Edge Intelligence: Enabling Distributed ML Applications in Cable Networks

A technical paper prepared for presentation at SCTE TechExpo24

Karthik Sundaresan
Distinguished Technologist and Director of HFC solutions
CableLabs
k.sundaresan@cablelabs.com

Table of Contents

List of Figures

1. Introduction

Machine learning (ML) services are commonly deployed in centralized data centers. However, for cable network applications, such as identifying anomalies within the cable network using DOCSIS® network performance data, a centralized model can lead to delays in processing and classifying data, hindering quick problem identification for operators. Implementing edge intelligence offers a solution to this issue, combining centralized training with edge inference.

During the learning phase, large amounts of data is utilized to calculate the weights and biases for training the model, necessitating high-performing machines easily found in centralized data centers. Once the learning phase concludes and the ML network is trained, the model can be deployed for inference, executing on devices with lower computational and memory capabilities at the edge. Models can be deployed on edge devices such as Cable Modems (CMs), gateway devices, or Access Points (APs), with enhancements like model compression and inference acceleration. While edge devices must possess sufficient power for specific tasks, embedding ML capabilities into such devices is achievable using certain class of algorithms.

This paper proposes a model to facilitate distributed edge intelligence on DOCSIS network equipment, including CMs and gateways, and potentially extending to other devices like Distributed Access Architecture (DAA) nodes or amplifiers. It aims to develop an architecture to support this deployment model in a DOCSIS network, detailing the process of downloading ML models to edge devices, implementing necessary security mechanisms, and providing the required application programming interfaces (APIs) to enable such functionality.

2. Machine Learning at the Edge

Machine learning is currently widely used in a variety of applications, including PNM, profile management and cable network health detection. End devices such as CMs and RPDs, are generating data that need to be analyzed in real-time using deep learning or used to train deep learning models. However, deep learning inference and training require substantial computation resources to run quickly. Using the compute at the core network, is a viable way to meet the high computation requirements of ML/deep learning processes.

The deployment of machine learning applications at the edge presents a significant opportunity for several HFC network scenarios. The benefits include the lower latency associated with performing on-device training and inference close to the data sources. However, the limited computational, memory, and energy resources, differences in hardware, of the edge devices in a cable network (e.g. CMs, RPDs, Amplifiers) pose a significant challenge to performing heavy learning tasks on them.

2.1. ML Phases: Training and Inference

Training is the first phase for an ML application. Training involves a process of teaching the model examples of the desired inputs and outputs, or both, and helping the model learn the main characteristics of each example over a large number of samples. Inference is the process that follows ML training. Inference is the process that a trained machine learning model uses to draw conclusions and make decisions on brand-new data. The more trained a model is, the better its inference results will be.

Inference allows machine learning models to be used in real-world applications, where the goal is to apply the knowledge gained during the training phase to make useful predictions or decisions on new data. This contrasts with the training phase, where the model learns the underlying patterns and relationships from a dataset.

To get to the point of being able to say as an example, identifying ingress in cable signal, machine learning models go through the process of training. For the cable RF ingress detection, the MSO developers may develop a model, by showing the model thousands data samples of ingress. The data may have been labeled by expert RF engineers who have looked at the data samples manually. Eventually, after enough training, the model will be able to identify ingress or other trained artifacts on its own.

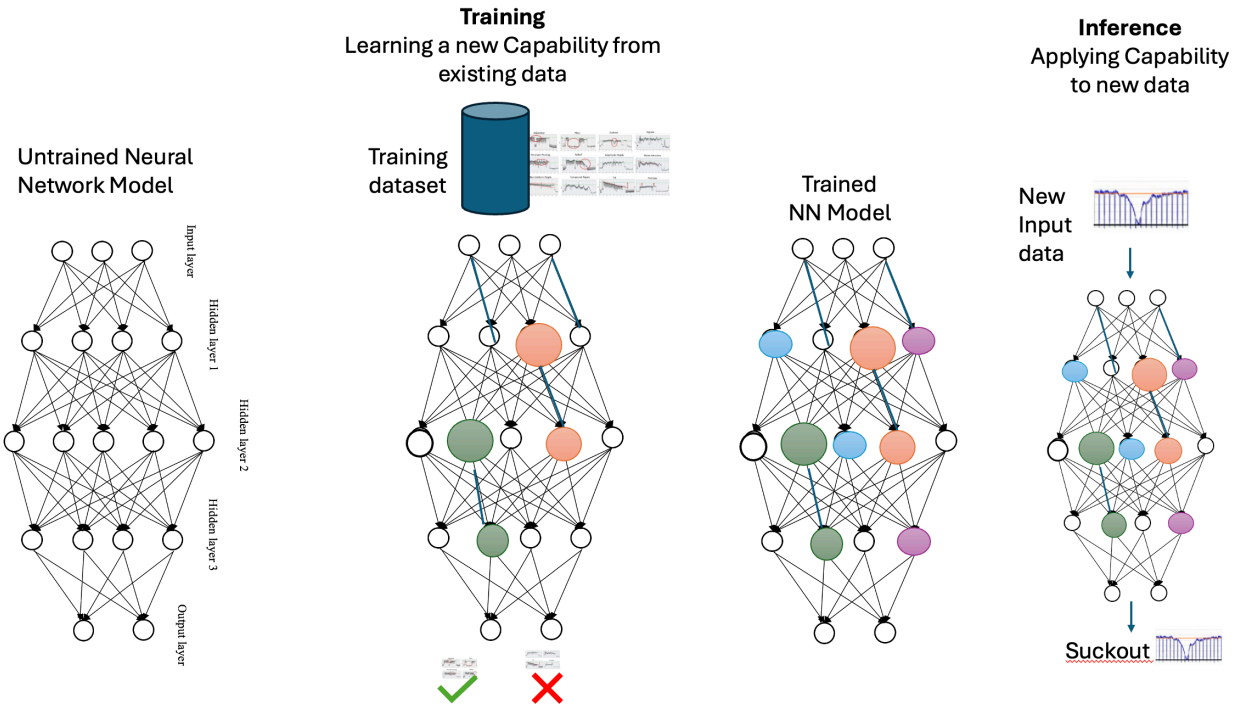

Figure 1 – ML Training and Inference

Inference refers to the process of using a trained machine learning model to make predictions or decisions on new field data. It involves taking input data and passing it through the trained model to generate an output or prediction. In the inference process for any new(previously unseen samples), applies the model's learned patterns and parameters to the input data, to generate an output or prediction based on the model's internal logic and decision-making.

2.2. Feature Engineering

Deep learning models, particularly those involving convolutional neural networks and recurrent neural networks, have the capability to automatically learn and extract features from raw data. This is one of the advantages of deep learning over traditional machine learning methods, which often require extensive feature engineering. However, the need for feature engineering in deep learning can vary depending on the context and type of data.

While deep learning models reduce the need for extensive feature engineering by automatically learning features from raw data, especially in domains like image or text processing, there are still scenarios where feature engineering can be beneficial. This is particularly true in cable domain-specific applications, and smaller datasets. The balance between manual feature engineering and leveraging the automatic feature extraction capabilities of deep learning models depends on the specific problem and dataset characteristics.

2.3. ML at the Edge

Machine-learning algorithms can be run at the device or local level, closest to the components collecting the data, this would be combining Machine learning with any available edge compute. ML at the edge can be thought of as a method for lowering dependency on cloud infrastructure and networks by allowing devices to analyze data locally, at the device level using machine learning techniques. The ability of specific data to be processed locally limits the data that needs to be sent up to the cloud and enables real-time data processing and reaction to important events. Of course, Edge devices continue to transmit data to the cloud as needed for various purposes such as centralized learning.

Machine learning on the edge is key to enabling a new suite of autonomous system applications on the Cable Network. The move from the traditional centralized HFC architecture to distributed architectures in recent years means that new ML deployments can be power efficient and reduce latency for ML inference.

The efficiency and accuracy of the inference process are crucial, as it determines how effectively the trained model can be deployed and utilized to solve practical problems. Techniques like optimizing model architecture, quantization, and model compression can be used to improve the inference performance. The main approach is to train large and accurate models on high-performance machines in the cloud and then use compression techniques, such as low-rank approximation, knowledge distillation, pruning, and parameter quantization, to reduce model size. However, smaller models often result in lower accuracy, thus the tradeoff between accuracy and costs must be carefully considered. [ScaleMLEdge]

2.3.1. Related research

There is previous work in this area. The [EdgeML] library provides a set of open-source algorithms for building machine learning models that can run directly on edge devices, with much lower memory requirements than traditional ML algorithms. Other efforts focus on trained models, such as tree-based classifiers [ResEffML], k-nearest neighbors (kNN) classifiers [ProtoNN], and recursive neural networks (RNNs) [FastGrNN], can be loaded onto edge devices, such as IoT devices and sensors, to make fast and accurate predictions. The [TinyML] project and Tensorflow Lite Micro [TensorFLMicro] focus on optimizing deep learning inference for edge devices with limited memory, such as microcontrollers, enabling the efficient deployment of ML applications in the context of edge.

2.4. ML Problems in the Cable Network

Data sourced from the cable network is a ripe source of information for the operator to analyze and Ultimately gain insights and knowledge from. Proactive network maintenance (PNM) functionality in HFC networks, yields a lot of downstream and upstream spectrum data. Operators can manually look at these data plots and identify issues in the network. This is of course not scalable with millions of samples across millions of devices in the network. This is a problem which can be more easily solved using machine learning. The basic idea is to create models from a set of training data which has been labeled by subject matter experts. Once the model is accurate then it can be used in automatically identifying events in the new input data. See [AcData], [AppML] [MLPNM] [DetClasOFDMA] for other related problem statements for data analysis challenges in the access network.

Below are some examples of problems that could be solved via machine learning. Ingress Identification is an important one While some sources of interference are well known beforehand, deployments have encountered and identified additional interference types.

In the Upstream, MSOs see VHF over-the-air (OTA) ingress. [DetClasOFDMA], a common ingress sources in OFDMA deployments. Depending on the location of TV Transmitters within a geographical

area, one or more channels may be impacted. Analog Modulator impairments are narrow band ingressors caused by older devices, or the wrong connector on an older set-top box connected to an outlet in the home. RFoG impairments are caused by customers who have disconnected their service but are still connected to the network. *Figure 2 source:[DetClasOFDMA]*

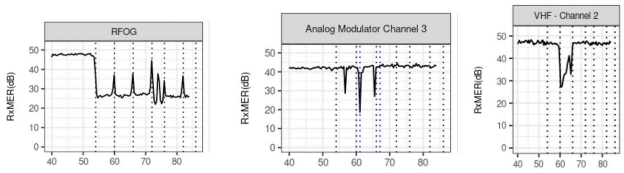

Figure 2 – Example Upstream Interefence

In the Downstream, the Rolloff is an impairment characterized by a gradual, non-linear, exponential looking decrease in amplitude and power. Rolloff maybe caused by older passive components not rated beyond a certain frequency. Tilt is an impairment characterized by amplitude differences between higher and lower frequencies, this can be a positive or negative slope across the spectrum. There are many other plant/RF issues shown below which can be solved by machine learning algorithms.
Figure 3 source:[PNMPractices]

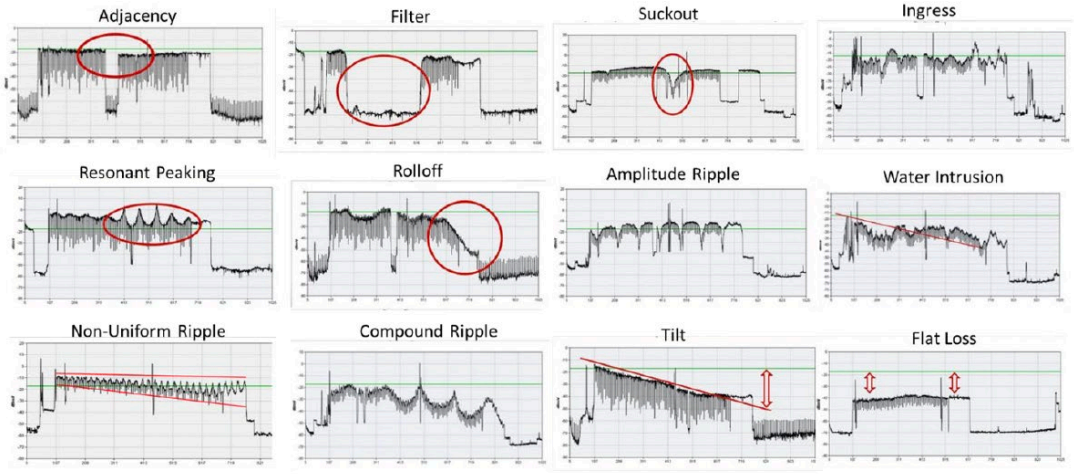

Figure 3 – Example RF interference Downstream

In a full band capture of the cable modem RF spectrum one can identify various ingress sources as shown in the figure below. *Figure 4 source:[PNMPractices]*

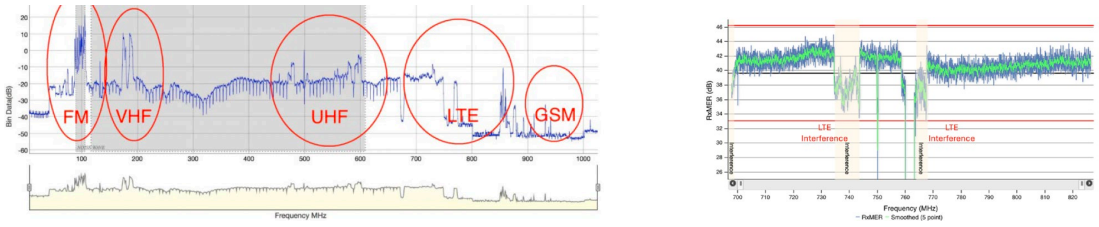

Figure 4 – Wideband Spectrum Analysis of RF Spectrum

For each of the examples shown above machine learning can be applied to identify these issues in the plant data, giving the operators knowledge about their networks and how to effectively manage and maintain the health of the network.

3. Machine Learning considerations

3.1. Training & Inference

Neural Networks training starts out with the forward propagation calculation. As Figure 5 illustrates, after forward propagation, the results are compared against the known/correct answer to compute an error value. A backward propagation phase propagates the error back through the network's layers and updates their weights using gradient descent to improve the network's performance at the task it is trying to learn. It is common to batch hundreds of training inputs (e.g. RxMER samples with ingress issues for an RF ingress detection network) and operate on them simultaneously during NN training in order to prevent overfitting and, spread the loading weights across many inputs, increasing computational efficiency.

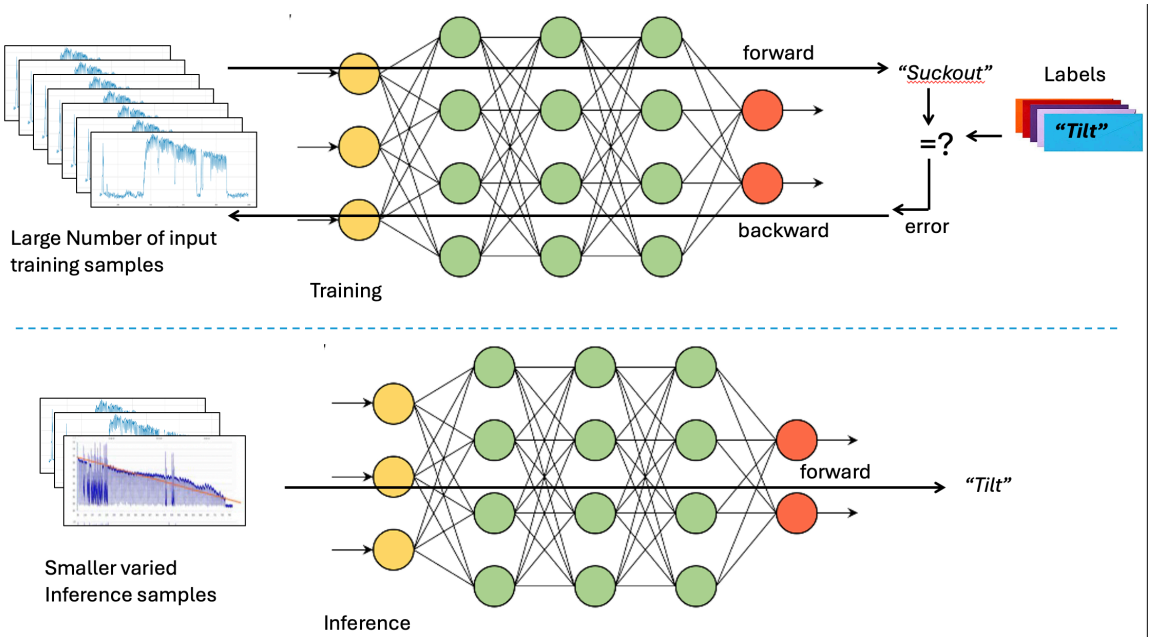

Figure 5 – Training and Inference of Neural Networks

For inference, which goes through only the forward propagation calculation. the performance goals are different. To minimize the network's end-to-end response time, inference typically batches a smaller number of inputs than training, as services relying on inference to work (for example, a cloud-based RF ingress detection pipeline) are required to be as responsive as possible. In general, workload for training is higher than for inference.

3.2. GPUs vs CPUs

Deep neural networks (DNNs) and convolutional neural networks (CNNs) demand substantial computational power, particularly during the training phase. The training process involves feeding inputs through the network to produce activations, which then reach the output layer. The resulting output is compared to the correct answer, and an error is computed for each unit in the output layer. This error is backpropagated through the network, adjusting each connection weight incrementally. Consequently, training involves a forward pass to generate outputs and a backward pass to propagate error information and update weights. When the network is deployed for inference, only the forward pass is used.

Neural Networks can have thousands to over millions of parameters that need adjustment through backpropagation and require a large amount of training data to achieve high accuracy, often necessitating

hundreds of thousands of input samples to undergo both forward and backward passes. Due to their structure, neural networks are inherently parallel, making them well-suited for GPUs, which offers significant speed improvements over CPU-only training. Various benchmarks have demonstrated substantial increases in training speed when using GPUs compared to CPUs.

ML inference is the process of using a trained ML model to make predictions or decisions based on new data. While ML training is a compute-intensive task that benefits significantly from the parallel processing power of GPUs, inference tasks can often be run efficiently on CPUs, especially when optimized properly. This is due to the fact that inference tasks generally require less computational power compared to the training phase.

Some of the CPU benefits for inference include cost, availability and optimized software libraries. CPUs are generally less expensive than GPUs, both in terms of upfront costs and operational expenses. This makes CPU-based inference an attractive option for cable operators looking to deploy ML solutions without the heavy investment required for GPU support in the edge devices like CMs and RPDs. CPUs are ubiquitous and available in virtually all edge devices in the Cable Network making it easier to deploy and scale ML applications at the edge without being limited by the availability of GPU resources.

Advances in software libraries and frameworks have improved the efficiency of running ML inference tasks on CPUs. There are many libraries e.g. Intel's oneDNN (part of oneAPI Deep Neural Network Library) [IntelOneDNN] and OpenVINO toolkit [OpenVINO], Microsoft's Embedded Learning Library [ELL] have been optimized for high performance on CPUs, making it feasible to achieve near-GPU performance for certain inference tasks. To achieve optimal performance on CPUs, ML models and inference code may need to be specifically optimized for CPU architecture. This can include leveraging specific software libraries, adjusting batch sizes, and tuning model parameters, which may require additional development effort.

Performance Limitations: While CPUs can be efficient at handling ML inference tasks, GPUs still offer superior performance for complex models and large-scale applications due to their parallel processing capabilities. Therefore, the choice for model training is almost always GPU, for many of requirements of an application, including the model complexity and latency requirements. GPUs also have an edge in terms of performance per watt for high-intensity computing tasks. This aspect is crucial for large-scale deployments where energy consumption directly impacts operational costs.

While GPUs are the main choice for training ML models and handling complex inference tasks, CPUs are a viable and cost-effective alternative for many applications. By carefully evaluating the specific needs of their ML applications and optimizing their models existing CPUs in devices can be leverages to reduce costs and enable new applications. As software libraries and CPU technologies continue to advance, the gap between CPU and GPU performance for ML inference is expected to narrow, further enhancing the attractiveness of CPU-based inference solutions.

3.3. Separate Training from Inference

The main idea, for enabling distributed ML applications in cable networks, is to separate the training part of the process from the inference part of the process. The ML training process needs more compute power and large sets of input data to build a successful model. This needs powerful GPUs and large data servers , and data from a large number of devices, to successfully complete the training process.

The inference process, i.e. making an actual classification decisions, using a trained model, based on a single input datapoint, is much less process intensive and can be run at the edge of the network. In case of the HFC network, this can be at the cable modem(CM), the RPD or potentially even within an amplifier.

While working with many of the opensource machine learning libraries, one can save the trained models in a file (serialization) and restore them (deserialization) in order to reuse them to compare the model with other models, and to test the model on new data.

CPUs are well-suited for small to medium-sized models where the inference latency meets the application requirements, applications with low request rates, where the cost savings of CPUs outweigh the need for GPU support and for deployments where minimizing operational expenses is a priority, and the slightly lower performance of CPUs is acceptable. CPUs in the HFC Edge should be able to handle many of the simpler inference tasks. We are working with a few applications where a trained ML model is being made to run inference on small platforms like a RaspberryPi4, these results will be noted in a future paper.

4. Architecture to Enable Intelligence at the Edge

4.1. Cable Industry Distributed CCAP Architecture

Now that we have introduced the idea of separating the training process from the inference process and running them at different locations let's take a deeper look into the architecture within the cable access network.

Traditional HFC networks have used analog optics to carry signals downstream, and either similar analog optics or digital return systems that act to digitize the return spectrum. With DCA/RPHY, new Physical layer modules are developed into node housings. The major advantages of RPHY include standardized, digital optical links, typically enabling 10GbE transport from the facility into the cable access network. Improved reach and wavelength efficiency of digital fiber versus analog, Fidelity gains (MER) that coincide with the removal of analog optical link degradation, physical scalability in the inside plant with the removal of RF cabling and combining networks are some of the benefits.

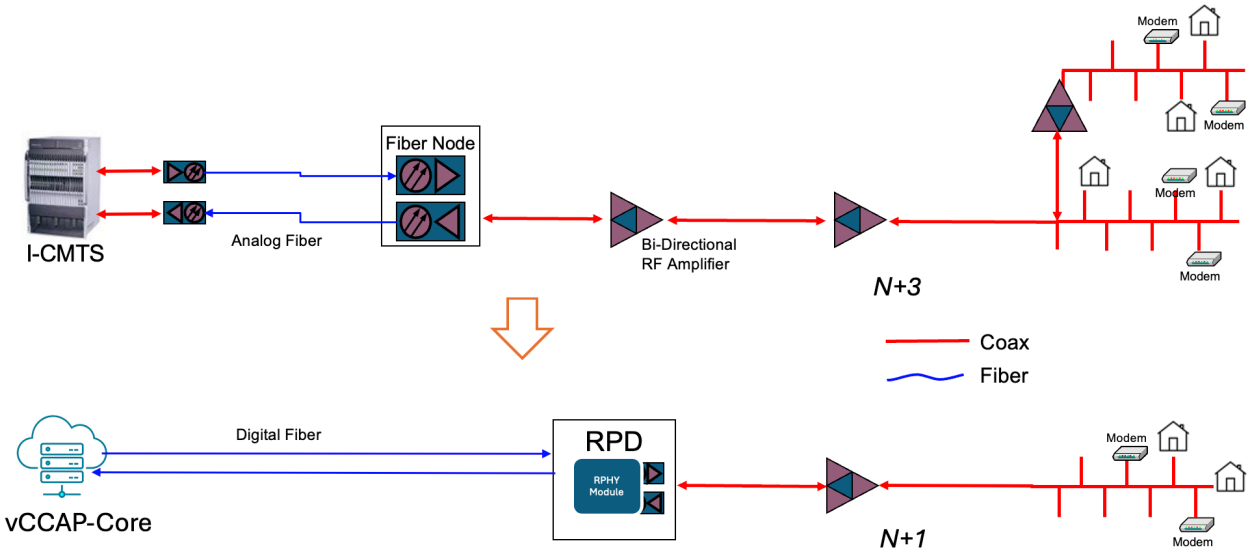

Figure 6 – Integrated CMTS vs Distributed CMTS Architecture

Centralized functions of the DOCSIS CMTS are suited to being implemented in software and to run on generic servers/hardware, giving better scalability over time and adaptability for different deployment scenarios. The CMTS function in the headend, now known as the CCAP-Core, having been separated from Physical layer functionality, includes packet processing, switching, storage, and scheduling of

network resources. With today's compute power and the ability of software systems to deliver real-time services, a purpose-built DOCSIS machine is no longer required to provide this functional capability. A virtual CMTS software that is purpose built to run on commodity servers, as a vCMTS platform. A network interface card (NIC) attached to a switch connects the server to an R-PHY node.

4.2. Edge ML Architecture

Typically, an operator also hosts a lot of compute power in a centralized location perhaps collocated with the network operations center. This centralized MSO cloud infrastructure support various applications run by the MSO to provision, configure, monitor and manage the network. This infrastructure provides the operator the opportunity to host different services in a virtualized environment deploy services faster and automate a lot of the network management operations.

The centralized MSO cloud is a location where a lot of the data from the cable network gets collected. This could be file uploads from the cable modem, streaming telemetry from the RPDs, and other data collection and logging from a variety of network devices.

With the amount of computer resources available the training portion of a machine learning application is well suited to be run at the centralized location. As data is collected from each of the CMs and RPD's in the network they get stored within a data lake in the centralized location. From here the data is fed into the machine learning training VMs which need GPU support. Once the training process is complete with oversight from HFC engineers and data analysts, the model is ready to be used in an application.

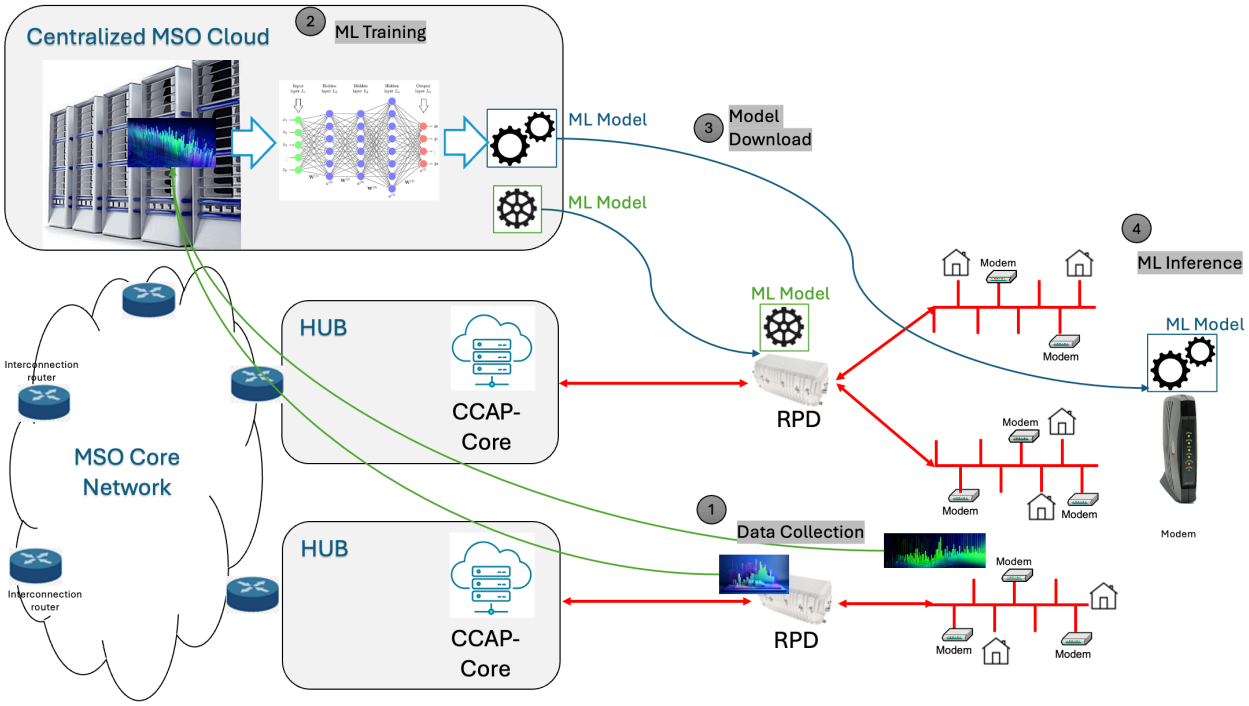

Figure 7 – Edge ML Architecture: Learning in the cloud, inference at the edge

Now instead of running this application at the centralized cloud location, the idea is to deliver this model into the edge devices, in our case this would be the CM, the RPD and potentially the amplifiers. The sections below define some methods for delivering this model down to the edge devices. Once the models

have been verified and validated the edge device can use those models within the machine learning for addiction applications. An example could be identifying ingress in the RF which say an application running on the modem would collect RxMER samples for a channel and send those through the model to identify ingress and then report it to the operator.

The edge ML architecture consists of the following high level steps: (also shown in Figure 7)

1. Data collection from the network to a centralized location
2. ML model training with human oversight and creating compact models that can be run remotely
3. Downloading the trained models to the edge devices in a secure fashion
4. Running the model within a particular application to making prediction /classification decisions

The edge device (CM/RPD) needs to be able to evaluate the trustworthiness of a pre-trained model. The question is how an operator builds a trustworthy dissemination protocol for sharing the pretrained ML models to the edge devices.

4.3. Control Mechanisms

This section talks about methods to download ML models into a cable modem. There need to be control mechanisms to enable the ML Model download. To keep compatibility with existing network management methods, one can imagine SNMP based control mechanisms on the CM with some new objects to initiate HTTP or TFTP based file download mechanisms. Alternatively, if the specific ML applications are identified and standardized within the specification one can also envision the Model download to be part of the MAC Management messages and under control of the CMTS which can be updated by the MSO/ML training entity. The whole download process can be built into DOCSIS MAC Management Messages. The below sections assume an HTTP or TFTP download of the model files to the cable modem.

For the RPD, the CCAP Core can be instructed via SNMP or other methods (e.g., CLI) to order the RPD to perform the software upgrade at any time. From the perspective of the RPD, the software upgrade is initiated by Principal Core via GCP software update option. This control mechanism would need to be extended to support for downloading trained ML models. This looks to be a logical extension to the functionality.

4.4. Download Mechanisms

Current DOCISIS CMs and RPDs are capable of being remotely reprogrammed in the field via a software download over the network. This software download capability allows the functionality of the cable modem/RPD to be changed without requiring that MSO personnel physically revisit and reconfigure each unit. This field programmability is used to upgrade CM/RPD software to improve performance, accommodate new functions and features, correct any design deficiencies discovered in the software, and to allow a migration path as the DOCSIS technology evolves. This paper proposes taking the same secure software download methods defined in the DOCSIS specifications and reusing them for downloading machine learning models.

The CM today implements a TFTP client and alternatively also implements an HTTP client compliant with for software file downloads. The transfer is SNMP-initiated, as described in [DOCSIS OSSIv3.0], or configuration file-initiated. This mechanism can be extended to download machine learning models as well. The CM/RPD verifies that the downloaded image(ML Model) is appropriate for itself. If the image is appropriate, the CM writes the new ML model image to non-volatile storage. Once the file transfer is completed successfully, an ML application within the CM can start using this model.

4.5. ML Model File Format

The ML Model will need to be standardized to be in the format the CM or RPD can understand and use.

The following file format is proposed for the code file, it is built using a [PKCS#7]-compliant structure, similar to the CM software image, this includes the following components: A code image; i.e., the trained ML model; A Code Verification Signature (CVS); i.e., the digital signature over the image, and lastly a Code Verification Certificate (CVC); i.e., an [X.509]-compliant certificate that is used to deliver and validate the public code verification key that will verify the signature over the code image. The DOCSIS Certificate Authority, a trusted party whose public key is already stored in the CM, signs this certificate.

4.6. Verification and Instantiation

Once the ML model has been downloaded and verified, there needs to be control mechanisms to start to use the ML Model with its specific ML inference application. The MSO will always apply a digital signature to the ML Model code file. The signature is verified with a certificate chain that extends up to the Root CA. The CM/RPD verifies the signatures with a certificate chain that extends up to the Root CA before accepting a code file. In current DOCSIS CMs/RPDs, the Root CA certificate is installed in each device as a trust anchor.

After the training phase, the MSO will take the ML training output model and build a code file, as described above. The operator can load the code file on the software download server after adding its signature and operator CVC and issuing CVC CA certificate to the code file. During the code download process, the CM will retrieve the code file from the software download server, (alternatively the RPD will get the image via the GCP process) and verify the new code image using the Root CA Certificate trust anchor before installing it. This is essentially reusing the secure software download process and the certificate infrastructure already defined in the DOCSIS specification.

4.7. Enabling applications and standardized ML model API

This architecture can be applied and used for one application or for multiple applications being instantiated on the CM/RPD. One can imagine multiple machine learning inference applications at a cable modem. For example, for one application, the modem could be analyzing RxMER values to identify ingress sources. A second application could be to identify changes in baseline latency. Another application could be looking at full band captures to identify RF issues across the whole spectrum. All of these applications could be run at the same time and be supported by machine learning models trained in the MSO cloud. The MSO cloud infrastructure has more resources to run the training process and resources to create a models specific to even that node segment. These models can then be downloaded to the modem, for use within these applications.

Each application and machine learning model that it uses, would need its own well-defined API. One parameter would be the set of input features that would need to be extracted from the raw data. These features would be the needed input to the inference model within the ML application. Once these features are passed on to the inference model, it can make a prediction or classify events or identify other labels, as per the design of the ML application. These output labels would be another parameter of the API. Perhaps different type of applications along with the data they need as input and the output data labels could be standardized. Based on the design and needs of the MSO, the ML application can raise the appropriate events back to the MSO central office either via logging or alarms or other communication processes. By having a standardized ML model API, the operator now has the flexibility of deploying the application and then changing the prediction/inference model at a later point using the architecture described here.

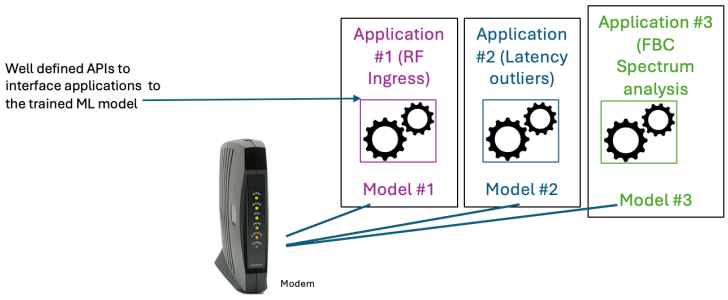

Figure 8 – Multiple ML applications can run on a edge device

5. Conclusion

Machine learning algorithms are solving many problems in the cable access network, especially in the area of proactive network maintenance and data analytics of the data coming from the cable network. Machine learning can be logically split into the training phase and the inference phase. Training an ML model requires a lot of computational resources and so will need to be run in the MSO cloud with access to lot of compute resources /GPUs. The inference (actual decision making based on a trained model) can be done at a relatively lower cost and can be suitable to run locally at a cable modem or an RPD. Given the lower latency due to the quicker access to data and performing only inference computations locally at the CM or the RPD, the MSO can enable newer applications and functionality at these devices. This ultimately provides a faster response to network events for the MSO, enabling quicker visibility into network failures or other applications.

The secure software download functionality on a cable modem and an RPD along with the certificate already installed on these devices allows for a secure and well-understood and debugged way to download new software models onto these devices. With a few additional control mechanisms to enable the download and allowing the installation of these trained models on different applications running on these edge devices, the operator can unlock ML inference at the edge. These control mechanisms may need to be standardized in the DOCSIS specifications. Having different models being able to be downloaded and used with specific applications will need a rigorous API definition between those trained models and the applications. Enabling machine learning applications at the edge of the cable access network opens up new features and functionality for the cable operator.

Abbreviations

API	application programming interface
DCA	distributed CCAP architectures
DOCSIS	data over cable system interface specification
CM	cable modem
HFC	hybrid fiber-coax
GPU	graphical processing unit
ML	machine learning
MSO	multiple system operator (network operator)
NN	neural networks
RPD	remote PHY Device
RxMER	receive modulation error ratio
RF	radio frequency
R-PHY	remote PHY

Bibliography & References

[OpenVino] OpenVino Toolkit : https://github.com/openvinotoolkit/openvino

[IntelOneDNN] Intel oneAPI Deep Neural Network Library (oneDNN) https://www.intel.com/content/www/us/en/developer/tools/oneapi/onednn.html#gs.cbj4id

[EdgeML] Microsoft, The Edge Machine Learning library: https://github.com/Microsoft/EdgeML. Dennis Don Kurian, et al., EdgeML: Machine Learning for resource-constrained edge devices

[ELL] Microsoft Embedded Learning Library (ELL) https://github.com/Microsoft/ELL

[TinyML] MIT Tiny ML Projects https://hanlab.mit.edu/projects/tinyml

[TensorFLMicro]. R. David, et al., "Tensorflow lite micro: Embedded machine learning for tinyml systems", Proceedings of Machine Learning & Systems, 2021 https://proceedings.mlsys.org/paper_files/paper/2021/hash/6c44dc73014d66ba49b28d483a8f8b0d-Abstract.html

Deploy a Machine Learning Model to a Real-Time Inference Endpoint, Tutorial, 2023 https://aws.amazon.com/tutorials/machine-learning-tutorial-deploy-model-to-real-time-inference-endpoint/

[AcData]Access Network Data Analytics, SCTE Expo 2017, Karthik Sundaresan & Jay Zhu, CableLabs

[AppML] Applications of Machine Learning in Cable Access Networks SCTE Expo 2016, Karthik Sundaresan, Nicolas Metts, Greg White, Albert Cabellos-Aparicio, CableLabs

[MLPNM] Machine Learning and Proactive Network Maintenance: Transforming Today's Plant Operations, Brady Volpe

[DetClasOFDMA] Detection and Classification of OFDMA Spectrum Impairments by Machine Learning, Jude Ferreira, et.al SCTE Expo 2023

[PNM Practices]PNM Current Methods and Practices in HFC Networks (DOCSIS® 3.1) CM-GL-PNM-3.1-V05-230927, CableLabs

[ScaleMLEdge] "Scaling Machine Learning at the Edge-Cloud: A Distributed Computing Perspective," F. Marozzo, et al., 2023 19th International Conference on Distributed Computing in Smart Systems and the Internet of Things

[ResEffML] A. Kumar, et al., "Resource-efficient machine learning in 2 kb ram for the internet of things", International Conference on Machine Learning, 2017. https://proceedings.mlr.press/v70/kumar17a.html

[ProtoNN] C. Gupta, et al., "Protonn: Com-pressed and accurate knn for resource-scarce devices", International Conference on Machine Learning, 2017, https://proceedings.mlr.press/v70/gupta17a.html

[FastGrNN] A. Kusupati, et al., "Fastgrnn: A fast accurate stable and tiny kilobyte sized gated recurrent neural network", Advances in Neural Information Processing Systems, 2018. https://arxiv.org/abs/1901.02358