the cost of false positives is high, and it is essential to avoid misclassifying objects.
3. Recall: Recall measures the proportion of true positives among all the objects that belong to the positive class in the dataset. It is useful when the cost of false negatives is high, and it is essential to detect all objects in the dataset.
4. F1 Score: The F1 score is the harmonic mean of precision and recall and provides a balanced view of the model's performance.

For object detection tasks, the metrics are:

1. Intersection over Union (IoU): IoU is one of the most commonly used metrics for evaluating object detection algorithms. It measures the overlap between the ground truth bounding box and the predicted bounding box. IoU is computed as the ratio of the intersection of the two boxes to the union of the two boxes. A higher IoU score indicates better object detection accuracy.
2. Precision and Recall: Precision measures the fraction of true positives (correctly identified objects) out of all predicted positives (objects identified by the algorithm). Recall measures the fraction of true positives out of all ground truth positives (objects that should have been identified). A high precision score indicates that the algorithm is correctly identifying objects, while a high recall score indicates that the algorithm is not missing any objects.
3. Average Precision (AP): AP is a commonly used metric for evaluating object detection algorithms. It measures the precision at different levels of recall and then averages them. AP provides a single number that summarizes the overall performance of the algorithm. A higher AP score indicates better object detection accuracy.
4. F1 Score: The F1 score is the harmonic mean of precision and recall. It provides a single number that summarizes the overall performance of the algorithm. A higher F1 score indicates better object detection accuracy.

For object tracking tasks, the metrics are:

1. Intersection over Union (IoU): IoU is also commonly used for evaluating object tracking algorithms. In this case, it measures the overlap between the ground truth bounding box and the predicted bounding box for each frame in the sequence. A higher IoU score indicates better object tracking accuracy.
2. Precision and Recall: Precision and recall can also be used to evaluate object tracking algorithms. In this case, precision measures the fraction of frames where the algorithm correctly identified the object, while recall measures the fraction of frames where the algorithm correctly tracked the object.
3. Mean Average Precision (mAP): mAP is a commonly used metric for evaluating object tracking algorithms. It measures the average precision at different levels of overlap between the ground truth and predicted bounding boxes over the entire sequence. A higher mAP score indicates better object tracking accuracy.
4. Tracking Precision (TP) and Tracking Recall (TR): TP measures the fraction of frames where the predicted bounding box overlaps with the ground truth bounding box by a certain threshold, while TR measures the fraction of ground truth bounding boxes that were successfully tracked. A high TP score indicates that the algorithm is accurately tracking the object, while a high TR score indicates that the algorithm is not losing track of the object.

For other non-object related tasks, examples model performance metrics may include:

* Regression Model Metrics (MSE, MAE)
* Ranking Model Metrics (MRR, DCG, NDCG)
* Statistical Model Metrics (Correlation)
* Computer Vision Model Metrics (PSNR, SSIM, IoU)
* NLP Model Metrics (Perplexity, BLEU score)

For split inference and model compression related scenarios, other feasibility/performance metrics that should also be considered are:

* Video quality: depending on the scenario, the input or output video quality should also be documented. For example, a video super resolution scenario has to evaluate the quality of the resulting video. For the tasks, the impact of the quality of the input video on the accuracy should also be evaluated.
* Complexity: complexity of the entire process, including video compression and decompression, model compression and decompression (where relevant), and inference process.
* Bitrate: the total bitrate needed for performing the task. This may be 0 for the device anchor. For the network anchor, this includes the video bitrate for the uplink and the bitrate for sharing the task results back to the device. For split inference related scenarios, this should include the intermediate data bitrate.
* Split model size: model size and comparison ratio of the test split model to be delivered (compared to anchor model)
* Intermediate data size or bitrate: a comparison ratio of the intermediate data to be delivered (compared to the data size or bitrate of the relevant data from the anchors)
* Compressed model size: the compression ratio of the compressed model compared to the original reference model.
* Compressed intermediate data ratio: compression ratio of the compressed intermediate data bitstream compared to the original intermediate data bitstream
* Latency: the latency requirements for each scenario must also be taken into account to evaluate the feasibility of the proposed solutions, in particular for split inference scenarios, such as:
	+ Inference latency metrics
		- local inference time
		- Remote inference time
		- Total local and inference time
		- End to end latency
	+ Other latency metrics
		- Encoding/decoding time.
		- intermediate data delivery time
* Resources metrics:
	+ Computing power consumption on node
		- * CPU time
			* GPU time
	+ Memory usage
	+ Energy consumption

7.6 Datasets and scripts

It is recommended to build a docker container that comes with the necessary scripts for downloading the models and datasets, and running the evaluation for each agreed scenario. The Dockerfile should be hosted on a publicly accessible location to all 3GPP members. As example for software management refer to TR 26.955, Annex E.

Potential openly accessible video datasets are:

* YouTubeVIS: [Video Instance Segmentation - YouTube-VOS](https://youtube-vos.org/dataset/vis/)
* SFU-HW-Objects-v1: [SFU Multimedia Lab](http://multimedia.fas.sfu.ca/data/)
* TVD: [Tencent Video Dataset (TVD) - Tencent Media Lab](https://multimedia.tencent.com/resources/tvd)

For some of the scenarios, companies may be asked to provide a suitable annotated data set to perform the evaluation. This may follow the principle in Annex B of TR 26.955 as well as the test sequence collection in Annex C of TR 26.955.

We offer to collect the data sets, anchors, etc here: https://dash-large-files.akamaized.net/WAVE/3GPP/AIML.

7.7 AI/ML frameworks and libraries

An AI/ML framework brings a set of services which are interfaces, libraries or tools. They are used to create models, train them and/or to infer input data and deliver a prediction.

Hereafter is a short list:

1. TensorFlow
2. PyTorch
3. Caffe
4. Keras
5. MXNET
6. Darknet

Some frameworks are especially designed for on-device (Mobile Phones) deep Learning, we may present the two main ones:

1. TensorFlow Lite [1]
2. PyTorch Live [2]

**Note**: Keras is running on top of TensorFlow, and both together provide a high-level APIs to make a more user-friendly framework. For the rest of the document TensorFlow and Keras frameworks are considered as one entity noted TensorFlow/Keras.

AI/ML frameworks can be completed and enriched with libraries, for example to provide optimization and compression tools such as:

* NNC : clause §6.5.7
* AI Model Efficiency ToolKit (AIMET) clause §6.6.

Both libraries support TensorFlow/Keras and PyTorch environments.

7.7.1 Framework popularity

PyTorch and Tensorflow/Keras are the two major and most popular frameworks for Deep Learning.

PyTorch appears significantly more in academics as shown in the next graph [4]



On the other hand, Tensorflow is much more popular in industry.

The TensorFlow eco-system comprises some deployment-oriented applications like TensorFlow Serving and TensorFlow Lite for AI/ML application to be deployed on cloud, edge, server, mobile or IoT devices.

PyTorch has filled the gap by proposing TorchServe [5] and PyTorch Live [2].

7.7.2 Detailed framework characteristics

Framework or library tools available (compression, quantization etc.):

* TensorFlow and Pytorch natively support optimization and quantization tools.

Hardware accelerator support:

List of tools for optimizing the ML models.

* It is very likely that the model performance will be evaluated with various processing conditions, being CPU, GPU, TPU or others like DSP.
* TensorFlow/Keras and PyTorch already integrate such capabilities:
	+ TensorFlow/Keras GPU or TPU usage in respectively [6] and [7]
	+ PyTorch GPU or TPU usage in respectively [8] and [9]

Supported models.

Natively both frameworks TensorFlow/Keras and PyTorch integrate many pre-trained models, this is described in document “models for evaluation”. If the model is not available, it can be reconstructed from its known architecture and trained.

A list of pre-trained model support is proposed for keras in [10], and for Pytorch in [11].

Split function

Splitting functionality shall be evaluated to point out the benefits it can bring to the 5G system (latency, energy, privacy), but also to measure and characterize the intermediate data. Therefore, the framework shall offer APIs/functions to split some models. This function is already available in TensorFlow/Keras framework as described in doc Split scenarios for evaluation and TensorFlow based split evaluation platform.

Mobile or on-device versions

Both PyTorch and TensorFlow/Keras have their own mobile solutions TensorFlow Lite [1] and PyTorch Live [2].

Language

Both PyTorch and TensorFlow/Keras are Python based.

TensorFlow supports additionally JavaScript, C++ and Java.

Supported formast for AI/ML models

PyTorch and TensorFlow/Keras support Open Neural Network eXchange (ONNX) and Neural Network Exchange Format (NNEF).

* ONNX: Tensorflow models (including Keras and TensorFlow Lite models) can be converted to ONNX [12]. PyTorch models can be exported to the ONNX format [13]. ONNX support tools for porting PyTorch model into TensorFlow or vice-versa.
* NNEF: supported by Khronos and designed to support both PyTorch and TensorFlow. NNEF tools can convert trained models from/to ONNX format [14].

7.8 AI/ML models

There may be several cases for the availability of AI/ML models:

1. Pre-trained models available from the AI/ML frameworks and libraries
2. Pre-trained models not available from the frameworks but from an external source, for instance GitHub
3. Non-trained models
4. New models

Case 1) can be illustrated by the ResNet50 model which is available from both PyTorch and TensorFlow/Keras frameworks.

Case 2) can be illustrated with the EDSR model, where the model authors proposed a PyTorch implementation of their model which is available from a GitHub repository.

Case 3) is where proponents want to perform experiments from a well-known model and retrain it with a specific dataset corresponding to the use case to be evaluated. For example, YOLO or AlexNet are not available in TensorFlow/Keras.

Case 4) is for proponents who propose new model architecture.

For case 2), the proponent shall share the information on how to get the model, and how to run the experiments.

For cases 3) and 4), the proponents shall share the AI/ML model data (dataset, hyperparameters, etc.…) and describe how they train the AI/ML model.

7.8.1 Model characteristics

Several characteristics that may define an AI/ML model:

* **Model Popularity within scientific community:** The model is often cited in scientific papers and as such is recognized as an efficient model by many frameworks, in particular the frameworks listed in doc “Frameworks for evaluation”. ResNet50 or MobileNet are good examples of such models.
* **Availability as a pre-trained version:** Pre-trained version of the model as proposed by the framework should be preferred. Untrained models are possible under conditions above.
* **AIML model Task:** It depends on the use cases and scenarios to evaluate. The preferred domain is computer vision, which include object detection, image recognition, segmentation, pose estimation, image classification.
* **Format:** By default, the model format is the framework model format to be supported, for example ONNX and/or NNEF.
* **Splitability:** Ability to split/partition the model in two subsets. Some models may be easier to split than others depending on the complexity of the relations between the layers.

7.8.1 Pre-trained model repositories

ModelZoo [15] is a popular repository providing open-source deep learning code and pre-trained models for a range of different frameworks (e.g., TensorFlow, Pytorch) and for different model tasks categories (e.g. computer vision, NLP).

TensorFlow proposes a collection of pre-trained models in [16], [17] and [18].

Keras Applications [17] are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

7.9 Scenarios

Individual clause for each scenario.

7.10 References

1. <https://www.tensorflow.org/lite>
2. <https://playtorch.dev/>
3. <https://github.com/quic/aimet>
4. <https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/>
5. <https://pytorch.org/serve/?ref=assemblyai.com>
6. <https://www.tensorflow.org/guide/gpu>
7. <https://www.tensorflow.org/guide/tpu>
8. https://pytorch.org/docs/stable/notes/cuda.html /GPU
9. https://pytorch.org/xla/release/2.0/index.html XLA/TPU
10. <https://modelzoo.co/framework/keras>
11. <https://modelzoo.co/framework/pytorch>
12. <https://onnxruntime.ai/docs/tutorials/tf-get-started.html>
13. <https://pytorch.org/docs/stable/onnx.html>
14. <https://www.khronos.org/api/nnef>
15. <https://modelzoo.co/frameworks>
16. <https://github.com/tensorflow/models/tree/master/official>
17. <https://keras.io/api/applications/>
18. <https://tfhub.dev/>