Knowledge Management in Optical Networks: Architecture, Methods and Use Cases [Invited]

Marc Ruiz, Fatemehsadat Tabatabaeimehr, and Luis Velasco

***Abstract*—Autonomous network operation realized by means of control loops, where prediction from machine learning (ML) models is used as input to proactively reconfigure individual optical devices or the whole optical network, has been recently proposed to minimize human intervention. A general issue in this approach is the limited accuracy of ML models due to the lack of real data for training the models. Although the training dataset can be complemented with data from lab experiments and simulation, it is probable that once in operation, events not considered during the training phase appear thus leading into model inaccuracies. A feasible solution is to implement self-learning approaches, where model inaccuracies are used to re-train the models in the field and to spread such data for training models being used for devices of the same type in other nodes in the network. In this paper, we develop the concept of *collective self-learning* aiming at improving models error convergence time, as well as at minimizing the amount of data being shared and stored. To this end, we propose a knowledge management (KM) process and an architecture to support it. Besides knowledge usage, the KM process entails knowledge discovery, knowledge sharing, and knowledge assimilation. Specifically, knowledge sharing and assimilation are based on distributing and combining ML models, so specific methods are proposed for combining models. Two use cases are used to evaluate the proposed KM architecture and methods. Exhaustive simulation results show that model-based KM provides the best error convergence time with reduced data being shared.**

***Index Terms*—Knowledge Management; Network Automation; Autonomic Transmission; Self-learning.**

I. Introduction

THE optical network is being extended toward the edges of operators' networks [1], fostered not only by the increased amount of traffic coming from current and future access segment, but also by the stringent requirements that they need to support, like low latency and high reliability. The added complexity, in addition to highly dynamic traffic, requires the network operation to be automated. In this regard, autonomous *control loops* based on Machine Learning (ML) techniques [2] have been proposed aiming at reducing human intervention as a way to minimize network operational costs. In general, an autonomous control loop *uses knowledge* discovered during a ML training phase to predict (near) future network conditions, so as to proactively prepare resources to deal with them (*decision-making*).

Several works in the literature have focused on implementing autonomous control loops entailing knowledge usage and decision making. The authors in [3] present a predictive Autonomic Transmission Agent (ATA) based Artificial Neural Networks (ANN) that predicts the right Forward Error Correction (FEC) algorithm configuration for short-term operation as a function of real-time monitoring of state of polarization (SOP) traces and the corresponding pre-FEC BER. Note that the control loop is performed at the device level, and so the knowledge usage and the decision-making process. The authors in [4] explore several ML approaches based on Decision Trees (DT) and Support Vector Machines (SVM) for fault management [5], specifically for soft-failure detection, identification and localization taking advantage of Optical Spectrum Analyzers (OSA) to monitor the optical spectrum. Note this is a distributed system where knowledge usage is placed at the device level and decision-making is placed close to the centralized Software-defined Networking (SDN) controller. The authors in [6] demonstrate the concept of autonomic networking in disaggregated scenarios through use cases for provisioning and self-tuning based on the monitoring of optical spectrum. Note that here the control loop entails collecting monitoring data from one device and tuning the configuration of another one, so knowledge usage and decision-making need to be placed in some centralized element. Finally, the authors in [7] model origin-destination (OD) traffic at the packet layer and use the traffic prediction to proactively reconfigure the virtual network topology (VNT) to adapt it to current and predicted traffic volume and direction. Note this is a purely centralized autonomous networking control loop case where knowledge usage and decision-making are placed in close to the centralized SDN controller.

In view that knowledge usage and decision making are needed not only at the controller level, but also at the local node/subsystem level, the control plane should be designed to support such variety of use cases and scenarios of autonomous networking. For instance, the authors in [8] present the benefits of adding a Monitoring and Data Analytics (MDA) system and present operators use cases looking at automating optical network operation. Several MDA architectures are overviewed, from the centralized to alternative hierarchical ones that allow to implement control loops at different levels. In addition, the works in [9] and [10] provide more details regarding MDA architectures and its integration with other elements in the control and data planes.

Enough real data to produce accurate ML models is rarely available owing to a plethora of reasons, like the existing legal and regulatory context that limits the availability of real network performance measurement, as well as the difficulty to

Manuscript received July 6th, 2019.

Marc Ruiz, Fatemehsadat Tabatabaeimehr, and Luis Velasco (lvelasco@ac.upc.edu) are with the Optical Communications Group (GCO), at Universitat Politècnica de Catalunya (UPC), Barcelona, Spain.

obtain training datasets belonging to specific pre-commercial and commercial technologies and use them in current and forecasted scenarios. In view of that, the authors in [11] proposed a learning life-cycle to facilitate ML deployment in real operator networks. In particular, they added a ML training phase to be carried out after detecting *model inaccuracies* (e.g., in the form of prediction errors), being this the basis of *self-learning* to progressively improve the ML models deployed in the network. Such improvement can be made faster in the case of the model is being used by several agents, which can share model's inaccuracies among them; they called this as *collective self-learning*. It was demonstrated that collective self-learning outperforms individual strategies. However, because the size of the training dataset might be large to reach high-accuracy and robustness, (*data-based*) collective self-learning increases data to be stored and to be exchanged among agents.

Instead of data, ML models can also be shared among agents. An example of such model sharing can be found in [12], where the authors proposed to model OD traffic in the core as an aggregation model of the conveyed metro flows models. In this case, metro flow models are trained by the metro SDN controllers and shared with the core SDN controller, which composes the model for the core OD.

In this paper, we go further and target at completing the knowledge management (KM) process for truly autonomous optical network operation. The KM process entails creating and sharing knowledge and it has been applied to achieve organizational objectives, like continuous improvement of an organization. Those *learning organizations* are able to adapt quickly and effectively to be superior to the competitors in their field or market [13]. Here, we apply KM in the context of optical transmission and networking and define it as the process to autonomously (i.e., without human intervention) *i*) *discover*; *ii*) *share*; *iii*) *assimilate*; and *iv*) *use* knowledge to improve the performance of a network. Note that networks, like organizations, consist of a set of networking devices, which would probably not achieve a global improvement in case of knowledge being individually managed.

As discussed above, the last pillar for KM, i.e., knowledge usage, has been extensively covered in the literature. Hence, in this work we concentrate on the other three pillars and present a generic architecture for KM, methods for knowledge assimilation, and use cases in optical networks. The challenge is to develop techniques for knowledge exchange that reduce the amount of exchanged data while keeping complexity low. Specifically, the contribution of this work is three-fold:

1 Section II overviews the KM process in optical networks and proposes a general architecture to support KM. The architecture extends previous works focused on supporting control loops and on supporting individual and collective self-learning, as it includes full support for KM.

2 Aiming at remarkably reducing the amount of data to be shared (and stored), we present an alternative strategy based on sharing and combining ML models that enables *model-based* collective self-learning. Note that ML models consist of a set of parameters of moderate size compared to the size of training datasets, while capturing their knowledge. Nonetheless, that benefit might be at the cost of adding complexity in the subsequent ML model combination process. Section III presents model combination strategies to assimilate knowledge.

3 Section IV particularizes the architecture and the methods for KM for two borderline use cases, namely: *i*) the purely distributed use case for autonomic transmission problem [3], and *ii*) the purely centralized use case for pro-active VNT reconfiguration based on traffic prediction [7].

The discussion is supported by the numerical results for the defined use cases presented in Section V.

II. Knowledge Management in Optical Networks

A. KM Process Overview

Fig. 1 presents the architecture proposed to enable KM, where two software agents in charge of networking devices are represented. Agents collect monitoring/telemetry data from the underlying device(s) e.g., an optical transponder (step 1 in Fig. 1a) that are consumed by a ML-based application, to produce some output (e.g., prediction) based on some ML models regarding some device/entity, e.g., the QoT of an optical connection. The results can be used by a decision maker module (2) to tune configuration parameters in the device(s) (3). Note that we just described the typical control loop (1-2-3), which focuses exclusively on *knowledge usage*.

Now let us assume that the output produced by the ML-based application based on the measured data is stored (4) and that such output could be compared to real data measured from the device(s) after some time. If this would be possible, we could conceive an algorithm that would monitor the accuracy of the current ML models and detect events for which the models return inaccurate output (5). For illustrative purposes, Fig. 2a shows an example where a model for regression has been trained with data points. Note that those data points do not need to be uniformly distributed in the regions and can form data clusters in some regions of the *features space*, whereas no data points can be found in other regions. A prediction for data in an unknown region would produce a response value that might be far from the actual response measured from the network. Thus, detecting such inaccuracies would open the opportunity to increase our training dataset with new labeled data (i.e., $<X, y>$, where X is the input data and y the predicted response) and apply ML training to produce more accurate ML models that can be immediately used by the ML-based application (6). This loop (4-5-6) entails *knowledge discovery* and it is the base for *self-learning* [11].

As an alternative to the single ML model covering the complete features space, one could analyze the structure of the training dataset and realize of the presence of data clusters. In

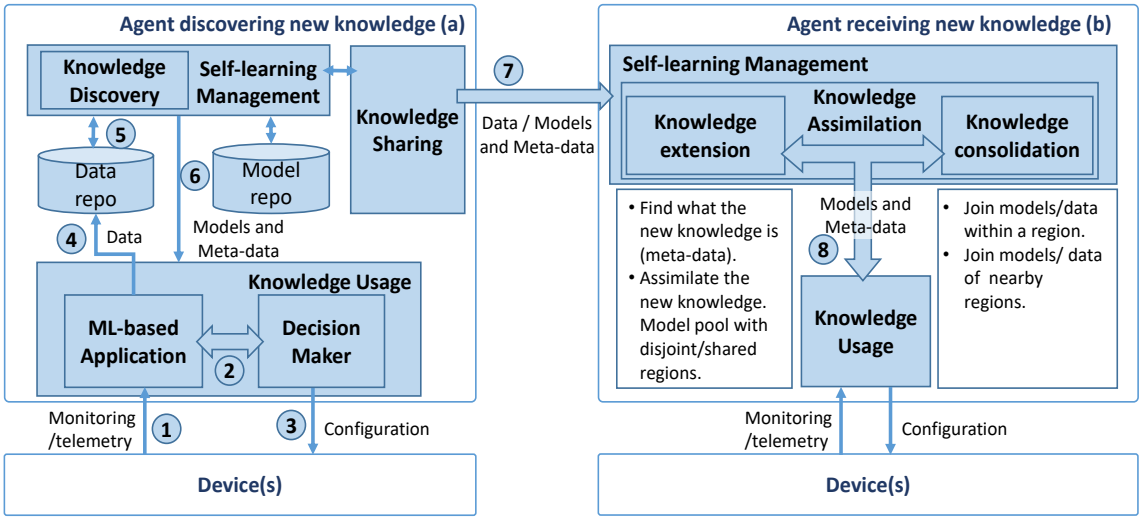

Fig. 1. KM Process. New knowledge is discovered (a) and assimilated for operation (b).

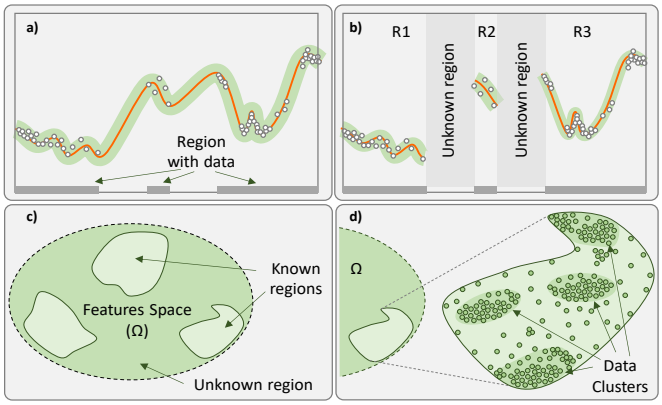

Fig. 2. Known and unknown regions in the features space.

such case, specific and more accurate ML models could be produced within each of the selected regions as it is suggested in the example in Fig. 2b (regions R1..3). In this case, some information (*meta-data*) is needed to specify the region of applicability of the model, as well as other important data, like the number of samples used to produce the model, etc. In addition, note that the lack of a model in the region of a collected measurement reveals a new unknown region; those collected data need to be stored until the corresponding label is obtained and can be used to extend the knowledge to that region.

Imagine now that the knowledge discovery process is performed individually per every different device/entity, as the measured data could be specific for such device/entity and so the corresponding ML models. In such case, knowledge discovered from one device/entity cannot be shared among different devices/entities. However, let us assume that either the measured data can be used unchanged by other devices/entities or there exists a function that normalizes the measured data (i.e., removes local dependences) so that the resulting normalized data can be used to train ML models for other devices/entities. Then, new *knowledge* in the form of labeled data can be *shared* with other agents as soon as it is discovered (7), thus enabling *collective learning* [11]. Note that the normalized data received from other agents can be used to complement the local training dataset; this increases the learning speed since the probability of rare events to be observed increases as there are more observers.

However, sharing knowledge in the form of labeled data might entail the exchange of large volumes until the accuracy of the ML models does not reach high values. Note that one single labeled data point consists of a tuple of values and that a complete training dataset can contain a large amount of data points. Another alternative to reduce the amount of data being exchanged is to produce specific models for the knowledge just discovered. These models can be very accurate in a particular region of the features space where the new knowledge has been discovered.

The components related to KM in the agent receiving the new knowledge are sketched in Fig. 1b. Note that the separation between the agent receiving the new knowledge and the one discovering it is done for illustrative purposes, as there is no limitation about being actually the same agent.

When a model and meta-data are used to share new knowledge, the receiving agent needs to *assimilate* such knowledge, starting by understanding what the new knowledge is. Assuming that the feature space is modeled in a per-region way, the received knowledge can be located (totally or partially) in one or more of the known regions or in the unknown region; in the former, the model is added to the found region(s) and a merge of regions could be performed, whereas in the latter, a new region is created. We name *knowledge extension* to the process of identifying the new knowledge and updating the regions. Note that a region can be modelled using one or more models, so region updating would entail generating a new model joining the previous model with the received one, or just adding the new model to the pool of models. Another process that we call *knowledge consolidation* is in charge of joining models within a region and joining nearby regions. Fig. 2c-d illustrate the features space of a given problem, where the training dataset contains labeled data grouped into three different regions. However, data points are not usually uniformly distributed along a region, as regions

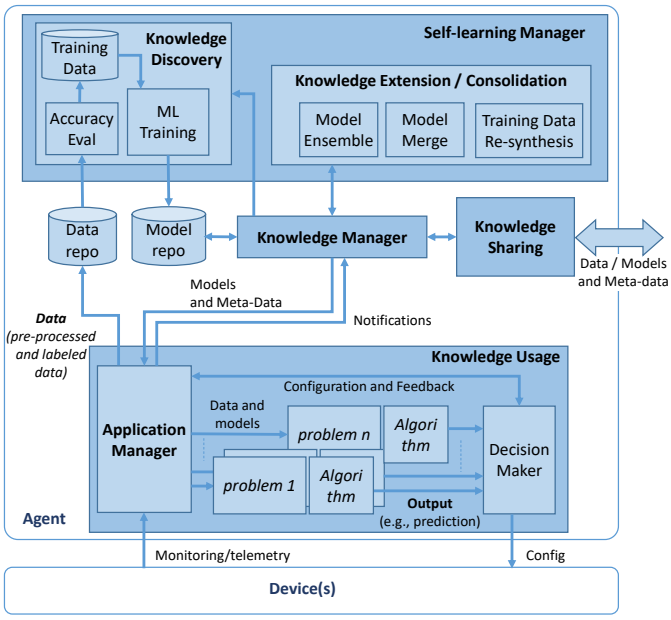

Fig. 3. Detailed architecture for KM

are dynamically re-defined as a result of a region merging process, triggered whenever new knowledge arrives.

Finally, changes in the regions and models and meta-data generate new *operational* models that are ready for knowledge usage (step 8 in Fig. 1b).

B. *Proposed Architecture*

Fig. 3 presents an extended architecture for KM, where more details of the agent are depicted; specifically, *knowledge discovery and knowledge assimilation* in the form of extension and consolidation (collectively named *self-learning*), *knowledge sharing*, and *knowledge usage* components are detailed. In addition, the *Knowledge Manager* component coordinates KM operations.

The data collected from the underlying physical device(s) is processed by an *application manager* that uses knowledge for the autonomous control of the device(s). For the sake of generalization, we consider that the configuration of the devices is based on a set of algorithms for different problems, which generate outputs to a decision maker module in charge of finding the best configuration for the forecasted conditions. Any problem might require a specific procedure combining several techniques (ML, statistics or mathematics) to generate its outputs. The role of the application manager in the device control loop is to feed the different problems with the required inputs and to adjust the decision maker according to the observed local performance.

In addition to these operational tasks, the application manager exports pre-processed and labeled data (including model predictions and real measurements) to be stored in the *data repository*. Such data is analyzed by the knowledge discovery module, which holds two essential roles: *i*) to identify inaccuracies in the current ML models and, *ii*) to populate its internal training dataset and perform ML training to produce new models that are stored in the *model repository*. The knowledge discovery loop is the main source of knowledge acquisition coming from real data from the operation of the underlying device(s). Such new knowledge can be afterwards shared with other agents through the knowledge sharing module thus, implementing collective self-learning. Consequently, knowledge discovered by other agents is also received and stored in the model repository.

The activity of knowledge discovery could lead to many ML models being stored in the repository, which would hinder knowledge usage. For example, in the case of keeping several ML models restricted to narrow region in the feature space or alternatives models for the same region. Owing to that fact, knowledge assimilation applies methods for knowledge extension and consolidation focused on reducing the number of models used for operation while keeping its overall accuracy. As illustrated in Fig. 3, we consider three different methods for such task, named *model ensemble*, *model merge*, and *training data re-synthesis*. The next section is devoted to providing the details for these assimilation methods.

Finally, following a given scheduling policy, e.g., every time a new ML model is made available or with some periodicity, the knowledge manager updates the ML models of every problem in the knowledge usage module, so the algorithms can use them for operational purposes.

Last but not least, the knowledge usage module plays a pro-active role to speed-up knowledge discovery, as the algorithm can discover that some given measured data locates into an unknown region of the features space of their problems. In such case, the application manager notifies the knowledge manager, which requests the knowledge sharing module to ask other agents about labeled data around the measured one, so as to produce a specific ML model for that unknown region.

III. Knowledge Assimilation

In this section, we describe in detail three elementary methods for assimilating knowledge in the previously described context. These options, presented in Fig. 4, are used for knowledge extension and consolidation.

For the sake of simplicity, let us assume that the agents focus on one single problem and that they are prepared to perform all type of modelling procedures including self-supervised learning. Regarding the typology of problems, let us consider both classification and regression ML-based applications; due to their properties, we selected SVM for classification and ANN for regression.

Without loss of generality, let f be a model that receives a set X of input data and provide predictions of the target response y. Input data can be monitoring data or pre-processed data after transforming monitoring data into features, whereas the response can be either a numerical value for regression, or a class for classification. A model is defined by a set f that contains, among others, the type of algorithm and/or technique that characterizes the model and the needed parameters, e.g., ANN and all the parameters and coefficients of the trained

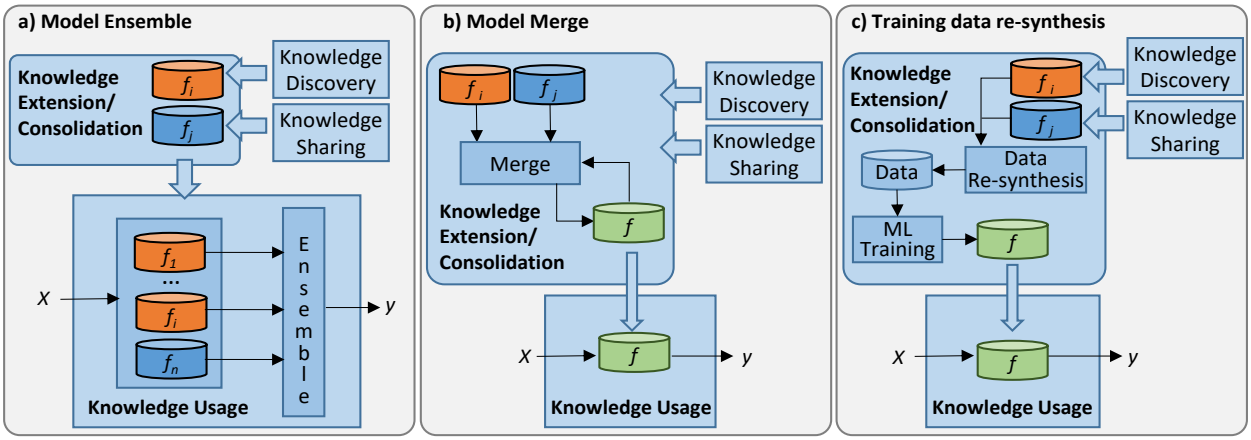

Fig. 4. Knowledge assimilation options: model ensemble (a), model merge (b), and training data re-synthesis (c).

model. In addition, the *meta-data* is coupled with the predictor and provides the context required to use properly the model. An example of meta-data is the characterization of the input features space region, i.e., the range of each feature in the training data set. Then, before doing a prediction, those ranges should be checked to know if the input data is within the ranges observed during the training phase or, on the contrary, the model will potentially extrapolate the response.

A. *Model ensemble*

This method considers no just one single ML model, but a set (ensemble) of models for a problem that e.g., correspond to different feasible scenarios that can be observed. Thus, under a specific scenario, some models will produce accurate predictions, whereas some other will produce inaccurate ones.

Under the model ensemble method, when a new model is trained, e.g., for a new scenario, it is added into the set of models used by the problem (Fig. 4a). The new model will be used according to the output algorithm to generate one single output from the predictions made by a (sub)set of individual models in the ensemble. Under this option, the algorithm is the responsible of discerning how to combine and/or select individual predictions.

The combination of individual predictions can be done according to strategies as simple as using a weighted average of the individual responses according to some meta-data parameters that serve as weights. However, the availability of monitoring data enabling the dynamic evaluation of the individual predictions allow the implementation of adaptive voting procedures that can approach predictions to actual measurements [14]. Model ensemble is an option for knowledge extension that requires low computational effort and that can be applied to any ML technique and even combine different types of ML models. A mathematical description for both classification and regression applications is provided next.

Let $E=\langle f_1, f_2, \ldots, f_n\rangle$ be the ensemble containing all available models for a given problem. Given an input data sample $X=\langle x_1, x_2, \ldots, x_m\rangle$, we define the subset of models $E^*(X) \subseteq E$ containing all the models within the region of the features space that contains X that are eligible for predicting the response of the sample. This eligibility can be computed in terms of the probability that the sample belongs to the statistical distribution of that training data used to fit the model. Then, assuming that π_i contains the characterization of the probability distribution of the input data variables of model f_i, such model can be included in E^* if and only if $P(X \mid \pi_i) > \varepsilon$, where $\varepsilon \in [0,1]$ needs to be selected beforehand. A typical conservative configuration skipping those models whose training data statistical characteristics largely differ from sample X could be $\varepsilon=0.05$ [15].

Once the ensemble subset selection has been carried out, the individual predictions y' are obtained for each model in $E^*(X)$, which are afterwards combined to produce a single combined prediction y^*. This combination is the result of applying a function that considers a weight $w_i \in \mathbb{R}^+$ for the prediction of every individual model $f_i \in E^*(X)$. In the case of classification where the response is one of the classes $c \in C$, y^* is the class of the most common response considering the weights of the models. Specifically, y^* can be computed as:

$$y^*(X) = \arg\max_{c \in C} \left\{ \sum_{f_i \in E^*(X)} w_i \cdot (y_i == c) \right\}. \quad (1)$$

In the case of regression, weighted average of the individual responses can be used, where W is the sum of the individual weights:

$$y^*(X) = \frac{1}{W} \cdot \sum_{f_i \in E^*(X)} w_i \cdot y_i \,. \quad (2)$$

Let us now focus on how the accuracy of the models can be evaluated. Let us assume that both individual and combined predictions are stored in the data repository (see Fig. 3) until the measured y is available. Then, by comparing the measured y with the individual predictions, the accuracy of each model in $E^*(X)$ can be evaluated. In particular, we define the subsets $E^*_{acc}(X)$ and $E^*_{ina}(X)$ as the accurate and inaccurate model subsets, respectively. Subset $E^*_{acc}(X)$ contains the models that produced good predictions, i.e., either those models that predicted the right class in a classification use case or those models that predicted a response within a confidence interval, e.g., 95%, in a regression use case. Note that $E^*_{ina}(X) = E^*(X) \setminus E^*_{acc}(X)$.

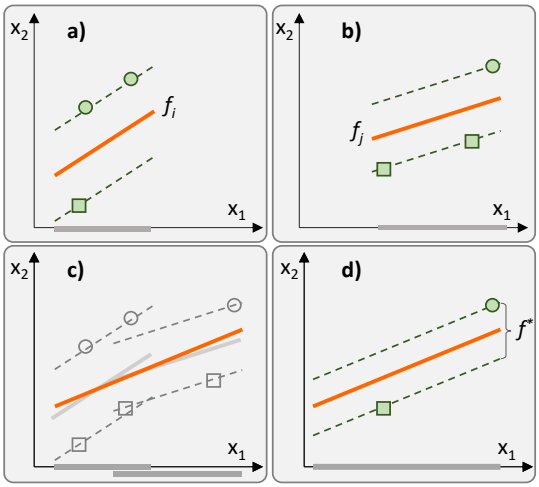

Fig. 5. Merging linear SVMs

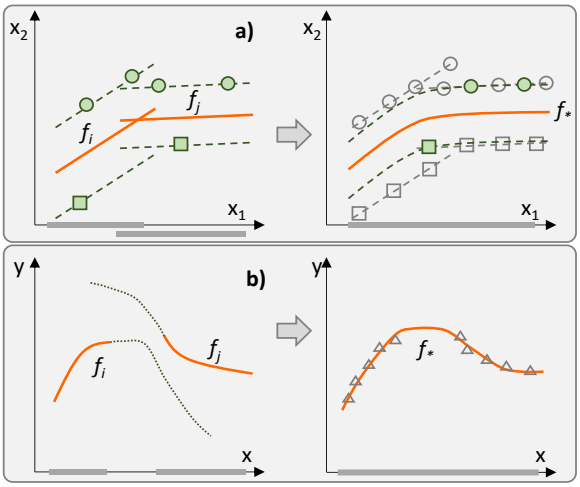

Fig. 6. Re-synthesis for classification (a) and regression (b)

By classifying the models into accurate and inaccurate, we can dynamically update the individual weights w_i used for combination purposes; minimum (w_{min}) and maximum (w_{max}) values are used to keep weights within a given range. Thus, the weight of inaccurate models can be reduced according to parameter $\rho \in [0,1]$, as:

$$w_i = \max\left(\rho \cdot w_i, w_{\min}\right), \quad \forall f_i \in E^*_{ina}(X), \tag{3}$$

whereas accurate models can be promoted by increasing its weight according to parameter $\tau \geq 1$, as:

$$w_i = \min\left(\tau \cdot w_i, w_{\max}\right), \quad \forall f_i \in E^*_{acc}(X). \tag{4}$$

Note that magnitudes and the cross-relation of ρ and τ allow configuring different strategies, ranging from a long-term persistence of past accurate models to a short-term memory configuration leading to fast changes towards current good models.

B. *Model merge*

This method consists in merging individual ML models obtaining one single model for using the knowledge, which simplifies its operation (Fig. 4b). Note that the combination of model parameters in this method is key to assimilate the individual knowledge. Parameters of the joint model can be modified by the merging procedure as soon as new models are available. This methodology can provide potential benefits for those cases where model parameters can be partially updated without affecting the robustness and accuracy of the non-updated part.

For simplicity, in this section we focus on merging a pair of individual models based on linear SVMs in the context of a binary classification problem, where two classes are linearly separable; merging n models can be defined as a concatenation of n-1 merge operations of model pairs.

Assuming that trained models f_i and f_j are linear SVMs, the coefficients of the decision hyperplanes of each model that perfectly divides the feature space region into two separated response classes can be easily obtained from the set of support vectors V_i and V_j [16]. Then, let $B_i = [\beta^0_i, \beta^1_i, \ldots, \beta^m_i]$ and $B_j = [\beta^0_j, \beta^1_j, \ldots, \beta^m_j]$ be the vector of linear coefficients (i.e., the coefficient of every feature plus the intercept) of f_i and f_j, respectively. Furthermore, in addition to meta-data π_i and π_j containing the statistical distributions of input features, the training data set size of every model (denoted as s_i and s_j) is available.

The combined model, defined by the coefficients vector B^*, can be computed using eq. (5), where the coefficients of the combined model are the weighted average of the coefficients of the individual models. Here, weights are computed by means of function $g(s)$ that depends on the number of training data samples of each model. Without loss of generality, we can assume that $g(s)$ is a simple transfer function such as the identity or the logarithm.

$$B^* = \left[\frac{\beta^k_i \cdot g(s_i) + \beta^k_j \cdot g(s_j)}{g(s_i) + g(s_j)}, \quad \forall k = 0..m \right]. \tag{5}$$

Equation (5) produces a combined model regardless of the characteristics of the individual models. However, it is worth noting that models with dissimilar characteristics can produce inaccurate combined models. A simple but efficient procedure to avoid worsening the overall accuracy is to guarantee that the combined model stays within the margin hyperplanes of both individual models. Fig. 5 illustrates the proposed procedure for a simple example with just two input features. Fig. 5a-b show two initial models to be combined, where the decision and margin hyperplanes are depicted with solid and dashed lines, respectively. Hyperplanes are depicted only in the range of the features observed for each variable; shadowed area in feature x_1 axis summarizes such range. In addition, the support vectors are depicted with markers on the corresponding margins, using a different marker shape for each class.

By solving equation (5) and assuming $g(s_i)=g(s_j)$, the combined decision hyperplane is depicted in Fig. 5c, where the original margins and support vectors are depicted; we observe in Fig. 5c that the combined decision hyperplane remains within the margin hyperplanes of the individual

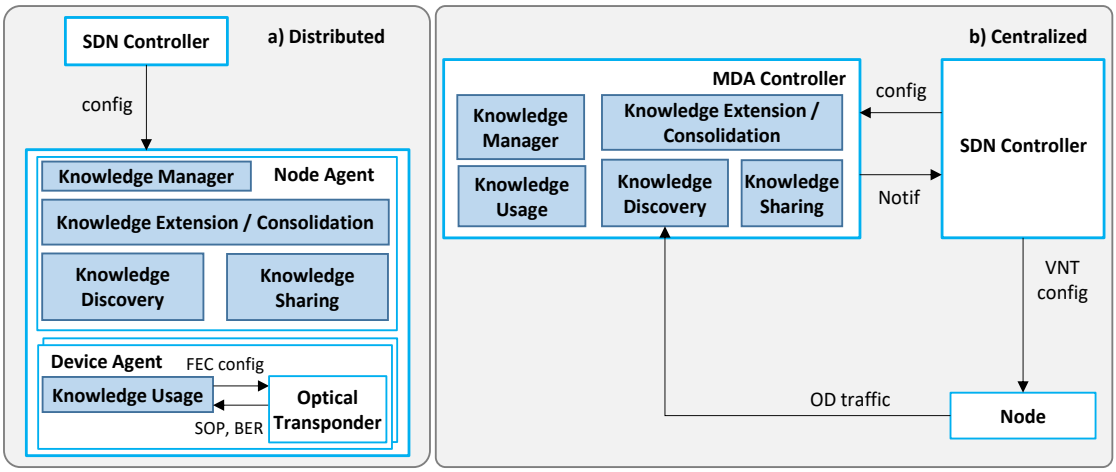

Fig. 7. KM applied to the purely distributed (a) and centralized (b) use cases

models along the corresponding feature spaces and as such, the combined model does not lead to worse decisions. In consequence, to validate the combined model, one just need to verify that combined decision hyperplane and original margins do not intersect in the regions; otherwise, model merge cannot be performed with enough goodness-of-fit assurance.

Assuming that the merged model is validated, it is important to update the new margin hyperplanes and support vectors. To keep the main properties of SVM, margins can be generated by finding those parallel hyperplanes with respect to the decision hyperplane, such that intersect with the closest support vector/s. Fig. 5d shows the combined margins and the support vectors associated to the combined model.

Finally, recall that meta-data is required also for the combined model. Particularly, the region in the features space where such model can be applied is found by computing the union of the regions of the individual models.

C. *Training data re-synthesis*

Finally, this method consists in generating the response from the individual ML models in the given regions to obtain a synthetic training dataset from which a new ML model is trained (Fig. 4c). The training data re-synthesis from ML models enables reducing the amount of data being exchanged among agents, as well as the data being locally stored.

The synthetic data generation procedure needs to consider the specifics of both the problem and the techniques for modelling, to guarantee the persistence of the characteristics of the observed data. Note that some of the shared models and/or part of the synthetic data could need to be kept for future retraining cycles.

This option can be applied to both classification and regression problems. In the case of classification using SVMs, we need to guarantee that synthetic samples are not generated inside the space defined by margin hyperplanes. Indeed, data re-synthesis should be restricted to generating samples on the margins, i.e., synthetic support vectors. Fig. 6a illustrates an example where two linear SVMs cannot be merged due to the intersection of the combined decision hyperplane with one of the margins. When the re-synthesis method is applied, a number of synthetic samples on the margins of every model are firstly generated (transparent markers) to afterwards train a new SVM. Note that the SVM training algorithm finds the best SVM configuration, including the most proper kernel. This can be easily automatized by simply training with different kernels and returning the most accurate model. In Fig. 6a, a polynomial kernel has been chosen for the combined model in order to keep separable classes, where some of the synthetic samples generated become the support vectors of the combined model (solid markers).

In the case of regression, the synthesis of data points is performed by generating random samples that fit the statistical properties of the input region of the features space of every original model, e.g., following a Montecarlo approach [17]. Then, the corresponding models are used to generate the response to label the sample. Once a significant amount of data samples has been generated for every model, the combined model is trained. Note that although in this paper we use ANN for regression, the above procedure can be applied to other techniques.

Fig. 6b shows a simplified regression problem where one single feature is used to predict the response *y*; two non-overlapping models are to be combined. Dashed lines illustrate how inaccurate each model can be when it is used for prediction using as input a data point that it is outside its region of the feature space (*extrapolation*). On the contrary, the combined model once trained from synthetically generated data samples (depicted as triangles) preserves the goodness-of-fit of both individual models.

As a conclusion, every method described in this section for knowledge assimilation has its pros and cons, which makes that the method fits better in some use cases than in others. Table I summarizes the main pros and cons of extension and consolidation methods.

IV. Use Cases

In view of Table I, this section defines two borderline use cases for illustrative purposes, where the architecture for KM

Table I. Pros and cons of knowledge assimilation methods

Extension	
• Model Ensemble	**Pros**: negligible assimilation complexity **Cons:** High storage and complex knowledge usage
Consolidation	
• Model Merge	**Pros:** Low storage and simple knowledge usage **Cons:** High assimilation complexity (algorithmic) and risk to degrade model accuracy
• Data re-synthesis	**Pros:** Simple knowledge usage **Cons:** High storage and high assimilation complexity (computational).

and the methods for knowledge sharing and assimilation presented in the previous sections are applied. The first use case uses KM in a *purely distributed* scenario, where knowledge is shared among the different network nodes, whereas the second use case uses KM in a *purely centralized* scenario, where although knowledge is shared among models, the whole KM process is entirely carried out in the MDA controller running besides the SDN controller. The use cases highlight the flexibility of the proposed architecture for KM, which can be easily adapted for different applications in multilayer network scenarios. In fact, the placement of knowledge components has been forced to fit these two borderline use cases, but it does not preclude other configurations to be feasible and even better in terms of performance. These use cases will be considered in the next section for the validation of the proposed architecture.

The architecture of the purely distributed use case is represented in Fig. 7a and is based on the autonomic transmission application in [3], where an autonomic agent running in the optical transponders collects and processes SOP and pre-FEC BER monitoring data at a rate of one sample every 278 μs, and it is able to anticipate QoT degradation caused by fiber stressing events. The prediction anticipates such degradation tens of ms before it actually happens by applying properly trained ML models; the output is used to configure the number of iterations to be performed by the error correction algorithm in the FEC module.

In this use case, it is clear the need of adopting continuous learning, justified by the impossibility to accurately train ML models to predict every possible physical fluctuation for all possible network scenarios before entering into operation. Moreover, since similar SOP fluctuations are plausible to happen in different links at different time, the benefits of sharing knowledge are expected to be high, as the relationship between SOP fluctuations and QoT in the event of gusts of wind in aerial fiber cables can be learnt in some part of the network and shared among the nodes.

In addition, knowledge usage needs to be embedded into the device agent due to the extremely high data collection rate and the need of rapid decision making and device configuration; it is a case of device-level control loop. Regarding knowledge discovery, recall that it entails analyzing predictions and real measurements to find inaccuracies (wrong classification) that could lead to training new ML models. The placement of this component cannot be done neither in the device agents because of their limited computational resources, not in the centralized SDN controller because of the large amount of data to be transferred. In this case, the node agent seems the most proper place to deploy the knowledge discovery component. Consequently, knowledge sharing is carried out among the node agents that exchange models and/or data and implement knowledge assimilation to complete the KM process.

The architecture of the purely centralized use case is represented in Fig. 7b and is based on the autonomic VNT reconfiguration in [7]. OD traffic monitoring samples are collected from the packet nodes in the network and used to predict the OD traffic expected for the next time interval, e.g. 1 hour. Traffic prediction is used to feed a VNT re-optimization problem that finds the best VNT configuration for the forecasted OD traffic matrix [18].

Here, a variety of reasons, like the continuous traffic increment, the introduction of new services with strong requirements, etc., make KM process implementation for continuous learning to be a good choice. In this use case, although different architectures could be feasible, the network-wide control loop entails that knowledge components are located in the MDA controller. Hence, monitoring traffic data can be collected at a coarse interval, e.g. 15 minutes, and analyzed in the MDA controller for dynamic VNT reconfiguration purposes. Continuous learning is needed to adapt models to traffic evolution; here an inaccuracy is defined as a prediction with error above some defined threshold. Notwithstanding the centralized architecture, knowledge sharing can be carried out among OD traffic models; here knowledge assimilation based on data exchange can be an option, in the case of enough storage is available. The selection of the subset of OD to whom share knowledge is also important in the case of ODs can be classified as a function of the type of traffic they convey.

Finally, note that in both use cases, the SDN controller should be in charge of setting the proper configuration parameters and policies for the KM process. In particular, policies should specify *what*, *when*, *how*, and to *whom* knowledge needs to be shared, *when* knowledge assimilation should be carried out, etc.

V. Results

In this section, we first introduce the simulation environment used for performance evaluation and define the specifics of the two selected use cases. Next, we study and compare the performance from applying KM and start by considering KM based on data exchange, where data related to the detected inaccuracies is distributed, as well as based on model exchange, where the knowledge assimilation techniques presented in Section III are applied.

A. Simulation Environment and Use Cases

For performance evaluation of the proposed KM process, a simulation environment has been developed in R. A network

consisting of a number of nodes, each composed of several devices, and connected by a set of links is reproduced. Specifically, we configured a scenario reproducing a small-size metro network consisting in 10 locations, where each location consists of both a packet node and an optical node each equipped with 10 ports.

Initial datasets for each use case were generated based on the topology characteristics and end-users information from [19] and initial ML models for each device were trained. Each device includes a data generator to synthetize monitoring data for the target use case. Operation was emulated by generating synthetic monitoring samples that include events that were not observed during the initial ML training phase, so new knowledge is discovered.

In the case of the purely distributed autonomic transmission use case, devices emulate optical receivers and generate synthetic monitoring samples at a rate of 278 µs (3600 samples/s). Each sample consists of a 42-byte tuple <*t*, *S*, *BER*>, where *t* is the timestamp, *S* is the set of values of the three Stokes parameters, and *BER* is the pre-FEC BER measurement. Realistic fiber stressing events causing correlated SOP and pre-FEC BER fluctuations were randomly generated based on the experimental measurements carried out in [3]. For this use case, we considered SVMs to predict the proper configuration of the FEC module (i.e., number of FEC iterations) as a function of pre-computed features gathering the current value and trend of each of the Stoke parameters. Note that those features can be easily pre-computed from the generated synthetic monitoring data [3]. Finally, an inaccuracy is defined as a misclassification, i.e., the model predicts a wrong number of FEC iterations.

For the purely centralized autonomic VNT reconfiguration use case, devices emulate network interfaces in packet nodes. We used the CURSA-SQ methodology in [20] to generate realistic packet traffic flow samples with granularity 15 minutes, emulating the monitoring data collected from those interfaces. Each sample consists of a 64-byte tuple <*t*, *OD*, *B*>, where *t* is the timestamp, *OD* is a string identifying the OD flow, and *B* is the bitrate measurement in b/s. OD traffic is predicted using ANNs whose inputs are the measurements in the last hour and the number of hidden neurons equals to the number of inputs, in line with the modelling approach presented in [7]. Here, an inaccuracy is defined as a prediction for which the magnitude of the error for a real measurement is greater than the percentile 95% of the error observed during the training phase.

The simulation environment follows the KM architecture proposed in Fig. 3, where the different KM components can be placed in node agents and/or the MDA controller to compose the distributed and centralized scenarios presented in Fig. 7, as well as any other intermediate configuration. Moreover, the configuration of the policies for knowledge discovery, assimilation, and sharing can be configured from the SDN controller. Finally, the MDA controller collects relevant network performance evaluation data, including the evolution of the accuracy of the models and the amount of shared data.

B. Data-based Knowledge Management

Let us first evaluate the performance of KM based on sharing data. We assume that inaccuracies are shared when

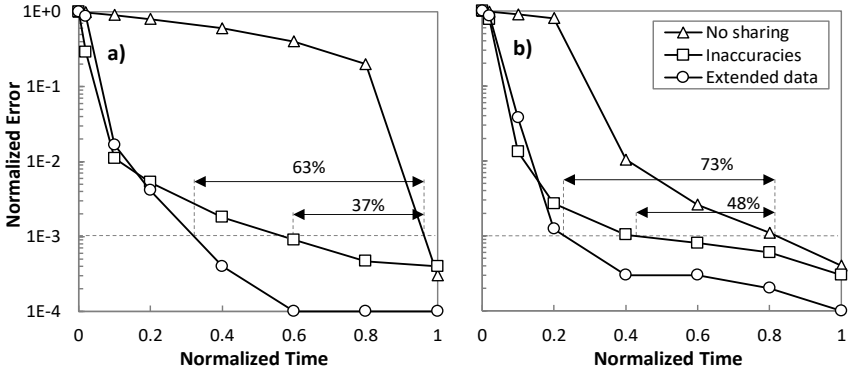

Fig. 8. Data-based KM performance for the distributed (a) and centralized (b) use cases

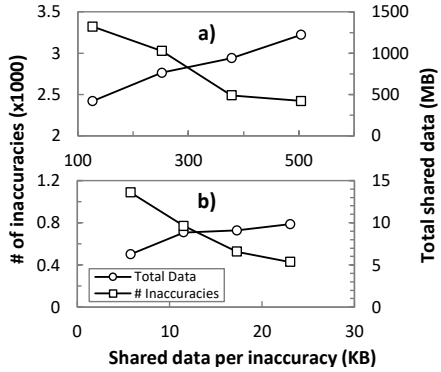

Fig. 9. Extended data policy analysis

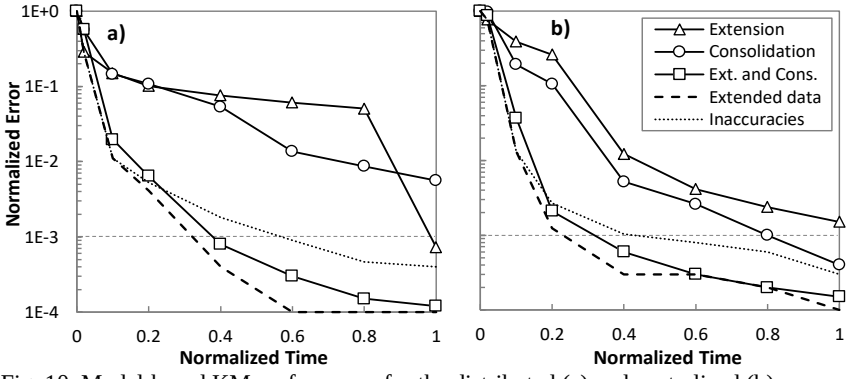

Fig. 10. Model-based KM performance for the distributed (a) and centralized (b) use cases

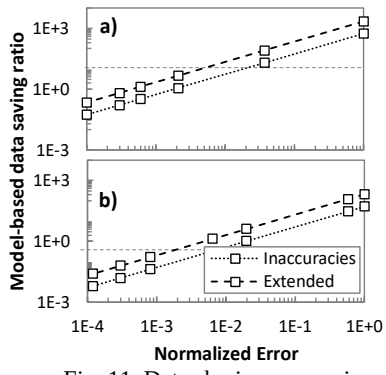

Fig. 11. Data sharing comparison

Table II. Convergence Time Gain w.r.t No Sharing (%)

	Multiplier of shared data per inaccuracy			
	x1	**x2**	**x3**	**x4**
Distributed	36.8	45.5	61.25	63.2
Centralized	47.5	66.25	71.3	72.5

Table III. Total amount of shared data (in MB)

Use case	**Model-based**	**Min data reduction**	**Data-based (inaccuracies)**	**Data-based (extended)**
Distributed	2.6	99%	333.4	1333.5
Centralized	1.5	90%	15.2	60.9

they are detected. Specifically, we consider two different policies for data sharing: *i*) *inaccuracies*, where inaccurate data points are shared. Specifically, we consider that a small window of samples (e.g., 30 samples) is needed to be shared to compute the features for the inaccuracy in the case of the purely distributed autonomic transmission, whereas just one sample is needed in the case of the purely centralized VNT reconfiguration. Note that this policy is adapted from the collective self-learning approach presented in [11]; and *ii*) *extended data*, where inaccurate data points go hand in hand with other data points that, although they were not been identified as inaccuracies, could be potentially useful to improve ML models. Although other options could be considered for selecting such additional data points, an extended window to allow compute the evolution of the features is shared in the case of autonomic transmission, whereas individual samples measured immediately before the inaccuracy are shared in the case of VNT reconfiguration. The amount of additional data points that provides the best trade-off between accuracy and data volume depends of the use case and scenario and it will be analyzed. Finally, ML model re-training is carried out periodically, e.g., every hour, provided that inaccurate data points are available.

Fig. 8 shows the performance of the proposed data-based KM in terms of the evolution of the prediction error against emulated operation time, for both the purely distributed autonomic transmission (Fig. 8a) and the purely centralized VNT reconfiguration (Fig. 8b) use cases. For benchmarking purposes, we included the performance of no sharing knowledge. For convenience, prediction error has been normalized to the error of the initial models, whereas operation time was normalized to the time when the less accurate approach reaches a low target error (e.g., 0.1%). Interestingly, the results show similar behavior for both use cases, where large benefits from knowledge sharing are observed. In particular, extended data sharing shows a better convergence time, reaching the target error 60-70% faster than without sharing knowledge. Moreover, is that policy the only one that achieves negligible errors around 0.01%. The inaccuracies sharing policy shows also excellent performance and although its convergence time is above than that of the extended data sharing one, it is over 35% faster than no sharing knowledge. In fact, both data sharing policies show a similar error evolution until reaching error around 3-4%, which makes that the policy selection needs to be based on other criterion in case the target error criterion can be relaxed.

In fact, particular interest should be payed to the amount of total data that is shared. This criterion is relevant mainly for the purely distributed use case, as such data is exchanged among agents that are not in the same location. Table II shows the gain in terms of convergence time as a function of the data shared per inaccuracy for the purely distributed and the purely centralized use case. The multiplier refers to the amount of additional data that is shared, where x1 is equivalent to the inaccuracies sharing policy. The amount of additional data that needs to be exchanged to achieve the gains showed in Fig. 8 represents an increment of 3 times (x4) the amount of data exchanged with the inaccuracies sharing policy.

Fig. 9a and Fig. 9b show the number of inaccuracies and the total data volume shared during the entire simulation as a function of the amount of data exchanged per inaccuracy for the purely distributed and the purely centralized use case, respectively. The evolution of the total number of inaccuracies shows how they are reduced when the amount of extended data is increased (about 1/3 in the case of the distributed and 60% in the case of the centralized use case). Such reduction is the base of the achieved convergence gain. Regarding the amount of total data shared, although acceptable for the purely centralized use case, it is above 1 GB for the purely distributed one. Recall that every accuracy entails 30*42 bytes in the case of autonomic transmission, and 64 bytes in the case of VNT reconfiguration to be shared with (10*10+9) agents. Even with the reduction of the number of inaccuracies, the volume of exchanged data is high under the extended data policy.

In view of these results, and considering that the probability of discovering inaccuracies decreases with time, a mixed data-based approach can be followed; the inaccuracy sharing policy can be first applied to allow an initial fast convergence with a reasonable amount of data being shared, followed by the extended data sharing policy after reaching a certain error level to increase even more models' accuracy.

C. Model-based Knowledge Management

Let us now explore policies based on sharing models and knowledge assimilation by means of the methods proposed in Section III. Recall that, in addition to the models, meta-data is needed to specify their region of applicability; specifically, we limit meta-data to specify the range (minimum and maximum) of each input feature. For the ongoing analysis, we assume that the model ensemble method is configured with a short-term memory tuning. Specifically, the following configuration was chosen: ρ=0.6, τ=1.5, w_{min}=1, w_{max} =10. Regarding model merge and training data re-synthesis, we used them according to the characteristics of the ML techniques used for the purely distributed (SVM) and purely centralized (ANN) use cases, respectively.

Aiming at evaluating the performance of different policies and the impact of the main blocks involved in knowledge

assimilation, i.e., knowledge extension and consolidation, we compare three basic policies: *i*) *extension*, where every new shared model is added to a device models pool and used together with the model ensemble method, without any consolidation action; *ii*) *consolidation*, where just a single model is maintained, i.e., incoming shared models update the model by either model merge or training data re-synthesis methods, depending on the use case; *iii*) *extension and consolidation*, where both knowledge extension and consolidation is continuously performed to keep moderated the size of the models' pool (we limited its size to 10 models). Meta-data is used to join models within a region or, if necessary, in nearby regions of the features space.

Fig. 10 shows the evolution of model error against time for the above model-based policies and the defined use cases. For the sake of comparison, we included the two data-based KM policies previously analyzed in Fig. 8. The results show that the policy combining knowledge extension and consolidation achieves a performance comparable to that of the data-based KM thus, validating in terms of accuracy a KM process based on sharing models instead of monitoring and pre-processed data. The other two policies show worse performance and lead to either an increasing number of models, which makes difficult a practical operation, because of the large number of models, and reduces the potential of incremental learning, or to a forced consolidation, which combines models with dissimilar characteristics in different regions, which increases errors that reduce the gain obtained by the acquired new knowledge (see Section III.B).

Once the excellent performance of the model-based KM with the policy combining knowledge extension and consolidation has been demonstrated, its practical applicability depends mainly on the amount of data involved in knowledge sharing, as compared to data-based KM policies. Fig. 11a and Fig. 11b show the evolution of the ratio between the data shared by each data-based policy and the combined policy of the model-based one as a function of model errors for the distributed and centralized use cases, respectively. Ratio equal to 1 (highlighted as a dashed line) represents the case where data-based and model-based policies exchange the same amount of data, whereas when the ratio is lower than (higher than) one entails data-based (model-based) policy exchanging less data. As it can be observed, data-based policies provide benefits in terms of exchanged data only when very low error are achieved. In the rest of cases, model-based KM reduces the amount of shared data several orders of magnitude in both the distributed and centralized use cases. Table III complements Fig. 11 and presents the total amount of data exchanged at the end of simulations by each of the policies for each of the use cases.

As a conclusion, the combined knowledge extension and consolidation policy of model-based KM provides virtually the best performance and it is the most scalable option by far. Nevertheless, one can combine different policies by selecting the one that better fits the current scenario. In particular, the selection of data-based and model-based policies at different times of the KM process as a function of model's accuracy could provide the best performance. This is highlighted in Fig. 12a and Fig. 12b, where a mixed strategy combining data-based and model-based policies are compared in terms of the total amount of shared data for the distributed and centralized use case, respectively. According to the performance results in Fig. 10, the mixed policy providing the optimal performance would consists of the model-based policy for the initial phase until models reach error around 1%, followed by the data-based inaccuracies policy, until the error reaches around 0.1% and complemented by the extended data-sharing policy to reach a negligible error around 0.01%. As it can be observed, the mixed policy allows reducing even more data volumes involved during knowledge sharing.

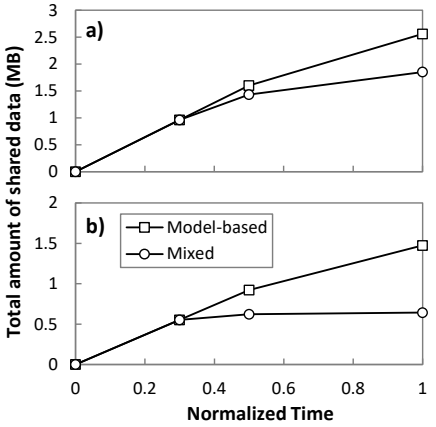

Fig. 12. Model-based and Mixed knowledge sharing

VI. Concluding Remarks

The Knowledge Management (KM) process has been proposed aiming at a truly autonomous optical network operation. KM is based on four main pillars: *i*) knowledge discover; *ii*) knowledge share; *iii*) knowledge assimilate; and *iv*) knowledge usage. These pillars allow optical networks to autonomously discover and disseminate knowledge that can be used to adapt its configuration to variable conditions without human intervention.

A general architecture to support KM has been proposed that extend beyond typical control loop implementation and allows for knowledge sharing among different agents disregarding they run distributed in the network nodes or centralized in a controller, like the Monitoring and Data Analytics (MDA) one. Such knowledge sharing enables collective self-learning, which has been demonstrated to reduce models error convergence time.

However, knowledge sharing entails data distribution and storage and hence, keeping limited the amount of data is a key issue. In that regard, two alternative strategies consisting on the distribution of data samples related to model inaccuracies (data-based) and models representing such inaccuracies (model-based) are studied. For the latter strategy, three methods for knowledge assimilation are proposed: *i*) model ensemble, *ii*) model merge, and *iii*) training data re-synthesis.

With these methods, knowledge assimilation can be implemented by means of two main actions to manage ML models: extension and consolidation. Such actions are carried out in the knowledge assimilation component in the architecture. In particular model ensemble, allows an efficient and accurate use of ML model pools, model merge allows combining the coefficients of different models to produce a combined model and training data re-synthesis allows to consolidate different models based on regenerating data from them that are used to train new models.

Two illustrative use cases have been used to illustrate the potential application of the KM architecture and to evaluate different policies for knowledge sharing and assimilation: *i*) the purely distributed autonomic transmission use case, where knowledge is used at the optical transponder system level and knowledge sharing and assimilation is carried out at the node level; and *ii*) the purely centralized VNT reconfiguration use case, where all the components run at the MDA controller level. Note that even in this case, a different model is kept for every of the origin-destination traffic flows in the VNT, so knowledge sharing and assimilation takes also place.

The KM process has been evaluated by simulation on a metro network scenario for the defined use cases in terms of model error convergence time and amount of data shared among agents. Two different data-based policies were studied and concluded that sharing data inaccuracies and retraining ML models leads to a fast error convergence time until reaching a certain low error, where error can be reduced even more when additional (extended) data was shared along with the inaccuracies. In addition, a model-based policy based on applying coordinated extension and consolidation actions demonstrated similar convergence time than data-based policies with few orders of magnitude less of data being shared among agents. Indeed, the combination of the three data-based and model-based policies at different phases of the network learning process reached minimal shared data volumes without compromising the convergence towards highly accurate models.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Commission for the H2020-ICT-2016-2 METRO-HAUL project (G.A. 761727), from the AEI/FEDER TWINS project (TEC2017-90097-R), and from the Catalan Institution for Research and Advanced Studies (ICREA).

REFERENCES

[1] L. Velasco, P. Wright, A. Lord, and G. Junyent, “Saving CAPEX by Extending Flexgrid-based Core Optical Networks towards the Edges,” IEEE/OSA Journal of Optical Communications and Networking, vol. 5, pp. A171-A183, 2013.

[2] D. Rafique and L. Velasco, “Machine Learning for Optical Network Automation: Overview, Architecture and Applications,” IEEE/OSA Journal of Optical Communications and Networking, vol. 10, pp. D126-D143, 2018.

[3] M. Ruiz, F. Boitier, P. Layec, and L. Velasco, “Self-Learning Approaches for Real Optical Networks,” in Proc. IEEE/OSA Optical Fiber Communication Conference (OFC), 2019.

[4] B. Shariati, M. Ruiz, J. Comellas, and L. Velasco, “Learning from the Optical Spectrum: Failure Detection and Identification [Invited],” IEEE/OSA Journal of Lightwave Technology, vol. 37, pp. 433-440, 2019.

[5] L. Velasco and D. Rafique, “Fault Management Based on Machine Learning [Invited],” in Proc. IEEE/OSA Optical Fiber Communication Conference (OFC), 2019.

[6] L. Velasco, A. Sgambelluri, R. Casellas, Ll. Gifre, J.-L. Izquierdo-Zaragoza, F. Fresi, F. Paolucci, R. Martínez, and E. Riccardi, “Building Autonomic Optical Whitebox-based Networks,” IEEE/OSA Journal of Lightwave Technology, vol. 36, pp. 3097-3104, 2018.

[7] F. Morales, M. Ruiz, Ll. Gifre, L. M. Contreras, V. López, and L. Velasco, “Virtual Network Topology Adaptability based on Data Analytics for Traffic Prediction,” IEEE/OSA Journal of Optical Communications and Networking (JOCN), vol. 9, pp. A35-A45, 2017.

[8] L. Velasco, A. Chiadò Piat, O. González, A. Lord, A. Napoli, P. Layec, D. Rafique, A. D'Errico, D. King, M. Ruiz, F. Cugini, and R. Casellas, “Monitoring and Data Analytics for Optical Networking: Benefits, Architectures, and Use Cases,” accepted in IEEE Network Magazine (DOI: 10.1109/MNET.2019.1800341), 2019.

[9] Ll. Gifre, J.-L. Izquierdo-Zaragoza, M. Ruiz, and L. Velasco, “Autonomic Disaggregated Multilayer Networking,” IEEE/OSA Journal of Optical Communications and Networking, vol. 10, pp. 482-492, 2018.

[10] L. Velasco, Ll. Gifre, J.-L. Izquierdo-Zaragoza, F. Paolucci, A. P. Vela, A. Sgambelluri, M. Ruiz, and F. Cugini, “An Architecture to Support Autonomic Slice Networking [Invited],” IEEE/OSA Journal of Lightwave Technology, vol. 36, pp. 135-141, 2018.

[11] L. Velasco, B. Shariati, F. Boitier, P. Layec, and M. Ruiz, “A Learning Life-Cycle to Speed-up Autonomic Optical Transmission and Networking Adoption,” IEEE/OSA Journal of Optical Communications and Networking, vol. 11, pp. 226-237, 2019.

[12] F. Morales, Ll. Gifre, F. Paolucci, M. Ruiz, F. Cugini, P. Castoldi, and L. Velasco, “Dynamic Core VNT Adaptability based on Predictive Metro-Flow Traffic Models,” IEEE/OSA Journal of Optical Communications and Networking (JOCN), vol. 9, pp. 1202-1211, 2017.

[13] P. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization*, Doubleday/Currency, 1990.

[14] T.G. Dietterich, “Ensemble Methods in Machine Learning,” in Proc. Multiple Classifier Systems (MSC), Lecture Notes in Computer Science, vol 1857, 2000.

[15] A. C. Rencher, *Multivariate Statistical Inference and Applications*, Wiley, 1st ed., 1997.

[16] B. Scholkopf and A. L. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, MIT Press Cambridge, 2001.

[17] D. P. Kroese, T. Taimre, and Z. Botev, *Handbook of Monte Carlo Methods,* Wiley, Wiley, 1st ed., 2011.

[18] L. Velasco and M. Ruiz, *Provisioning, Recovery and In-operation Planning in Elastic Optical Networks*, Wiley, 1st ed., 2017.

[19] METRO-HAUL project, “Deliverable D3.1: Selection of metro node architectures and optical technologies,” [on-line: https://metro-haul.eu], 2018.

[20] M. Ruiz, F. Coltraro, and L. Velasco, “CURSA-SQ: A Methodology for Service-Centric Traffic Flow Analysis,” IEEE/OSA Journal of Optical Communications and Networking (JOCN), vol. 10, pp. 773-784, 2018.